

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ ГРИ У ЖАНРІ TOWER DEFENSE
DEVELOPMENT OF A COMPUTER GAME IN THE TOWER DEFENSE
GENRE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220

Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

КОЛОКОЛ Я.О.

(ПРИЗВИЩЕ та ініціали)

Керівник КАПЛІЄНКО Т.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОЛОКОЛА Ярослава Олексійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної гри у жанрі Tower Defense.
Development of a Computer Game in the Tower Defense Genre

керівник проєкту (роботи) к.т.н., доцент, КАПЛІЄНКО Тетяна Ігорівна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “24” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис програми. 3. Розробка програми.
4. Експлуатація програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КАПЛІЄНКО Т.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2 тиждень	Розділ 1
3	Опис програми.	3 тиждень	Розділ 2
4	Розробка програми.	4 тиждень	Розділ 3
5	Експлуатація програми.	5 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
7	Нормоконтроль та рецензування.	7 тиждень	
8	Захист роботи.	8 тиждень	

Студент(ка)

(підпис)

Ярослав КОЛОКОЛ

(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Тетяна КАПЛІЄНКО

(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 226 с., 5 табл., 50 рис., 2 дод., 25 джерел.

2D ГРА, БІБЛІОТЕКА, СЕРЕДОВИЩЕ РОЗРОБКИ, API, C++, OPENGL, TOWER DEFENSE, UI.

Об'єкт дослідження – процес розробки комп'ютерної 2D гри.

Предмет дослідження – сучасні методи та технології розробки ігрових застосунків.

Мета роботи – програмна реалізація комп'ютерної 2D гри у жанрі Tower Defense, що містить базові механіки та простий рівень складності.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування C++, компілятор g++, текстовий редактор Notepad++, система будування програми make, бібліотека SDL для графіки та анімації, бібліотека Box2d для фізичних симуляцій, бібліотека GLM для математичних операцій, бібліотека SQLite для зберігання даних, персональний комп'ютер з процесором AMD Ryzen 7 5700U під управлінням операційної системи Microsoft Windows 10, ОЗУ 8ГБ.

Результати. Розроблено комп'ютерну гру у жанрі Tower Defense з високоякісною графікою, фізичною симуляцією об'єктів та системою зберігання даних.

Висновки. Був отриманий цінний досвід щодо розробки комп'ютерної гри з використанням мови програмування C++.

Галузь використання – галузь розважальних ігор, освіти та досліджень у сфері комп'ютерних ігор.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 226 pages, 5 tables, 50 figures, 2 appendixes, 25 sources.

2D GAME, LIBRARY, DEVELOPMENT ENVIRONMENT, API, C++, OPENGL, TOWER DEFENSE, UI.

The object of research is the process of developing a 2D computer game.

The subject of research is modern methods and technologies of game application development.

The goal of the work is the software implementation of a 2D computer game in the Tower Defense genre, containing basic mechanics and a simple level of difficulty.

Materials, methods and tools: object-oriented programming, C++ programming language, g++ compiler, Notepad++ text editor, make program building system, SDL library for graphics and animation, Box2d library for physical simulations, GLM library for mathematical operations, SQLite library for data storage, personal computer with an AMD Ryzen 7 5700U processor running the Microsoft Windows 10 operating system, 8GB of RAM.

The results. A computer game in the genre of Tower Defense with high-quality graphics, physical simulation of objects and a data storage system has been developed.

Conclusions. Valuable experience was gained in developing a computer game using the C++ programming language.

The field of use is the field of entertainment games, education and research in the field of computer games.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК	9
ВСТУП	10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	12
1.1 Основні поняття і визначення.....	12
1.2 Огляд існуючих патернів.....	14
1.2.1 Game Object.....	14
1.2.2 Component	16
1.2.3 Entity Component System.....	19
1.2.4 Custom System.....	22
1.3 Порівняльний аналіз існуючих патернів	23
1.4 Огляд існуючих аналогів.....	25
1.4.1 Mindustry	25
1.4.2 Bloons Tower Defense	27
1.4.3 Plants vs. Zombies.....	28
1.4.4 Thronefall	30
1.5 Порівняльний аналіз існуючих аналогів	31
1.6 Постановка завдання до роботи	32
1.7 Висновок	33
2 ОПИС ПРОГРАМИ	34
2.1 Вибір мови програмування	34
2.1.1 C++	34
2.1.2 C#	37
2.1.3 Python.....	38
2.2 Порівняння мов програмування	39
2.3 Вибір середовища розробки	42
2.3.1 Visual Studio	42
2.3.2 CLion.....	44
2.3.3 Custom IDE	45

2.4 Порівняння середовищ розробки	46
2.5 Вибір мультимедійної бібліотеки	49
2.5.1 SDL.....	49
2.5.2 SFML.....	50
2.5.3 Allegro	52
2.6 Порівняння мультимедійних бібліотек	53
2.7 Структурна схема ігрового циклу	55
2.8 Висновок	58
3 РОЗРОБКА ПРОГРАМИ	59
3.1 Структура проєкту	59
3.1.1 Загальна структура	59
3.1.2 Процес створення програмного забезпечення	61
3.2 Структура програмного забезпечення	62
3.2.1 Основні елементи	62
3.2.2 Компоненти	63
3.2.3 Головне меню.....	65
3.2.4 Захист веж	68
3.2.5 Вежі.....	71
3.3 Інтерфейс	73
3.3.1 Рендеринг	73
3.3.2 Анімація.....	75
3.3.3 Адаптація.....	78
3.4 Висновок	79
4 ЕКСПЛУАТАЦІЯ ПРОГРАМИ	80
4.1 Інструкція щодо запуску гри	80
4.2 Системні вимоги	80
4.3 Функціонал системи	81
4.4 Опис застосування програми.....	81
4.5 Висновок.....	88
ВИСНОВКИ.....	89

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	90
ДОДАТОК А Текст програми.....	92
ДОДАТОК Б Слайди презентації.....	219

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

GLM – OpenGL Mathematics;

HP – Health Point;

IDE – Integrated Development Environment;

NPC – Non-playable Character;

RAM – Random Access Memory;

SDL – Simple DirectMedia Layer;

TD – Tower Defense;

UI – User Interface;

БД – база даних.

ВСТУП

У сучасному світі інформаційних технологій відеоігри займають особливе місце серед розважальних форматів, відкриваючи безмежні можливості для творчості та розвитку. Один із таких популярних жанрів, який здобуває все більше шанувальників - це Tower Defense. Цей жанр відзначається стратегічним підходом та акцентом на оборонну тактику, пропонуючи гравцям захищати свою базу від хвиль ворожих атак. Він не тільки надає можливість користувачам насолоджуватися грою, але й стимулює їх стратегічне мислення та планування.

Розробка комп'ютерної гри у жанрі Tower Defense викликає особливий інтерес серед розробників та гравців. Цей процес має важливе практичне значення для індустрії розважального програмного забезпечення, адже від його результатів залежить якість та популярність гри. Крім того, розробка ігор у цьому жанрі може стати платформою для впровадження нових ідей та технологій, що розширить можливості галузі у нових напрямках.

Актуальність теми дослідження визначається не лише популярністю Tower Defense серед гравців, але й можливістю використання їх як навчального інструменту для розвитку логічного мислення та стратегічного планування. Зростання інтересу до ігор у жанрі Tower Defense створює потребу в подальшій розробці та вдосконаленні геймплею.

Популярність Tower Defense ігор швидко зростає, особливо серед широкого кола гравців, які шукають захопливий та випробувальний досвід гри. Це створює підґрунтя для подальшого дослідження та розробки нових ігор у цьому жанрі.

Об'єктом дослідження є процес розробки комп'ютерної гри у жанрі Tower Defense, а предметом - методи та інструменти програмування, які використовуються для створення таких ігор.

Мета даної дипломної роботи полягає у розробці комп'ютерної гри у жанрі Tower Defense з використанням сучасних технологій програмування,

бібліотек та методів геймдизайну.

Для досягнення поставленої мети у роботі визначаються наступні завдання:

- провести огляд стану сучасного розвитку ігор у жанрі Tower Defense;
- дослідити існуючі аналоги комп'ютерних ігор жанру Tower Defense та виявити їхні переваги та недоліки;
- розробити архітектуру гри та алгоритми її геймплею;
- реалізувати комп'ютерну гру на базі розробленої архітектури;
- провести аналіз результатів та зробити висновки щодо ефективності розробленої гри.

Методика дослідження буде включати аналіз літературних джерел, розробку програмного забезпечення за допомогою сучасних інструментів програмування та бібліотек, його тестування та оцінку ефективності.

Дана робота розкриє сутність та перспективи розвитку ігор у жанрі Tower Defense, сприяючи подальшому збагаченню ігрового ринку та розвитку комп'ютерної індустрії в цілому.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття і визначення

Відеогра - це комп'ютерна гра, в якій гравець використовує інтерфейс користувача для взаємодії з відеопристроєм та отримання зворотної інформації. Електронні пристрої, що використовуються для гри, відомі як ігрові платформи, такі як особистий комп'ютер або ігрова консоль. Для управління грою використовується ігровий контролер, такий як джойстик, клавіатура з мишею, геймпад або сенсорний екран [1].

Tower defense (TD) - це піджанр стратегічних ігор, в якому гравець має захищати свої території або пункти входу, заважаючи атакам ворогів або не дозволяючи їм дістатися до вихідних точок. Зазвичай це досягається шляхом розміщення захисних споруд вздовж маршруту нападу. Це може включати будівництво різноманітних структур, які автоматично блокують, заважають, атакують або знищують ворогів. Жанр Tower defense розглядається як піджанр стратегічних відеоігор у реальному часі через те, що основні дії відбуваються в режимі реального часу, але багато сучасних ігор у цьому жанрі включають аспекти покрокової стратегії. Важливим аспектом цього жанру є стратегічний вибір і розташування захисних елементів. Приклад простої гри у жанрі Tower Defense наведено на рисунку 1.1 [2].

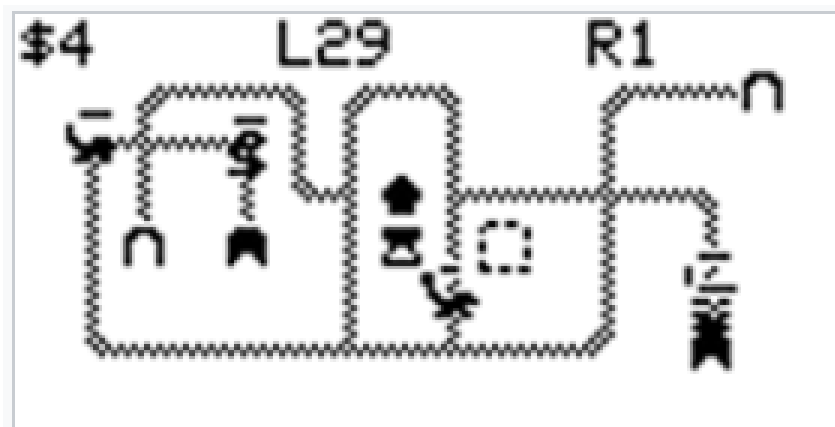


Рисунок 1.1 - Проста 1-розрядна TD гра для Arduboy [2]

Основні елементи ігрового процесу Tower Defence:

- території, володіння, місця проживання, які повинен захищати гравець або гравці;
- хвилі кількох наступаючих ворогів, від яких потрібно захищатися;
- розміщення веж або перешкод на шляху наступаючих ворогів [2].

Одним з ключових відмінностей TD ігор від інших ігор із захистом баз (таких як Space Invaders або інші ігри, де захищаються бази), є можливість гравця стратегічно розміщувати, будувати або створювати конструкції та перешкоди на шляху наступаючих ворогів. У TD іграх головною метою гравця часто є захист бази, а не власної персони, тому головний герой може виявитися непереможним [2].

Деякі особливості сучасних TD ігор:

- розставлені гравцем перешкоди, які можуть пошкодити або вбити ворожих нападників, перш ніж вони зруйнують базу;
- здатність ремонтувати вежі;
- можливість покращувати вежі;
- вороги, здатні долати декілька шляхів одночасно;
- кожна хвиля зазвичай має певну кількість і типів ворогів;
- вежа, яка має можливість пересуватися [2].

Багато сучасних TD ігор перейшли від режиму реального часу до покрокового, проходячи через окремі етапи гри, такі як будівництво, оборона, ремонт та відзначення перемоги. У багатьох іграх, таких як Flash Element Tower Defense, вороги проходять через лабіринт, що дозволяє гравцеві стратегічно розміщувати вежі для досягнення максимальної ефективності. Проте деякі варіанти цього жанру вимагають від користувача створення власного лабіринту з веж, як у Desktop Tower Defense. Деякі версії комбінують обидва підходи, маючи попередньо встановлені маршрути, які можна змінювати розташуванням веж або змінами маршрутів. Важливою стратегією часто є створення заплутаного лабіринту, щоб збільшити відстань, яку мають пройти вороги, перш ніж вони зможуть подолати оборону. Іноді

можна змінювати тактику, чергуючи блокування одного виходу, а потім іншого, щоб змусити ворогів шукати шлях, що затримує їх. Деякі ігри дозволяють гравцям також змінювати стратегію нападу, яку використовують вежі, за певну ціну, що робить гру більш цікавою [2].

1.2 Огляд існуючих патернів

Патерни програмування ігор — це набір багаторазових рішень типових проблем, з якими стикаються розробники ігор під час створення ігор. Ці шаблони призначені для покращення якості коду, зменшення складності та підвищення гнучкості та зручності обслуговування ігрових проєктів [3].

1.2.1 Game Object

Оскільки при розробці гри сцени стають складнішими, корисно помістити частину логіки в окремі файли об'єктів гри. Роблячи це, можна зберегти основний код для сцени чистим і простим для читання, одночасно інкапсулюючи багаторазові частини, які керують декількома об'єктами в сцені. Ігровий об'єкт містить усі властивості та методи для типу сутності, яку можна знайти у своїй сцені, наприклад, двері, кнопка чи монстр. Завдяки цій абстракції файли з основною логікою сцени можуть створювати повні ігрові об'єкти лише за допомогою одного рядка коду. Також можна з такою ж легкістю викликати методи цих об'єктів [4].

Патерн Game Object дозволяє гнучко створювати нові класи, створюючи один клас, кожен екземпляр якого представляє інший тип об'єкта (рис. 1.2) [5].

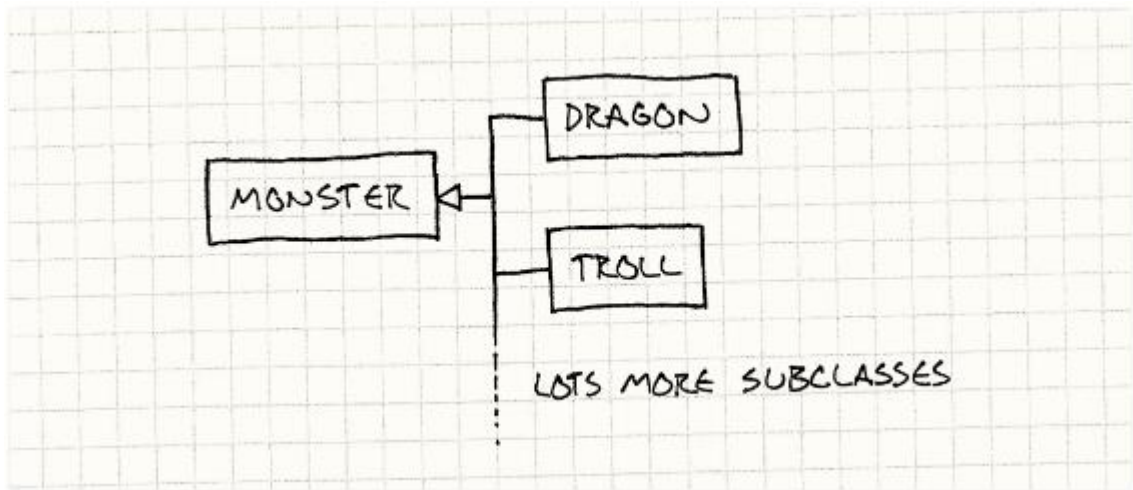


Рисунок 1.2 – Використання патерна Game Object [5]

Основні характеристики та принципи роботи патерну:

- кожен ігровий об'єкт у грі, такий як гравець, ворог, вежа або предмет, представлений як окремий ігровий об'єкт;
- кожен ігровий об'єкт може мати свої властивості (наприклад, координати, швидкість, життя) та методи (наприклад, рух, стрільба, здоров'я). Ці властивості і методи дозволяють об'єктам взаємодіяти один з одним та з ігровим світом;
- ігрові об'єкти можуть бути організовані у ієрархічну структуру. Наприклад, ігровий об'єкт може бути дочірнім до іншого об'єкта, що дозволяє організувати складні взаємозв'язки та логіку гри;
- ігрові об'єкти можуть взаємодіяти між собою та з ігровим світом. Наприклад, ворог може взаємодіяти з гравцем шляхом атаки, або вежа може стріляти в ворогів;
- ігрові об'єкти можуть спостерігати за подіями в грі та реагувати на них відповідно до внутрішньої логіки. Наприклад, вежа може спостерігати за наближенням ворога та почати стріляти;
- ігрові об'єкти можуть бути повторно використані у різних частинах гри або навіть у різних проектах.

Патерн Game Object є базовим і надзвичайно важливим в розробці ігор, оскільки він надає зручний та ефективний спосіб організації та взаємодії

з елементами гри. З його допомогою розробники можуть легко створювати складні системи та геймплей, зберігаючи при цьому чіткість та структуру коду.

Переваги даного патерну:

- патерн дозволяє розробникам швидко створювати та організовувати ігрові об'єкти без складних архітектурних рішень;
- кожен ігровий об'єкт може мати власні властивості та методи, що дозволяє створювати різноманітні елементи гри та легко їх змінювати;
- об'єкти легко взаємодіють один з одним та з ігровим світом, що дозволяє легко реалізувати різні механіки та геймплей;
- ігрові об'єкти можуть бути повторно використані у різних частинах гри або навіть у різних проєктах, що дозволяє ефективно використовувати код.

Недоліки даного патерну:

- якщо гра має велику кількість ігрових об'єктів, може виникнути проблема накопичення функціональності у кожному об'єкті, що може призвести до збільшення складності та погіршення швидкодії;
- збільшення кількості взаємозв'язків між ігровими об'єктами може призвести до високої залежності між ними, що ускладнює розуміння та підтримку коду;
- у великих проєктах з великою кількістю ігрових об'єктів даний патерн може привести до погіршення швидкодії через велику кількість обчислень та взаємозв'язків.

1.2.2 Component

Патерн Component є одним із ключових шаблонів проєктування в розробці комп'ютерних ігор. Він базується на ідеї, що кожен ігровий об'єкт складається з невеликих, незалежних компонентів, які визначають його поведінку та властивості. Цей патерн дозволяє створювати складні, гнучкі та

повторно використовувані об'єкти гри.

Одна сутність охоплює кілька доменів. Щоб зберегти домени ізольованими, код для кожного розміщується у власному класі компонентів. Сутність зводиться до простого контейнера компонентів. Такий підхід дозволяє уникнути дублювання коду в програмі (рис. 1.3) [6].

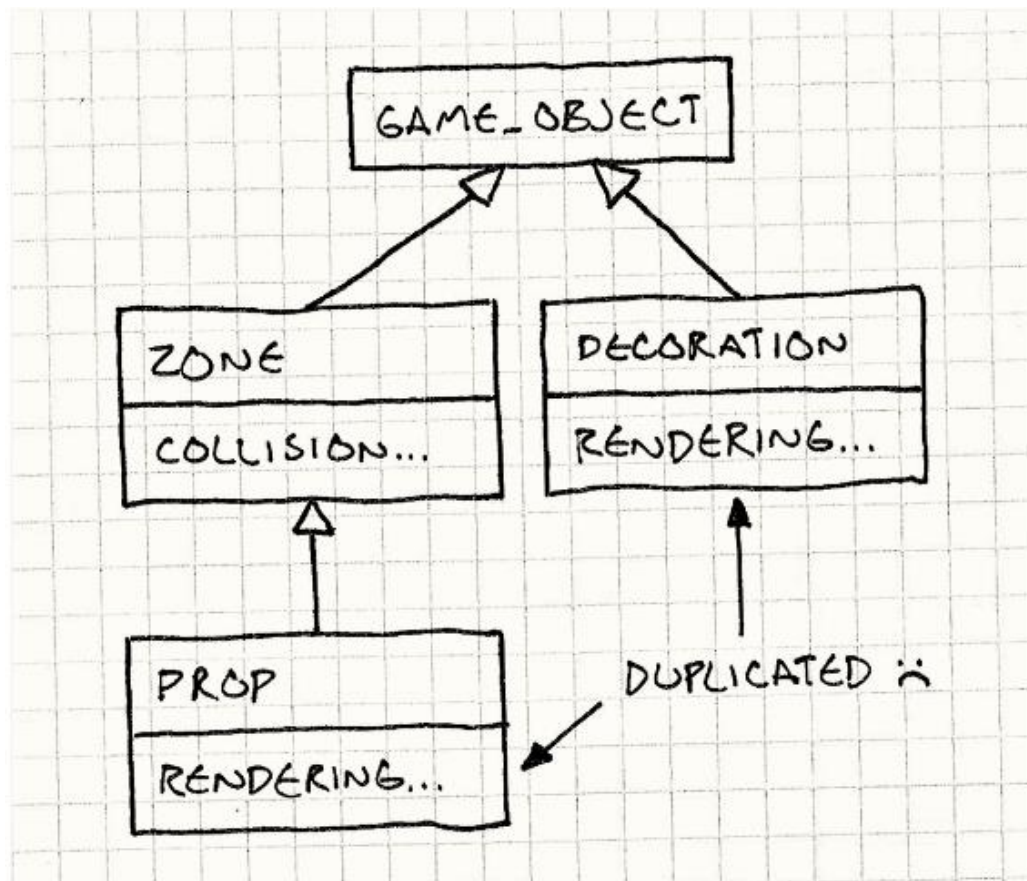


Рисунок 1.3 – Дублювання коду у Game Object [6]

Основні характеристики та принципи роботи патерну:

- ігрові об'єкти розбиваються на невеликі, незалежні компоненти, кожен з яких відповідає за конкретну функціональність або аспект поведінки;
- кожен компонент може бути повторно використаний у різних об'єктах гри, що дозволяє ефективно використовувати код та зменшувати дублювання;
- компоненти можуть взаємодіяти між собою та з іншими компонентами, що дозволяє створювати складні системи поведінки та геймплею;

- компоненти можуть динамічно додаватись та видалятись з ігрових об'єктів під час виконання програми, що надає гнучкість та можливість маневрування функціональністю;
- нові компоненти можуть легко додаватись до існуючих об'єктів без необхідності зміни вже наявного коду;
- кожен компонент відповідає за конкретну функціональність, що сприяє полегшенню розуміння та підтримки коду;
- ігрові об'єкти можуть складатися з різних компонентів, що дозволяє створювати різноманітні та унікальні об'єкти гри.

Загалом, патерн Component дозволяє створювати гнучкі та ефективні системи гри, які легко розширюються та підтримуються. Цей патерн є основою для багатьох ігрових двигунів та фреймворків і використовується в багатьох сучасних іграх.

Переваги даного патерну:

- компонентний підхід дозволяє створювати ігрові об'єкти, які легко розширюються та модифікуються. Нові компоненти можуть бути додані або видалені без значних змін у існуючому коді;
- компоненти можуть бути повторно використані в різних ігрових об'єктах або навіть у різних проєктах, що спрощує розробку та зменшує дублювання коду;
- компоненти дозволяють чітко визначити поведінку кожного ігрового об'єкта, роблячи код більш структурованим та легким у розумінні;
- компоненти дозволяють розділити функціональність ігрового об'єкта на невеликі, незалежні модулі, що спрощує тестування та підтримку коду;
- зміни у функціональності ігрового об'єкта можуть бути впроваджені швидко та легко, не впливаючи на решту коду.

Недоліки даного патерну:

- у складних проєктах з великою кількістю ігрових об'єктів, може виникати проблема з керуванням та управлінням великою кількістю компонентів;

- збільшення кількості компонентів може призвести до збільшення складності взаємодії між ними та зі зростанням розміру коду;
- уникнення зайвого перекомпонування та забезпечення ефективного керування компонентами вимагає уважного планування та архітектурного проєктування;
- несправна архітектура компонентів може призвести до переузгодження функціональності або конфліктів між компонентами.

1.2.3 Entity Component System

Entity Component System (ECS) - це архітектурний шаблон програмного забезпечення, який широко використовується в розробці відеоігор для організації об'єктів у грі. ECS складається з сутностей, які представляють собою об'єкти гри, включаючи компоненти даних, що описують їх властивості, і системи, які обробляють ці компоненти [7].

У ECS використовується принцип композиції замість успадкування: кожна сутність формується на основі свого складу компонентів, а не шляхом успадкування властивостей з ієрархії типів. Системи виконуються глобально над усіма об'єктами, які мають необхідні компоненти [7].

Сучасні процесори мають кеш-пам'ять для прискорення доступу до пам'яті. Вони можуть отримати доступ до пам'яті, суміжної з нещодавно доступною пам'яттю, набагато швидше. ECS може скористатися цим, щоб підвищити продуктивність за рахунок збільшення локальності даних — зберігання даних у безперервній пам'яті в порядку їх обробки (рис. 1.4) [8].

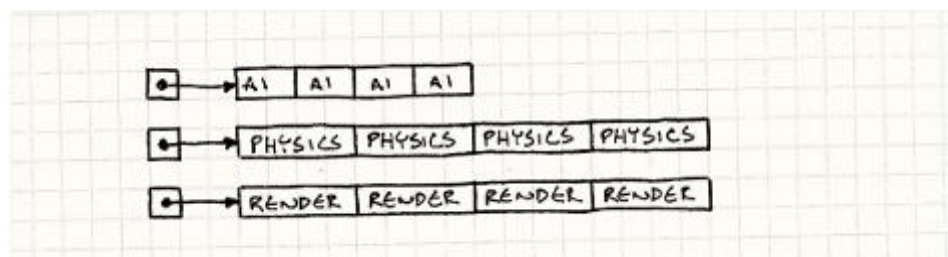


Рисунок 1.4 – Суміжні комірки пам'яті [8]

Основні складові ECS - це сутності (entities), компоненти (components) і системи (systems), які працюють разом для створення функціональності гри.

Сутність визначає загальний об'єкт. У випадку ігрового движка, кожен грубий ігровий об'єкт є прикладом сутності. Зазвичай вона має лише унікальний ідентифікатор, який часто представляється у вигляді звичайного цілого числа [7].

Компонент - це характеристика об'єкта, яка визначає його певний аспект і містить необхідні дані для моделювання цього аспекту. Наприклад, у випадку ігрових об'єктів, які можуть зазнавати пошкоджень, може існувати компонент здоров'я, який пов'язаний з їх сутністю. Зазвичай для реалізації використовують структури даних, класи або асоціативні масиви [7].

Система - це процес, який оперує всіма сутностями, що мають необхідні компоненти. Наприклад, фізична система може опрацьовувати сутності з компонентами маси, швидкості та положення, і виконувати фізичні обчислення для кожної з них, повторюючи результати [7].

Поведінка сутності може бути змінена під час виконання системами, які додають, видаляють або змінюють компоненти. Це усуває проблеми неоднозначності глибоких і широких ієрархій успадкування, які часто зустрічаються в техніках об'єктно-орієнтованого програмування, які важко зрозуміти, підтримувати та розширювати (рис. 1.5) [7].

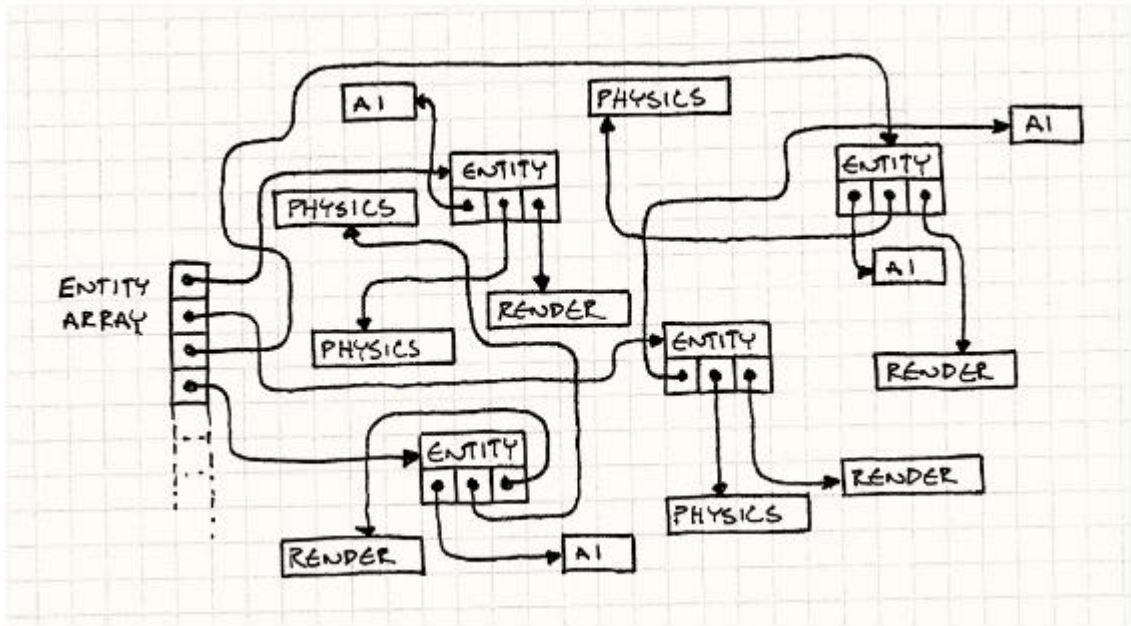


Рисунок 1.5 – Об'єктно-орієнтований підхід [8]

Загальні підходи ECS ідеально взаємодіють з методами проєктування, орієнтованими на дані, і часто поєднуються з ними. Дані для всіх екземплярів компонентів постійно зберігаються разом у фізичній пам'яті, що забезпечує ефективний доступ до пам'яті для систем, які обробляють дані багатьма об'єктами [7].

Архітектура ECS застосовує композицію замість дерева успадкування. Сутність зазвичай складається з ідентифікатора та списку компонентів, які до неї додаються. Ігровий об'єкт можна створити, додавши потрібні компоненти до сутності. Це дозволяє розробнику легко додавати функціонал до сутності без проблем із залежностями. Наприклад, до гравця можна додати компонент `bullet`, і тоді він буде відповідати вимогам для маніпулювання системою `bulletHandler`, що може призвести до того, що гравець завдасть шкоди об'єктам, на які він нарізається [7].

Переваги даного патерну:

- ECS сприяє покращенню продуктивності гри через ефективне управління пам'яттю та швидкий доступ до компонентів сутностей;
- компоненти можуть бути повторно використані у різних сутностях, що дозволяє легко змінювати та адаптувати гру до нових вимог;

- додавання нових функціональностей в гру зазвичай зводиться до створення нових компонентів і систем, що дозволяє легко розширювати функціональність;

- ECS дозволяє чітко розділити відповідальності між різними частинами коду, що полегшує розуміння та підтримку проєкту;

- ECS може бути легко оптимізований для паралельного виконання, що дозволяє ефективно використовувати багатоядерні процесори та підвищує продуктивність гри.

Недоліки даного патерну:

- ECS може призвести до збільшення кількості коду, особливо у великих проєктах, що може ускладнити розробку та підтримку;

- введення ECS у проєкт може вимагати певного часу та зусиль для розуміння та впровадження, особливо для розробників, які не мають досвіду з цим підходом;

- неправильне використання ECS може призвести до проблем з управлінням пам'яттю, таким як фрагментація пам'яті або витоки;

- у деяких випадках ECS може бути складно інтегрувати з іншими системами чи бібліотеками, що вже використовуються у проєкті.

Загалом, ECS є потужним патерном для розробки складних та оптимізованих ігор, який дозволяє розробникам створювати гнучкі та ефективні системи. Проте він вимагає ретельного планування та оволодіння концепціями для успішної реалізації.

1.2.4 Custom System

Патерн Custom System - це архітектурний шаблон в розробці комп'ютерних ігор, який полягає в створенні власних систем для обробки різних аспектів гри. Цей патерн дає розробникам велику свободу в проєктуванні та реалізації специфічної логіки та функціональності для гри.

Переваги даного патерну:

- системи можуть бути розділені на модулі, що робить код більш гнучким і полегшує розвиток та підтримку;
- патерн дозволяє створювати оптимізовані системи для обробки великої кількості даних, що може підтримувати високу швидкодію гри;
- системи можуть бути легко розширені або модифіковані для відповіді на зміни вимог гри, що дозволяє швидко реагувати на зміни та вдосконалювати гру з часом.

Недоліки даного патерну:

- велика кількість систем та їх взаємозв'язки можуть призвести до збільшення складності системи та ускладнити її розуміння;
- для створення і підтримки систем може знадобитися значна кількість коду, що може призвести до збільшення обсягу проєкту та ускладнити його розробку;
- для успішного використання цього патерну потрібен високий рівень абстракції та розуміння архітектурних принципів, що може бути викликом для менш досвідчених розробників.

Загалом, патерн Custom System є потужним інструментом для створення складних та оптимізованих ігор, проте його використання варто ретельно обмірковувати з урахуванням конкретних потреб та особливостей проєкту.

1.3 Порівняльний аналіз існуючих патернів

Ознайомившись з описаними патернами було проведено порівняльний аналіз для розробки комп'ютерної гри у жанрі Tower Defense (табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика наявних патернів для розробки комп'ютерної гри у жанрі Tower Defense

Характеристика / Назва патерну	Game Object	Game Object with Components	Entity Component System	Custom System
Складність реалізації	Середня	Низька	Висока	Висока
Складність інтеграції	Низька	Середня	Висока	Низька
Швидкість розробки	Середня	Висока	Середня	Низька
Ступінь дублювання коду	Середній	Низький	Низький	Високий
Навантаження на CPU	Середнє	Високе	Низьке	Низьке
Ефективність управління RAM	Середня	Низька	Висока	Висока
Тип проєкту	Indie	AA	AAA	AAA

Під час порівняння патернів для розробки гри, було визначено, що Component патерн, за допомогою якого можна створювати ігрові об'єкти разом з компонентами, є найбільш оптимальним вибором для розробки невеликого проєкту, ніж патерни Game Object, ECS та Custom System. Component патерн має низький рівень складності реалізації та до нього не складно інтегрувати сторонні бібліотеки, що є важливим для гри жанру Tower Defense, оскільки знадобляться бібліотеки для обчислення фізичних симуляцій, математичних операцій та відображення анімації. Однак, використання Component патерну може бути часово і ресурсомістким

процесом, особливо при великій кількості ігрових об'єктів.

1.4 Огляд існуючих аналогів

Роблячи огляд існуючих аналогів, найбільш цікавими критеріями будуть геймплей, режим гри, системні вимоги та графіка.

1.4.1 Mindustry

Mindustry - це комп'ютерна гра, доступна на різних платформах, яка комбінує стратегію в реальному часі з елементами Tower Defense (рис. 1.6). Гру створював Anuke протягом трьох днів спеціально для змагання GDL - Metal Monstrosity Jam, де вона зайняла перше місце [9].



Рисунок 1.6 – Ігрова сцена Mindustry [9]

Гравець керує різними видами техніки, яка дозволяє будувати та розбирати споруди, а також стріляти. Головна мета гри полягає в тому, щоб витримати якомога більше хвиль і зібрати якнайбільше ресурсів. На мапі присутня зона анігіляції у формі кола, де висаджуються ворожі наземні одиниці або повітряні сили з найближчого краю мапи. У момент висадки

ворожих військ у коло, споруди та союзні одиниці, разом з союзними одиницями, знищуються. Гра має велику кількість різних мап з різними видами ресурсів і різними силами ворогів. На мапах можуть бути різноманітні ресурси, від міді, свинцю та піску до титану, торію, нафти та металобрухту. Для збору ресурсів гравець може використовувати бури, насоси та дрони. Ресурси можна переносити вручну, за допомогою ресурсних гармат і конвеєрів, або за допомогою трубопроводів. Для будівництва гравець повинен мати ресурси в своєму ядрі. У грі доступні різні види турелей, які потребують різні ресурси для зарядження. Усі зібрані ресурси можна витратити на дослідження (рис. 1.7) [9].

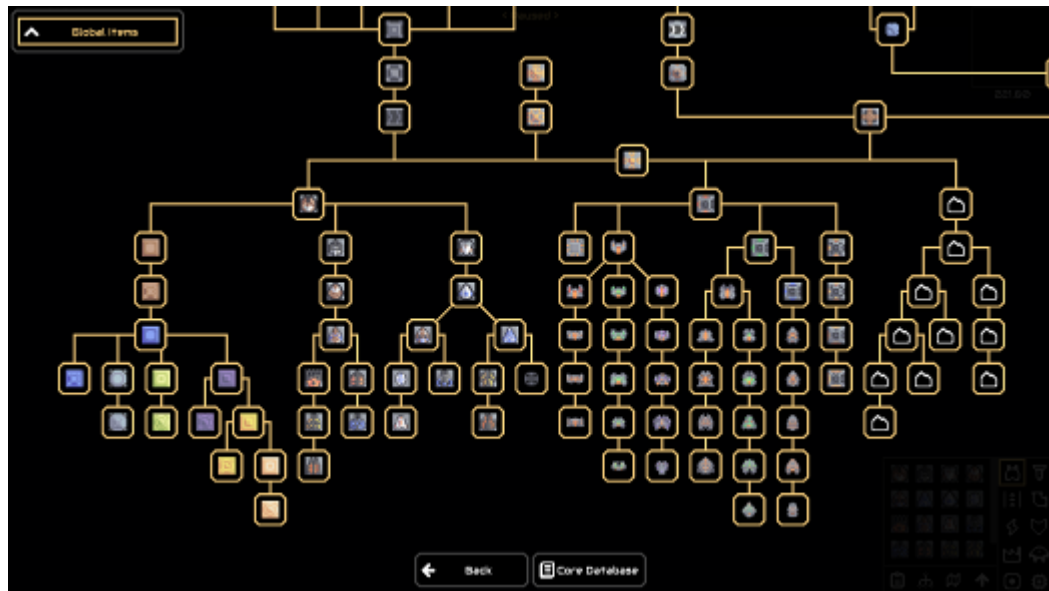


Рисунок 1.7 – Дослідження в Mindustry [9]

За допомогою отриманих ресурсів можна відкривати нові бури, вежі, заводи тощо. Дослідження поділяються на п'ять гілок:

- різні методи транспортування ресурсів на різні відстані;
- різні види турелей та блоків для оборони;
- заводи для виробництва ресурсів;
- насоси для видобування води з-під землі та з озер/річок;
- генератори, які виробляють електроенергію за допомогою вугілля та інших ресурсів [9].

Хоча в грі доступні чотири режими (Survival, Sandbox, Attack, PvP), можна створити власний, скориставшись користувацькими правилами у вікні вибору режиму. Ці правила дозволяють налаштувати параметри, такі як множники здоров'я, шкоди ворогів та гравців, швидкості, вартість будівництва та інші [9].

1.4.2 Bloons Tower Defense

Bloons Tower Defense — це серія відеоігор у жанрі Tower Defense, яка входить у серію Bloons і створена компанією Ninja Kiwi (рис. 1.8). Початково гра була розроблена як браузерна версія на платформі Adobe Flash і була випущена в середині 2007 року. Пізніше ігри серії були адаптовані для різних мобільних платформ, включаючи Android, iOS, Windows Phone, PlayStation Portable, Nintendo DSi, а також для операційних систем Windows, Linux і MacOS. Ігри з серії Bloons, випущені до Bloons TD 6, доступні для придбання у каталозі Ninja Kiwi в Steam [10].



Рисунок 1.8 – Ігрова сцена Bloons Tower Defense [10]

Основна ціль в грі Bloons TD полягає в тому, щоб не допустити bloons

досягти кінця визначеної траси на карті, яка містить один або кілька входів і виходів для bloons. Гра є типовою грою Tower Defense, тому гравець може обирати різні типи веж і пасток, щоб розставити їх навколо доріжки та захиститися від bloons. За кожен bloon, який виходить на трасу, гравець отримує 1 долар у грі. Якщо bloon досягає кінця шляху, гравець втрачає життя (у наступних версіях гри - здоров'я). Якщо життя закінчується, гра завершується. Bloons завжди рухаються вздовж маршруту, встановленого на карті, поки вони не досягнуть виходу(ів), не будуть знищені або не будуть переміщені на попередню частину шляху за допомогою особливих властивостей вежі [10].

У грі доступні три класи bloon: звичайний (не має власної назви у грі), МОАВ-клас і бос-клас. У Bloons TD 6 звичайні bloon поділяються на: червоні, сині, зелені, жовті, рожеві, чорні, білі, свинцеві, зеброві, райдужні, фіолетові і керамічні. Bloons класу МОАВ мають форму дирижабля і складаються з: МОАВ (Massive Ornary Air Blimp), BFB (Brutal Flying Behemoth), DDT (Dark Dirigible Titan), ZOMG (Zeppelin Of Mighty Gargantuaness) і BAD (Big Airship of Doom). Найскладніші варіанти більшості типів bloon включають в себе певну кількість менших. У пізніших версіях гри звичайні bloons іноді мають спеціальні характеристики, такі як камуфляж (який більшість веж не можуть виявити), відростання (здатність bloons повільно відростати до початкового розміру) і укріплені (що подвоює здоров'я кулі). З кожним рівнем інтенсивність bloons зростає пропорційно [10].

1.4.3 Plants vs. Zombies

Серія Plants vs. Zombies складається з багатьох ігор різних жанрів, розроблених PopCap Games і виданих Electronic Arts. У цій серії є дві великі фракції, які постійно борються між собою, рослини та зомбі (рис. 1.9) [11].



Рисунок 1.9 – Ігрова сцена Plants vs. Zombies [11]

У кожній грі є людина на ім'я Божевільний Дейв, який допомагає гравцеві зупинити зомбі та їх ватажка доктора Едгара Зомбоса, головного антагоніста серії. Кожна гра складається з багатьох різних ігрових режимів, щоб збільшити можливості та задоволення від гри. Серія також доступна для різноманітних платформ, від портативних консолей, таких як DS, DSiWare, до домашніх консолей, таких як Xbox 360 і PlayStation 3 [11].

В основній серії, яка складається з Plants vs. Zombies, Plants vs. Zombies 2 і Plants vs. Zombies 3, які слідує жанру Tower Defense, гравцям потрібно використовувати різні рослини, надані для захисту від зомбі, які намагаються вторгнутися на будинок гравця, щоб з'їсти їхні мізки. Серія складається з безліч рослин. Кожна рослина унікальна завдяки своїм різним здібностям. Зомбі також мають різноманітні особливі здібності. Наприклад, вони використовують дорожні конуси та відра, щоб захиститися, деякі можуть використовувати предмети, щоб стрибати через рослини, а до сильніших зомбі належать Гаргантуари та Гіга-гаргантуари [11].

1.4.4 Thronefall

Thronefall — це майбутня стратегічна відеогра, розроблена та видана Grizzly Games (рис. 1.10). Він був випущений у ранньому доступі для Windows 2 серпня 2023 року. У грі гравці керують монархом на коні, який повинен будувати своє королівство вдень, а вночі захищати його від зростаючих хвиль ворогів. Монарх може битися в битві разом зі своїми солдатами, вибираючи між кількома видами зброї, такими як лук або спис, які мають свої особливі властивості. На початку кожного дня гравець повертає всі свої зруйновані будівлі та одиниці та отримує певну кількість золота залежно від того, скільки його споруд залишилося недоторканими [12].



Рисунок 1.10 – Ігрова сцена Thronefall [12]

Королівство зосереджено навколо великого залу, який монарх повинен захищати. Вони можуть захистити його, витрачаючи золото на будівництво стін, оборонних веж та інших корисних будівель. Монарх може заробляти золото, будуючи будинки та сільськогосподарські угіддя всередині королівства, на користь до інших економічних структур, таких як шахти та

рибальства. Такі солдати, як арбалетники та лицарі, можуть бути набрані в будівлях казарм і можуть бути розміщені монархом навколо рівня для захисту певних територій від нападу. Коли монарх завершить підготовку свого королівства, вони можуть вибрати ніч, щоб прибути й розпочати захист від ворожих хвиль. Крім оборонних споруд і одиниць, гравець також може захищати королівство, атакуючи ворогів зброєю монарха, яка включає лук і спис, які володіють особливими здібностями. Завершення ночі перемогою над усіма ворогами дозволяє гравцеві повернути всі зруйновані будівлі та одиниці, а також надає монарху певну кількість золота залежно від кількості будівель, які пережили атаки [12].

1.5 Порівняльний аналіз існуючих аналогів

Ознайомившись з описаними аналогами було проведено порівняльний аналіз для розробки комп'ютерної гри у жанрі Tower Defense (табл. 1.2).

Таблиця 1.2 – Порівняльна характеристика наявних аналогів для розробки комп'ютерної гри у жанрі Tower Defense

Характеристика / Назва гри	Mindustry	Bloons Tower Defense	Plants vs. Zombies	Thronefall
1	2	3	4	5
Геймплей	Стратегічне будування та захист фабрик	Різноманітність веж та ворогів, динамічні рівні	Унікальний геймплей з різними типами рослин та ворогів	Стратегічний геймплей з елементами фентезі
Графіка	Висока	Середня	Середня	Висока

Продовження таблиці 1.2

1	2	3	4	5
Режим гри	Одиночна гра, мультиплеєр, режими викликів	Одиночна гра, мультиплеєр, спеціальні режими	Одиночна гра, пазли, міні-ігри	Одиночна гра, мультиплеєр, режими викликів
Системні вимоги	Середні	Середні	Низькі	Високі

Під час порівняння існуючих аналогів, було визначено, що Mindustry є найбільш цікавим проектом, ніж інші аналоги. Тому, для розробки власної комп'ютерної гри в жанрі Tower Defense, було прийнято рішення взяти декілька цікавих механік з цієї гри, а саме: режим викликів, будування споруд у довільному просторі, переміщення ворогів по довільній траєкторії, управління власною одиницею.

1.6 Постановка завдання до роботи

Метою роботи є програмна реалізація комп'ютерної гри у жанрі Tower Defense.

Під час аналізу наявних патернів для розробки гри, було встановлено, що найбільш оптимальним патерном є Component. Тому було вирішено розробити систему з ігровими об'єктами, які складаються з компонентів.

Під час аналізу наявних аналогів для розробки гри, було встановлено, що найбільш цікавим проектом є Mindustry. Тому було вирішено розробити деякі схожі механіки гри:

- управління власною одиницею;
- система викликів;
- будування споруд на довільній площі;

- нескінченні хвилі ворогів;
- переміщення ворогів по довільній траєкторії.

Задачі роботи:

- аналіз вимог до системи;
- вибір інструментів розробки;
- розробка архітектури системи;
- програмна реалізація комп'ютерної гри;
- тестування фінального проєкту.

1.7 Висновок

У цьому розділі було проведено аналіз предметної області, визначено основні поняття, пов'язані з предметною областю і об'єктом дослідження та розробки. Було розглянуто проблему розробки комп'ютерної гри у жанрі Tower Defense.

Було досліджено основні патерни для розробки комп'ютерної гри та проведено порівняльний аналіз цих патернів, який показав перевагу використання Component патерну.

Проведено порівняльний аналіз існуючих аналогів та були визначені основні вимоги до системи та механіки гри.

Було сформульовано і поставлено завдання до роботи.

2 ОПИС ПРОГРАМИ

2.1 Вибір мови програмування

Мова програмування - це штучна система символів, створена для видачі інструкцій комп'ютерам та іншим машинам. Вона використовується для розробки програм, які керують поведінкою машин та для опису алгоритмів [13].

У кожної мови програмування є свої переваги та недоліки. Найчастіше, для розробки комп'ютерної гри, використовують такі мови програмування як: C++, C#, Python.

2.1.1 C++

C++ - це мова програмування загального призначення, яка підтримує кілька парадигм програмування, включаючи об'єктно-орієнтовану, узагальнену, процедурну та інші. Б'ярн Страуструп розпочав розробку C++ в AT&T Bell Laboratories у 1979 році. Початково мова мала назву "Сі з класами". У 1984 році Страуструп перейменував її на C++. Перший офіційний опис мови був представлений у міжнародному стандарті ISO/IEC 14882:1998 (C++98), а найбільш актуальним стандартом на даний час є ISO/IEC 14882:2020 (C++20) [14].

У 1990-х роках C++ стала однією з найбільш поширених мов програмування загального призначення. Цю мову використовують для широкого спектру завдань, таких як системне програмування, розробка застосункового програмного забезпечення, написання драйверів, створення потужних серверних та клієнтських програм, а також для розробки розважального програмного забезпечення, наприклад, відеоігор. C++ має значний вплив на інші популярні мови програмування сучасності, такі як C# та Java [14].

При створенні C++ було прагнення зберегти сумісність з мовою С.

Більшість програм, написаних на C, мають здатність працювати без проблем і з компілятором C++. Синтаксис C++ базується на синтаксисі мови C [14].

Інновації в C++ включають в себе наступне:

- можливість об'єктно-орієнтованого програмування через використання класів;
- підтримку узагальненого програмування за допомогою шаблонів;
- введення додаткових типів даних;
- механізм обробки винятків;
- використання просторів імен;
- перевантаження операторів;
- перевантаження імен функцій;
- використання посилань та операторів для керування вільно розподіленою пам'яттю [14].

Стандартна бібліотека C++ включає в себе стандартну бібліотеку C з невеликими модифікаціями, які роблять її більш адаптованою для використання в мові C++. Ще однією важливою складовою бібліотеки C++ є Стандартна Бібліотека Шаблонів (STL). Вона надає широкий спектр інструментів, таких як контейнери (наприклад, вектори та списки) і ітератори (загальноновживані вказівники), що дозволяють працювати з цими контейнерами як зі звичайними масивами. Крім того, STL забезпечує можливість аналогічно працювати з іншими видами контейнерів, такими як асоціативні масиви, стеки та черги [14].

Шляхом використання шаблонів можна створювати загальні алгоритми, які можуть опрацьовувати будь-які контейнери або послідовності, доступ до елементів яких забезпечують ітератори [14].

Успіху STL, особливо її включення до стандартної бібліотеки C++, сприяло спрямування на широкий спектр завдань та узагальнена структура. У цьому відношенні, бібліотека Boost сьогодні дуже близька за духом до STL. Також є бібліотекою загального призначення, і впливає на формування стандартної бібліотеки C++ [14].

Переваги:

- C++ дозволяє написання швидкодіючих програм, оскільки вона набагато ближча до машинного коду порівняно з вищими рівнями мов програмування, такими як Python або Java;
- У C++ є можливість прямого керування пам'яттю, що дає розробникам більш гнучкий контроль над ресурсами;
- C++ підтримує об'єктно-орієнтоване програмування, що дозволяє створювати класи та об'єкти, що спрощує структуру програми та полегшує її розширення та обслуговування;
- в C++ можна прямо взаємодіяти з апаратним забезпеченням, наприклад, з використанням вказівників, асемблерних вбудовок та інших низькорівневих конструкцій;
- багато мов програмування можуть взаємодіяти з C++ через її низькорівневі функції та механізми взаємодії, такі як C API.

Недоліки:

- C++ може бути складною для вивчення та використання, особливо для новачків;
- безпека пам'яті та інших аспектів програми може бути проблемою в C++, оскільки розробники мають прямий доступ до пам'яті та можуть зробити помилки, що призводять до сегментації пам'яті, переповнення буфера та інших серйозних проблем;
- C++ не має автоматичного збирання сміття, що означає, що розробник повинен вручну керувати виділенням та звільненням пам'яті, що може призвести до витоку пам'яті та інших проблем;
- у C++ існують різні версії та стандарти, що може призвести до несумісності між кодом, написаним для одного стандарту, та іншим;
- зазвичай програми на C++ вимагають більшого обсягу коду порівняно з іншими мовами програмування для досягнення того ж функціоналу, що може збільшити час розробки та підтримки.

2.1.2 C#

C# - це мова програмування, яка орієнтована на об'єкти, з безпечною системою типізації, розробленою Андерсом Гейлсбергом, Скотом Вілтанутом та Пітером Гольде під егідою Microsoft Research для платформи .NET [15].

Синтаксис C# схожий на той, що використовується в мовах програмування C++ та Java. Мова має строгу статичну типізацію та підтримує такі концепції як поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки та коментарі у форматі XML. Хоча C# спадкує багато від своїх попередників, таких як мови C++, Object Pascal, Модула і Smalltalk, він також виключає деякі моделі, що можуть бути проблематичними при розробці програмних систем. Наприклад, у відміну від C++, мова C# не підтримує множинне успадкування класів [15].

Специфікація C# визначає найменший необхідний набір типів і класів бібліотек, на які може розраховувати компілятор. У реальних проєктах C# часто використовується з певною реалізацією Common Language Infrastructure (CLI), яка є стандартизованою під назвою ECMA-335 Common Language Infrastructure (CLI) [15].

Переваги:

- C# має простий і зрозумілий синтаксис, що робить її дуже доступною для новачків у програмуванні;
- C# розроблена компанією Microsoft і широко використовується для розробки програм для різних платформ, таких як Windows, Linux, і macOS;
- C# поставляється з великою стандартною бібліотекою, яка містить багато корисних функцій і класів для різних завдань програмування, що дозволяє розробникам швидко створювати програми;
- C# має вбудовану систему автоматичного збору сміття (garbage collection), яка звільняє розробників від необхідності ручного керування пам'яттю та запобігає витокам пам'яті.

Недоліки:

- хоча C# тепер підтримується на багатьох платформах, вона все ще найбільш повністю інтегрована з платформою Microsoft, що може обмежити її використання на інших платформах;
- у порівнянні з мовами, такими як C++, C# має обмежений доступ до низькорівневих операцій та апаратних ресурсів;
- деякі функції C# можуть залежати від платформи або версії .NET Framework, що може призвести до проблем з сумісністю між версіями;
- хоча є платформи, такі як Xamarin, для розробки мобільних застосунків на C#, вона не має вбудованої підтримки мобільних платформ, як це має місце, наприклад, у Java з його платформою Android.

2.1.3 Python

Python - це мова програмування високого рівня з об'єктно-орієнтованим підходом та гнучкою динамічною типізацією, яку розробив Гвідо ван Россум у 1990 році. Її використання спрощує швидку розробку програм завдяки високорівневим структурам даних, динамічній семантиці та динамічному зв'язуванню. Python сприяє модульності та повторному використанню коду за допомогою підтримки модулів та пакетів модулів. Інтерпретатор Python і стандартні бібліотеки доступні як у скомпільованій, так і у вихідній формі для всіх основних платформ. Мова підтримує кілька парадигм програмування, таких як об'єктно-орієнтована, процедурна, аспектно-орієнтована та функціональна [16].

Серед головних переваг Python можна виділити наступні:

- чіткий синтаксис, де блоки коду визначаються за допомогою відступів;
- переносність програм, що є характерною рисою для більшості інтерпретованих мов;
- широкий вибір корисних модулів у стандартному дистрибутиві,

включаючи модулі для створення графічних інтерфейсів;

- можливість використання Python у діалоговому режимі, що дуже зручно для експериментів та вирішення простих завдань;
- просте, але потужне середовище розробки у стандартному дистрибутиві, відоме як IDLE, яке написане мовою Python;
- зручність для розв'язання математичних завдань, зокрема можливість працювати з комплексними числами, великими цілими числами та використання Python як калькулятора у діалоговому режимі;
- відкритий код, що дозволяє користувачам редагувати його за власним бажанням [16].

Python пропонує ефективні структури даних високого рівня і простий, але дієвий підхід до об'єктно-орієнтованого програмування. Його елегантний синтаксис, динамічна обробка типів і інтерпретований характер роблять його ідеальним для написання скриптів та швидкої розробки програм у багатьох галузях на різних платформах [16].

Можливості інтерпретатора Python можна розширити за допомогою функцій та типів даних, що розроблені на C або C++ (або іншій мові, яку можна викликати з C). Крім того, Python зручна як мова розширення для прикладних програм, які потребують подальшої модифікації [16].

2.2 Порівняння мов програмування

Ознайомившись з описаними вище мовами програмування, було проведено порівняльний аналіз у напрямку розробки ігор (табл. 2.1).

Таблиця 2.1 – Порівняльна характеристика мов програмування

Характеристика / Назва	C++	C#	Python
1	2	3	4

Продовження таблиці 2.1

1	2	3	4
Продуктивність	Дуже висока швидкодія, ідеально підходить для розробки високопродуктивних ігор	Швидкодійна мова програмування, відмінно підходить для створення ігор для платформ Microsoft	Повільніший, але досить прийнятний для прототипування та розробки невеликих ігор
Використання в геймдеві	Широко використовується для розробки ігор, включаючи AAA-проекти, консольні та ПК-ігри, особливо завдяки Unreal Engine	Популярний в геймдеві, особливо завдяки Unity, яка дозволяє розробляти ігри для різних платформ, включаючи консолі, ПК, мобільні телефони	Використовується для створення простих ігор, прототипів, а також ігор з важким акцентом на інді-проекти, арт-стиль та геймплей
Графічний інтерфейс	Прямий доступ до графічного движка та апаратного забезпечення	Має зручний інтерфейс для створення графіки, але може бути менш гнучким порівняно з C++	Часто використовується з бібліотеками, такими як Pygame або Pyglet, для створення 2D графіки та ігор

Продовження таблиці 2.1

1	2	3	4
Контроль ресурсів	Має повний контроль над ресурсами, що дозволяє ефективно використовувати ресурси та оптимізувати продуктивність	Надає вищий рівень абстракції, але все ще дозволяє ефективне управління ресурсами	Менший контроль над ресурсами порівняно з C++ та C#, але відмінно підходить для невеликих проєктів з обмеженими ресурсами
Інтеграція з іншими інструментами	Має велику кількість бібліотек та фреймворків, що підтримуються в геймдеві, включаючи графічні рушії, фізичні симуляції, звукові бібліотеки та інше	Підтримує широкий спектр інтеграції з іншими інструментами, зокрема Unity	Існують бібліотеки та фреймворки, але вони можуть бути менш доступними порівняно з C++ та C#
Комунітет та підтримка	Має велику та активну спільноту, зокрема в середовищі Unreal Engine	Є широка спільнота розробників, зокрема в середовищі Unity	Має велику та активну спільноту, але може бути менш спеціалізованою

Отже, порівнявши три мови програмування, було вирішено надати

перевагу використання C++ та розробити комп'ютерну гру. Основні переваги вибору C++ включають високу швидкість та ефективність, повний контроль над ресурсами, гнучкість та масштабованість, а також широкий вибір бібліотек та фреймворків, які підтримуються для розробки ігор. Однак варто також врахувати, що розробка на C++ може бути складнішою порівняно з іншими мовами програмування, і вона вимагає глибокого розуміння мови та концепцій програмування, а також більше часу на вирішення проблем та оптимізацію коду.

2.3 Вибір середовища розробки

Інтегроване середовище розробки - це комплексне програмне рішення, що використовується для створення програмного забезпечення. Зазвичай воно включає в себе редактор початкового коду, інструменти для автоматизації компіляції та відлагодження програм. Більшість сучасних інтегрованих середовищ розробки мають функцію автодоповнення коду, що спрощує процес написання програм [17].

2.3.1 Visual Studio

Microsoft Visual Studio є серією продуктів від компанії Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та різноманітні інструменти. Ці продукти дозволяють розробляти як консольні, так і графічні програми, включаючи підтримку технології Windows Forms, а також вебсайти, вебзастосунки та вебслужби як на рідному, так і керованому коді для різних платформ, таких як Windows Phone, Microsoft Windows, Windows CE, Windows Mobile, .NET Framework, .NET Compact Framework та Microsoft Silverlight [18].

Переваги:

- Visual Studio надає все необхідне для створення програм - від

редактора коду до інструментів для налагодження, відстеження та керування версіями;

- Visual Studio підтримує не лише мови програмування з сімейства .NET, такі як C# та VB.NET, але і мови програмування, такі як C++, Python, JavaScript;

- Visual Studio інтегрується з різними сервісами та інструментами, такими як Azure DevOps, GitHub, Docker, що полегшує роботу у команді та розробку хмарних застосунків;

- Visual Studio має потужні засоби для налагодження програм, включаючи крок-за-кроком виконання, відстеження значень змінних, аналіз пам'яті;

- від Windows і Xbox до мобільних пристроїв та хмарних сервісів - Visual Studio дозволяє розробляти програми для різних платформ.

Недоліки:

- Visual Studio важка для використання на менш потужних комп'ютерах, оскільки вимагає велику кількість ресурсів пам'яті та процесора;

- для новачків може бути важко освоїти всі можливості Visual Studio через велику кількість функціоналу та налаштувань;

- хоча існує безкоштовна версія Visual Studio Community, але повні функціональні версії можуть бути дорогими для окремих користувачів чи невеликих команд;

- залежно від версії та налаштувань, можуть виникати проблеми зі сумісністю, особливо коли працюєте зі старими проектами або використовуєте сторонні плагіни;

- хоча Visual Studio підтримує багато мов програмування, для деяких з них може бути менша підтримка порівняно з іншими IDE, що спеціалізуються на цих мовах.

2.3.2 CLion

CLion — це комерційна кросплатформна IDE для C/C++. JetBrains користується вірними прихильниками серед розробників, які люблять їхні чудові редактори коду, що містять чудові інструменти рефакторингу коду. CLion включає аналіз коду, підтримку CMake, моделювання проєкту з інтелектуальною підтримкою коду. CLion — одна з небагатьох IDE, яка підтримує локальну та віддалену (через SSH), що дозволяє вам кодувати на локальній машині, але компілювати на віддалених серверах. Хоча CLion зосереджений виключно на C/C++, він базується на середовищі IDE JetBrains із версіями для найпопулярніших мов програмування [19].

Переваги:

- CLion спеціалізується на роботі з мовою програмування C++, надаючи розширений функціонал та інструменти, спеціально призначені для розробки на цій мові;
- CLion має вбудовану систему відладки, а також підтримку інтеграції з зовнішніми відладчиками, що дозволяє ефективно вирішувати проблеми та відстежувати помилки;
- CLion підтримує роботу на різних операційних системах, включаючи Windows, macOS та Linux, що дозволяє розробникам використовувати її на своєму улюбленому середовищі;
- інтегровані інструменти аналізу коду допомагають виявляти потенційні проблеми, такі як синтаксичні помилки, неправильне використання змінних;
- CLion має систему плагінів, яка дозволяє розширити його функціонал за допомогою сторонніх розширень та інтеграцію з іншими інструментами розробки.

Недоліки:

- порівняно з іншими середовищами розробки, CLion може мати високі витрати на придбання ліцензії для комерційного використання;

- робота з CLion може вимагати значних обсягів оперативної пам'яті та процесорної потужності, що може вплинути на продуктивність на менш потужних комп'ютерах;
- перехід на нове середовище розробки може потребувати часу на навчання та адаптацію, особливо якщо розробники вже звикли до іншого інструменту;
- хоча CLion спеціалізується на мові програмування C++, він може мати обмежену підтримку для інших мов, що може бути недоліком для розробників, які працюють з різними мовами;
- як і в будь-якого комерційного програмного забезпечення, користувачі CLion повністю залежать від компанії-розробника для підтримки, оновлень та виправлень помилок.

2.3.3 Custom IDE

Custom IDE - це інтегроване середовище розробки, яке може бути налаштоване користувачем з урахуванням його власних потреб та вимог. Основна ідея полягає в тому, щоб створити робоче середовище, яке оптимально відповідає конкретним завданням та способу роботи розробника.

Переваги:

- користувач може налаштувати середовище розробки згідно з власними потребами, включаючи вибір редакторів коду, плагінів, тем оформлення та інших налаштувань;
- врахування індивідуальних пристрастей та робочих потреб розробника може покращити продуктивність, оскільки інструменти будуть оптимізовані саме для його робочого процесу;
- можливість вибору різних інструментів та їх комбінування дозволяє створювати гнучкі та ефективні робочі потоки;
- користувач може виключити або приховати функції, які йому не потрібні, що дозволяє спростити інтерфейс та зосередитися на важливих

завданнях.

Недоліки:

- створення та налаштування власного IDE може вимагати значного часу та зусиль;
- власні IDE можуть виявитися менш стабільними або менш сумісними з певними інструментами або мовами програмування;
- підтримка та оновлення власного IDE також може бути вимогливою задачею, особливо якщо вона базується на великій кількості різних компонентів та плагінів;
- використання власного IDE може ускладнити процес навчання нових членів команди або співпраці з іншими розробниками, які не використовують такий самий набір інструментів.

2.4 Порівняння середовищ розробки

Ознайомившись з описаними інтегрованими середовищами розробки, було проведено порівняльний аналіз для того, щоб вибрати найкращий варіант (табл. 2.2).

Таблиця 2.2 – Порівняльна характеристика інтегрованих середовищ розробки

Характеристика / Назва	Visual Studio	CLion	Custom IDE
1	2	3	4
Мови програмування	Підтримує різні мови, такі як C#, C++, Python, JavaScript	Основна підтримка для мови програмування C++, Python	Залежить від конфігурації, але може бути налаштований для будь-якої мови програмування

Продовження таблиці 2.2

1	2	3	4
Вбудовані інструменти	Має широкий набір вбудованих інструментів для розробки, тестування, налагодження	Має інструменти для роботи з проєктами на C++, включаючи вбудований відлагоджувач, автодоповнення	Залежить від того, які компоненти були додані користувачем. Включає будь-які інструменти за вибором користувача
Налагоджувальні можливості	Має потужний відлагоджувач з різними можливостями налагодження	Має вбудований відлагоджувач зі схожими можливостями для проєктів на мові C++	Може мати різні відлагоджувальні інструменти, залежно від вибору користувача
Гнучкість	Має обмежену гнучкість, адаптовану для роботи з продуктами Microsoft	Має обмежену гнучкість, адаптовану для роботи з проєктами на мові програмування C++	Максимально гнучкий, може бути налаштований з урахуванням конкретних потреб користувача

Продовження таблиці 2.2

1	2	3	4
Підтримка плагінів	Має деяку підтримку плагінів, але в основному обмежується плагінами від Microsoft та сторонніми розробниками	Має підтримку плагінів, але знову ж таки, в основному обмежується плагінами, які підтримують розробники CLion	Має повну підтримку плагінів, включаючи можливість додавання будь-яких плагінів згідно з власними потребами користувача
Вартість	Є різні версії, включаючи безкоштовну Community версію та платні Professional і Enterprise версії	Має пробний період та платну ліцензію для комерційного використання	Залежить від того, які компоненти використовуються для створення власного IDE. Може бути як безкоштовним, так і платним

Отже, порівнявши три IDE, було вирішено надати перевагу використанню Custom IDE для розробки комп'ютерної гри. Основні переваги вибору Custom IDE включають максимальну гнучкість та індивідуалізацію у своєму робочому середовищі. Custom IDE дозволяє мати повний контроль над своїм робочим процесом та інструментами, що може допомогти досягти оптимальної продуктивності та забезпечити зручність роботи.

2.5 Вибір мультимедійної бібліотеки

Мультимедійна бібліотека - це набір програмних інструментів, функцій та ресурсів, які призначені для розробки програм, що працюють з різноманітними мультимедійним вмістом, таким як зображення, звук, відео та графіка. Ці бібліотеки зазвичай надають зручний інтерфейс програмування (API), що дозволяє розробникам легко взаємодіяти з мультимедійними ресурсами та використовувати їх у своїх програмах.

2.5.1 SDL

Simple DirectMedia Layer (SDL) — це бібліотека, яка пропонує кросплатформовий абстрактний рівень для мультимедійних компонентів комп'ютера. Розробники програмного забезпечення можуть використовувати її для створення високопродуктивних комп'ютерних ігор та інших мультимедійних застосунків, які працюють на різних операційних системах, таких як Android, iOS, Linux, macOS і Windows [20].

SDL забезпечує управління відео, аудіо, пристроями введення, CD-ROM, потоками, завантаженням спільних об'єктів, мережею та таймерами. У сфері 3D-графіки вона підтримує контекст OpenGL, Vulkan, Metal або Direct3D. Поширеною помилкою є вважати, що SDL є ігровим движком. Проте ця бібліотека може бути використана для прямого створення ігор або застосовується як база для розробки ігрових движків [20].

SDL широко використовується в галузі як у великих, так і в малих проєктах. На сайті бібліотеки розміщено понад 700 ігор, 180 програм і 120 демонстрацій [20].

Переваги:

- SDL підтримує велику кількість операційних систем, включаючи Windows, macOS, Linux, а також інші, такі як Android та iOS, що робить її ідеальним вибором для розробки крос-платформених застосунків та ігор;

- SDL має простий та легкий у використанні інтерфейс програмування застосунків (API), що дозволяє швидко розпочати роботу з бібліотекою навіть початківцям;

- SDL надає зручний доступ до роботи з відео, аудіо, клавіатурою, мишею та іншими пристроями вводу та виводу, що робить її досить універсальною для розробки ігор та програм з мультимедійними можливостями;

- SDL розповсюджується під ліцензією zlib, що дозволяє використовувати її у комерційних проєктах, розробляти власні модифікації та вносити внески до розвитку бібліотеки.

Недоліки:

- деякі версії SDL можуть мати проблеми зі стабільністю або сумісністю з певними операційними системами чи конфігураціями обладнання;

- хоча SDL надає базовий набір функцій для мультимедійної розробки, в деяких випадках може бути обмеженою для складних або високопродуктивних застосувань порівняно з іншими бібліотеками;

- SDL в основному орієнтована на 2D графіку, тож для розробки 3D ігор чи програм може знадобитися додаткова бібліотека або фреймворк;

- у порівнянні з деякими іншими популярними мультимедійними бібліотеками, такими як SFML, SDL може мати меншу кількість активних користувачів та розвиваючу спільноту.

2.5.2 SFML

SFML — це кросплатформна бібліотека для розробки програмного забезпечення, спрямована на надання простого API для роботи з різноманітними мультимедійними компонентами комп'ютерів. Написана на C++, SFML має прив'язки для мов програмування, таких як Ada, C, Crystal, D, Euphoria, Go, Java, Julia, .NET, Nim, OCaml, Python, Ruby та Rust. Також

доступні експериментальні мобільні порти для Android та iOS, які з'явилися з випуском версії SFML 2.2 [21].

SFML керує створенням і введенням у вікна, а також створенням і керуванням контекстами OpenGL. Він також надає графічний модуль для простого апаратного прискорення двовимірної комп'ютерної графіки, який включає рендеринг тексту за допомогою FreeType, аудіомодуль, який використовує OpenAL, і мережевий модуль для базового зв'язку за протоколом керування передачею (TCP) і протоколом дейтаграм користувача (UDP) [21].

Переваги:

- SFML має досить простий та легкий у використанні API, що робить його привабливим для початківців та допомагає швидко розпочати роботу над проєктами;
- SFML підтримує різні операційні системи, включаючи Windows, macOS, Linux та інші, що дозволяє розробникам створювати крос-платформенні застосунки та ігри;
- бібліотека SFML розділена на невеликі модулі, що дозволяє користувачам використовувати лише ті функції, які їм потрібні, зменшуючи зайвий обсяг коду та ресурсів.

Недоліки:

- SFML в основному призначений для роботи з 2D графікою, і його можливості у сфері 3D графіки обмежені;
- SFML, хоч і має активну спільноту, може мати меншу кількість користувачів та ресурсів порівняно з іншими популярними мультимедійними бібліотеками, що може ускладнити пошук допомоги та розв'язання проблем;
- у порівнянні з деякими іншими мультимедійними бібліотеками, такими як SDL, SFML може мати меншу кількість функцій та можливостей.

2.5.3 Allegro

Allegro — це програмна бібліотека, спеціально призначена для розробки відеоігор. Вона включає в себе ряд корисних функцій, таких як базова 2D-графіка, робота з зображеннями, виведення тексту, аудіо та MIDI-відтворення, а також можливості для роботи з введенням, таймерами, матричною арифметикою, рядками Unicode, доступом до файлової системи і 3D-графікою. Бібліотека написана на мові програмування C і призначена для використання з C, C++ або Objective-C, а також має прив'язки для Python, Lua, Scheme, D, Go та інших мов програмування. Крім того, Allegro постачається з обширною документацією та великою кількістю прикладів використання [22].

Allegro підтримує Windows, macOS, Unix-подібні системи, Android та iOS, абстрагуючи їхні інтерфейси прикладного програмування (API) в один портативний інтерфейс. Він також може працювати поверх простого рівня DirectMedia, який використовується для запуску програм Allegro у веббраузері за допомогою Emscripten [22].

Переваги:

- Allegro підтримує різні операційні системи, такі як Windows, macOS, Linux та інші, що робить його ідеальним вибором для крос-платформенної розробки;
- бібліотека Allegro надає доступ до різних функцій, включаючи роботу з графікою, аудіо, вводом користувача, анімацією та багато іншого;
- Allegro має простий та легкий у використанні інтерфейс програмування застосунків (API), що дозволяє швидко розпочати роботу з бібліотекою, навіть початківцям.

Недоліки:

- документація бібліотеки Allegro може бути менш повною або структурованою порівняно з іншими мультимедійними бібліотеками, що може затруднити початківцям розуміння та використання її можливостей;

- у порівнянні з деякими іншими мультимедійними бібліотеками, такими як SDL або SFML, Allegro може мати меншу кількість користувачів та менш активну спільноту, що може ускладнити отримання допомоги та розв'язання проблем;

- Allegro в основному орієнтований на роботу з 2D графікою, тож для створення 3D ігор або застосунків може знадобитися додаткова бібліотека або фреймворк.

2.6 Порівняння мультимедійних бібліотек

Ознайомившись з описаними вище мультимедійними бібліотеками, було проведено порівняльний аналіз (табл. 2.3).

Таблиця 2.3 – Порівняльна характеристика мультимедійних бібліотек

Характеристика / Назва	SDL	SFML	Allegro
1	2	3	4
Крос-платформенність	Підтримує Windows, macOS, Linux, Android, iOS, та інші	Підтримує Windows, macOS, Linux, iOS, Android та інші	Підтримує Windows, macOS, Linux, Android, iOS, та інші
Мови програмування	C, C++, та інші	C++, з підтримкою для C та інших мов	C, C++, та інші
Простота використання	Простий у використанні, але знадобиться додатковий код для деяких функцій	Легкий у використанні та має простий API	Менш простий у використанні, порівняно з SDL та SFML

Продовження таблиці 2.3

1	2	3	4
Мультимедійні можливості	Надає доступ до аудіо, відео, графіки та інших операцій	Підтримує роботу з 2D графікою, аудіо та відео	Надає функції для роботи з графікою, аудіо та відео
Спільнота користувачів	Активна спільнота користувачів та широкі можливості підтримки	Має активну спільноту користувачів	Є спільнота користувачів, але менш широка порівняно з SDL та SFML
Розширення	Має підтримку великої кількості плагінів та розширень	Підтримує додаткові модулі та розширення, але може бути обмежена	Має певну кількість розширень, але може бути менш гнучкою
3D графіка	Має обмежену підтримку 3D графіки	Може бути інтегрована з OpenGL для 3D графіки	Має базову підтримку 3D графіки

Під час порівняння мультимедійних бібліотек для розробки гри, було визначено, що SDL є найбільш оптимальним вибором для розробки проєкту. SDL надає доступ до різних низькорівневих операцій з відео, аудіо, клавіатурою, мишею та іншими мультимедійними пристроями. Також розробку цієї бібліотеки спонсорує компанія Valve.

2.7 Структурна схема ігрового циклу

У комп'ютерних іграх, включаючи ігри жанру Tower Defense, ігровий цикл є важливою концепцією, що описує послідовність подій та операцій, які відбуваються протягом кожного кадру гри. Для взаємодії ігрових об'єктів між собою, було прийнято рішення створити простий ігровий цикл.

Ігровий цикл працює безперервно під час гри. Кожен поворот циклу обробляє введені користувачем дані без блокування, оновлює стан гри та рендерить гру. Він відстежує плин часу, щоб контролювати швидкість гри (рис. 2.1) [23].

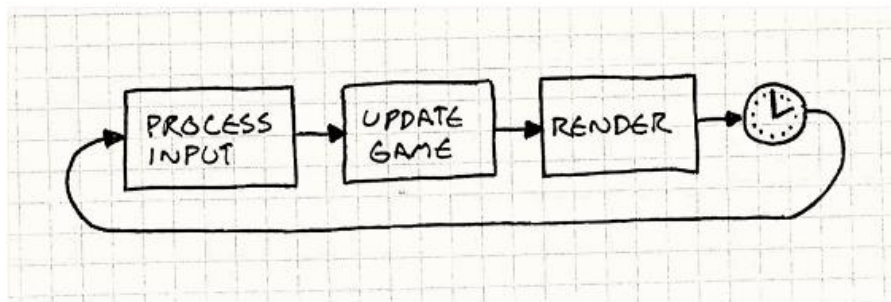


Рисунок 2.1 – Ігровий цикл [23]

Як вже зазначалось у розділі 1.3, для розробки комп'ютерної гри в жанрі Tower Defense було прийнято рішення розробити ігрові об'єкти з компонентами які будуть знаходитися в ігровому циклі та взаємодіяти один з одним.

Ігровий об'єкт буде складатися з компонентів, а компоненти будуть слугувати основними елементами ігрових об'єктів. Самі ігрові об'єкти будуть циркулювати по ігровому циклу, де кожен необхідний метод буде викликатися кожен фрейм. Схема, яка демонструє базовий принцип роботи ігрових об'єктів зображена на рисунку 2.2.

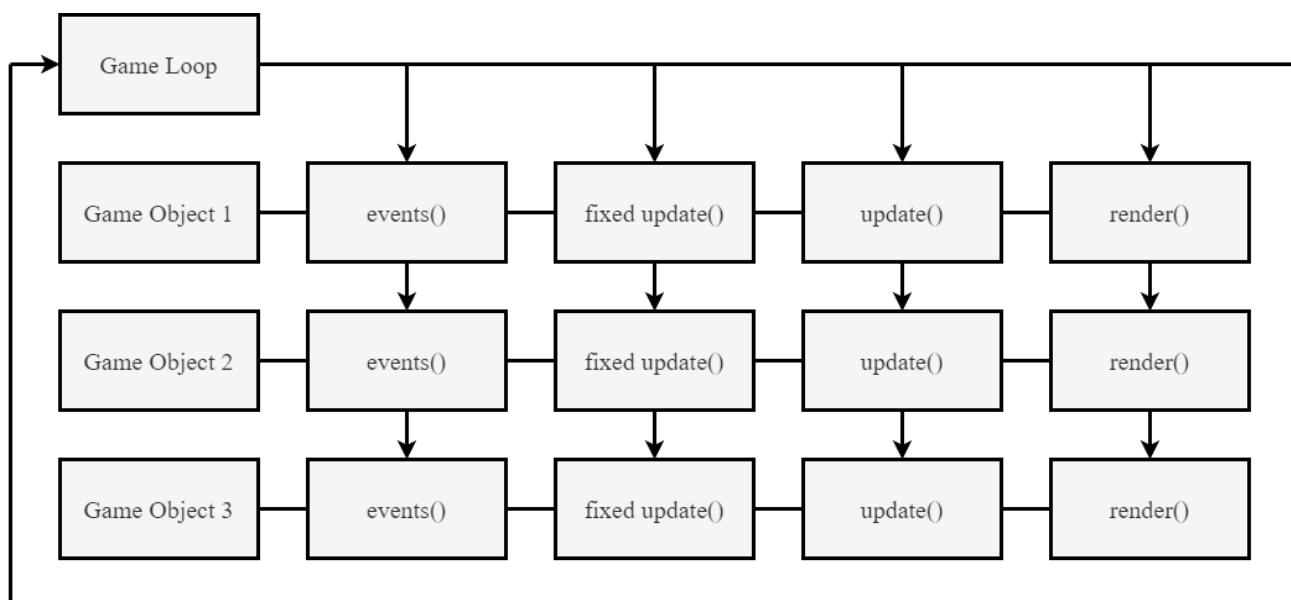


Рисунок 2.2 – Принцип роботи ігрових об'єктів

Діаграма класів (рис. 2.3) була розроблена за допомогою мови UML. Вона відображає ключові класи, які необхідні для функціонування гри.

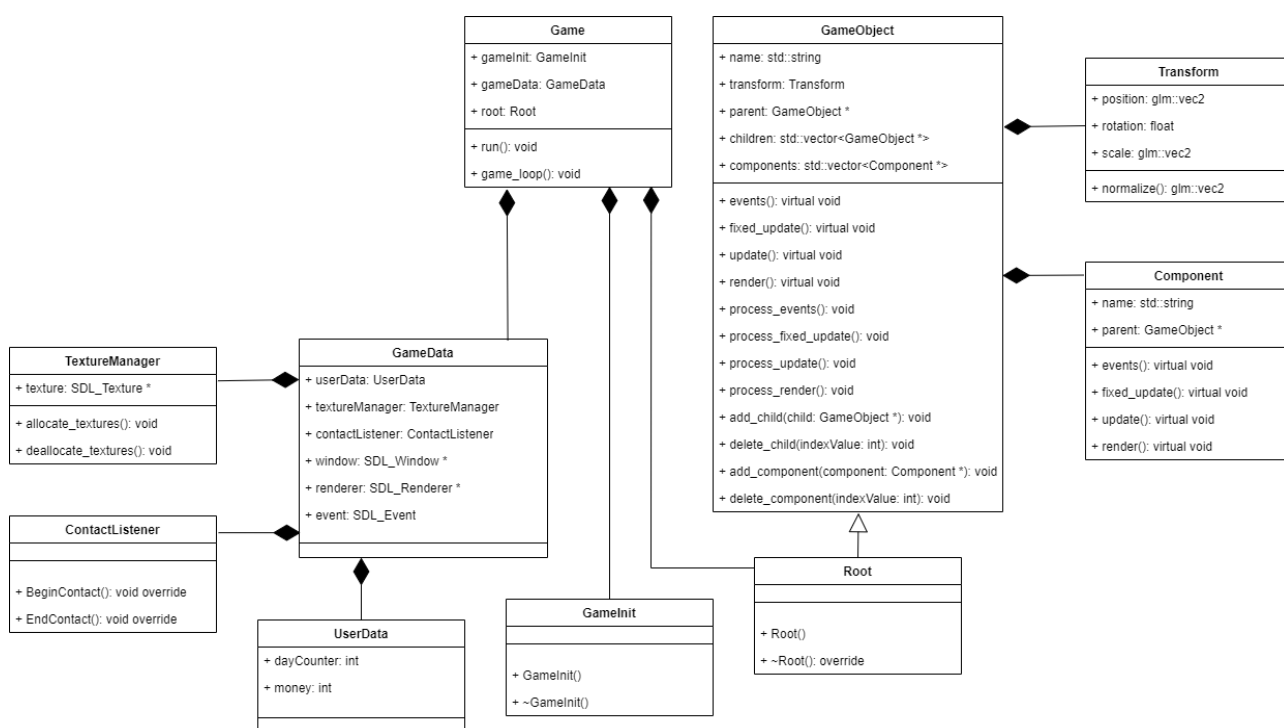


Рисунок 2.3 – Діаграма класів ігрового циклу

Клас Game – є головним класом всієї програми. Він створює екземпляри трьох класів: GameInit, GameData, Root. Реалізовує ігровий цикл,

який життєво необхідний для функціонування всієї гри. Має такі методи:

- run() – запускає гру;
- game_loop() – головний цикл гри.

Клас GameInit – призначений для ініціалізації та вимикання бібліотек.

Клас GameData – є головним класом для зберігання ігрових даних. Має такі необхідні дані, як window, renderer, event, targetFPS та інші. Також створює екземпляри класів: UserData, TextureManager, ContactListener.

Клас UserData – є головним класом для зберігання даних користувача. Взаємодіє з базою даних.

Клас TextureManager – призначений для управління ігровими текстурами. Виділяє та звільнює дані в залежності від дій користувача.

Клас ContactListener – призначений для взаємодії двох фізичний об'єктів, які зіткнулися один з одним.

Клас GameObject – є головним класом для створення ігрових об'єктів. Реалізовує Game Object патерн. Створює екземпляри інших ігрових об'єктів(дітей) і завжди має інформацію щодо одного фіксованого ігрового об'єкта(батька). Також створює екземпляри компонентів. Має такі методи:

- events() – обробляє всі події ігрового об'єкта;
- fixed_update() – обробляє всі фізичні симуляції ігрового об'єкта;
- update() – обробляє всі логічні операції ігрового об'єкта;
- render() – відображає всю графіку ігрового об'єкта;
- process_events() – обробляє всі події кожної дитини ігрового об'єкта;
- process_fixed_update() – обробляє всі фізичні симуляції кожної дитини ігрового об'єкта;
- process_update() – обробляє всі логічні операції кожної дитини ігрового об'єкта;
- process_render() – відображає всю графіку кожної дитини ігрового об'єкта.

Клас Component – є головним класом для створення компонентів. Реалізовує Component патерн. Завжди має інформацію щодо одного

фіксованого ігрового об'єкта(батька). Має такі методи:

- events() – обробляє всі події компонента;
- fixed_update() – обробляє всі фізичні симуляції компонента;
- update() – обробляє всі логічні операції компонента;
- render() – відображає всю графіку компонента.

Клас Root – є першим ігровим об'єктом в усій ієрархії комп'ютерної гри. Єдиний ігровий об'єкт, який не має батька. Ігровий цикл має прямий доступ до цього об'єкту.

Клас Transform – призначений для зберігання таких даних як розташування, поворот, масштаб.

2.8 Висновок

У цьому розділі було проведено аналіз та вибір інструментарію розробки ПЗ та було спроектовано архітектуру майбутньої комп'ютерної гри у жанрі Tower Defense. Було проведено порівняльний аналіз мов програмування для розробки ПЗ та встановлено, що найбільші переваги має мова програмування C++, тому що вона ефективна в розробці ігор, має багатий набір бібліотек та інструментів, що спрощує виконання завдань з розробки патернів, а також має повний контроль над ресурсами.

Було проведено порівняльний аналіз середовищ розробки, в ході чого було встановлено, що найбільші переваги має власне середовище розробки, тому що воно забезпечує повний контроль над інструментами розробки.

Було проведено огляд мультимедійних бібліотек для розробки комп'ютерної гри, та встановлено, що найбільші переваги має бібліотека SDL, тому що вона має багатий функціонал, має широку підтримку та спонсорується такою компанією як Valve. Було описано структурну схему ігрового циклу.

3 РОЗРОБКА ПРОГРАМИ

3.1 Структура проєкту

Багато комп’ютерних ігор створюються за допомогою ігрових рушіїв, які забезпечують широкий спектр інструментів та можливостей для розробників. Проте у даній роботі ставиться мета розробити комп’ютерну гру, використовуючи альтернативні методи, що не базуються на традиційних ігрових рушіях, а саме бібліотеках або фреймворках. Така робота потребує ретельної та уважної організації структури проєкту.

3.1.1 Загальна структура

Загальна структура проєкту представлена на рисунку 3.1.

backup	4/16/2024 10:41 PM	File folder
bin	5/1/2024 7:24 PM	File folder
build	4/16/2024 9:07 PM	File folder
fonts	2/29/2024 4:44 PM	File folder
images	4/7/2024 9:57 PM	File folder
include	4/16/2024 9:01 PM	File folder
info	2/26/2024 11:40 PM	File folder
source	4/16/2024 8:58 PM	File folder
tools	2/19/2024 6:11 AM	File folder

Рисунок 3.1 – Загальна структура проєкту

Директорія backup – призначена для зберігання резервних копій файлів проєкту. Регулярні резервні копії є важливим елементом забезпечення безпеки та можуть бути корисними у випадку втрати даних або несподіваних проблем з розробкою.

Директорія bin – призначена для зберігання виконуваних файлів програми, такі як головний виконуваний файл гри або інші важливі

виконувані файли. Ці файли можуть бути скомпільованими версіями програми або іншими важливими компонентами, необхідними для запуску проєкту.

Директорія `build` – призначена для зберігання файлів, які були створені в процесі компіляції або збірки проєкту. Це може включати проміжні файли, бінарні файли, що виходять з компілятора, а також інші файли, пов'язані з процесом збірки та розгортання.

Директорія `fonts` – призначена для зберігання шрифтів, які використовуються у грі. Шрифти можуть бути використані для відображення тексту на ігровому екрані або в інтерфейсі користувача.

Директорія `images` – призначена для зберігання зображень, які використовуються у грі. Це можуть бути текстури для об'єктів у грі, фонові зображення, іконки, анімації та інші візуальні елементи.

Директорія `include` – призначена для зберігання заголовних файлів, які необхідні для компіляції програмного коду проєкту. Ці файли містять описи функцій, класів та інших складових програми, які використовуються в різних частинах проєкту.

Директорія `info` – призначена для зберігання будь-якої додаткової інформації про проєкт, таку як документація, README файли, або будь-яку іншу важливу інформацію, яка допомагає розробникам зрозуміти або користувачам використовувати програму.

Директорія `source` – призначена для зберігання вихідних файлів програмного коду проєкту. Це може включати файли реалізації класів, модулів, скрипти і будь-який інший вихідний код, необхідний для розробки комп'ютерної гри.

Директорія `tools` – призначена для зберігання допоміжних інструментів, які використовуються під час розробки або тестування проєкту. Це може бути програмне забезпечення для автоматизації процесу збірки, інструменти для аналізу коду, відлагоджувальні засоби та інше.

3.1.2 Процес створення програмного забезпечення

Директорія tools має два необхідних файли, які забезпечують зручну розробку програмного забезпечення.

Файл `build_and_run.bat` - це скриптовий файл для системи Windows, який використовується для автоматизації процесу компіляції програмного коду та запуску програми після збирання. Цей файл містить команди для запуску Makefile та виконання самої програми. Якщо Makefile видав помилку, то виконання програми не відбудеться. Файл `build_and_run.bat` використовується в розробці програмного забезпечення для швидкого запуску та тестування програми після внесення змін у вихідний код. Це дозволяє ефективніше виконувати цикли розробки та тестування, оскільки можна швидко перевірити, чи працює програма з новими змінами.

Файл Makefile - це текстовий файл, який містить інструкції для програми make, яка використовується для автоматизації процесу збирання (компіляції) програмного коду. У файлі Makefile вказуються правила для збирання програмного коду з вихідних файлів у виконавчі файли та бібліотеки.

Зазвичай у Makefile визначаються цілі (targets), правила (rules) та залежності (dependencies). Цілі - це виконавчі файли або завдання, які можуть бути викликані за допомогою програми make. Правила вказують, як створити ці цілі з вихідних файлів, а залежності вказують, які файли потрібно використовувати для створення цільового файлу [24].

Makefile дозволяє розробникам ефективно управляти процесом збирання проєкту, визначаючи правила компіляції та лінування програмного коду залежно від потреб проєкту. Це робить його важливим інструментом у процесі розробки програмного забезпечення, особливо для проєктів, які складаються з багатьох файлів або бібліотек [24].

Коли відбуваються якісь зміни в коді програми, треба запустити bat скрипт, який в свою чергу запускає програму make. Якщо немає жодних

помилки під час компілювання та лінування, то далі bat скрипт запускає готову програму.

3.2 Структура програмного забезпечення

Директорії `include` та `source` мають всі основні файли з кодом програмного забезпечення. Загальна структура директорії `source` представлена на рисунку 3.2.

 components	4/8/2024 8:58 PM	File folder
 enemies	4/14/2024 8:42 PM	File folder
 exitRoom	4/16/2024 9:02 PM	File folder
 gameRoom	4/7/2024 6:48 PM	File folder
 infoRoom	4/16/2024 8:51 PM	File folder
 main	3/18/2024 7:21 PM	File folder
 mainMenu	3/28/2024 9:26 PM	File folder
 towerDefense	4/14/2024 5:27 PM	File folder
 tutorialRoom	4/16/2024 8:03 PM	File folder
 UI	4/6/2024 12:03 AM	File folder
 units	4/15/2024 10:49 PM	File folder

Рисунок 3.2 – Загальна структура директорії `source`

3.2.1 Основні елементи

Як вже зазначалось у розділі 2.7, основними ігровими елементами будуть слугувати ігрові об'єкти з компонентами. Клас `Game` буде слугувати ядром програми та відповідати за всі процеси, що відбуваються впродовж виконання програми. Також цей клас відповідає за створення головного циклу гри, де будуть оброблятися вхідні дані користувача, фізичні симуляції, анімація та рендеринг.

Клас `GameData` слугує сховищем для основних даних всієї гри. Кожен

ігровий об'єкт має прямий доступ до об'єкта цього класу. GameData відповідає за зберігання таких даних:

- інформація щодо вікна програми;
- інформація щодо рендера програми;
- інформація щодо поточних подій;
- час, що минув між кадрами;
- час, що минув між фізичними симуляціями;
- стан головного циклу;
- дані користувача;
- лічильник пройдених днів;
- лічильник ворогів.

Клас Root є першим і єдиним ігровим об'єктом, у якого немає батька. Забезпечує створення та видалення головного меню. Також забезпечує створення та видалення головного ігрового режиму – Tower Defense. Має такі методи:

- create_main_menu() – створення головного меню;
- delete_main_menu() – видалення головного меню;
- create_tower_defense() – створення головного режиму;
- delete_tower_defense() – видалення головного режиму.

3.2.2 Компоненти

Кожен компонент виконує певну роль у функціонуванні гри і може бути відповідальним за різні аспекти геймплею, візуальні ефекти, штучний інтелект, фізичне моделювання. Діаграма класів основних компонентів зображена на рисунку 3.3.

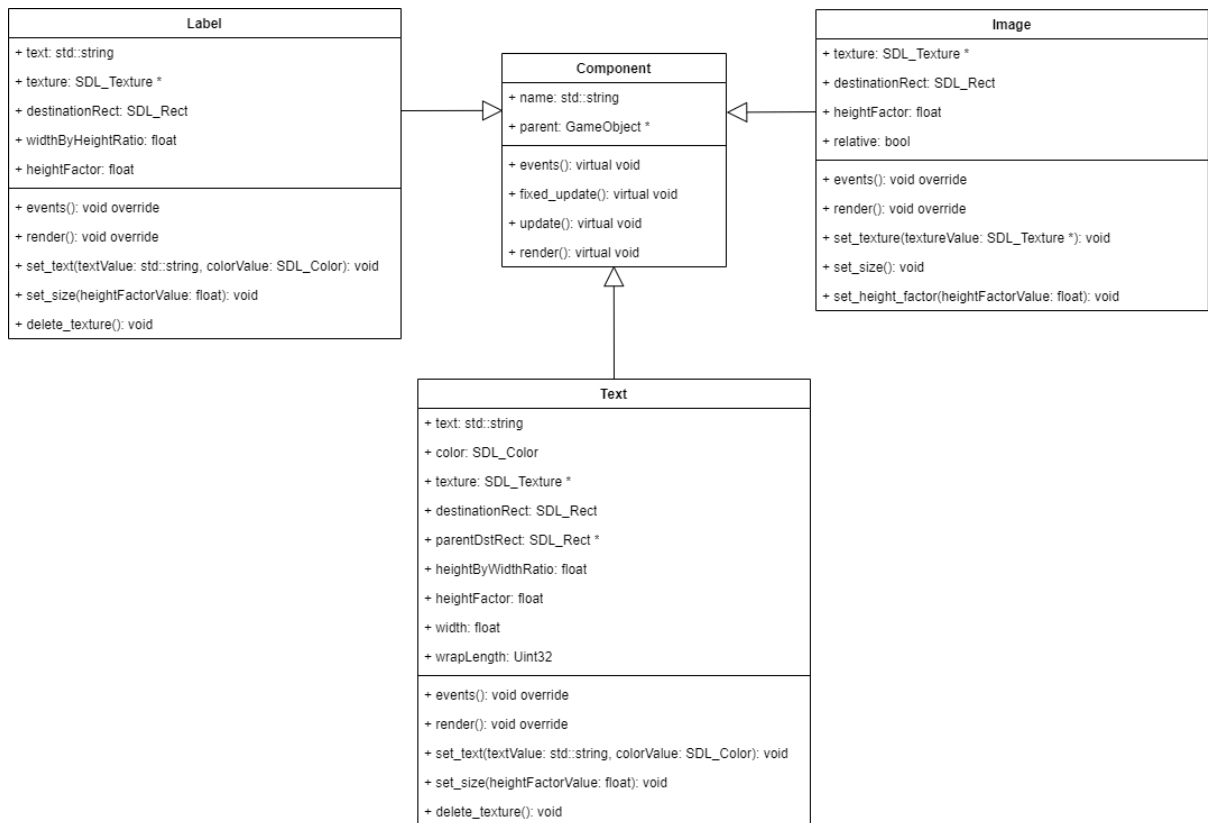


Рисунок 3.3 – Діаграма класів основних компонентів

Клас **Label** – призначений для створення простого тексту. Має особливість адаптуватися під будь-які розміри екранів. Має такі методи:

- **events()** – перевизначає наслідуваний метод з класу **Component** для власних потреб ігрових подій;
- **render()** – перевизначає наслідуваний метод з класу **Component** для власних потреб відображення графіки;
- **set_text()** – призначає новий текст з новим кольором;
- **set_size()** – встановлює розміри тексту;
- **delete_texture()** – видаляє текстуру тексту.

Клас **Text** – призначений для створення тексту, який здатен переноситися на інші рядки. Має особливість адаптуватися під будь-які розміри екранів. Має такі методи:

- **events()** – перевизначає наслідуваний метод з класу **Component** для власних потреб ігрових подій;
- **render()** – перевизначає наслідуваний метод з класу **Component** для

власних потреб відображення графіки;

- `set_text()` – призначає новий текст з новим кольором;
- `set_size()` – встановлює розміри тексту;
- `delete_texture()` – видаляє текстуру тексту.

Клас `Image` – призначений для створення рисунка. Має особливість адаптуватися під будь-які розміри екранів. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Component` для власних потреб ігрових подій;
- `render()` – перевизначає наслідуваний метод з класу `Component` для власних потреб відображення графіки;
- `set_texture()` – призначає нову текстуру для відображення рисунка;
- `set_size()` – встановлює розміри рисунка;
- `set_height_factor()` – призначає коефіцієнт адаптації рисунка.

3.2.3 Головне меню

Головне меню гри - це важлива частина геймплею, яка дозволяє гравцеві керувати грою перед початком, під час і після гри. Воно створює зручну інтерфейсну точку доступу до всіх основних функцій та опцій гри. Діаграма класів головного меню зображена на рисунку 3.4.

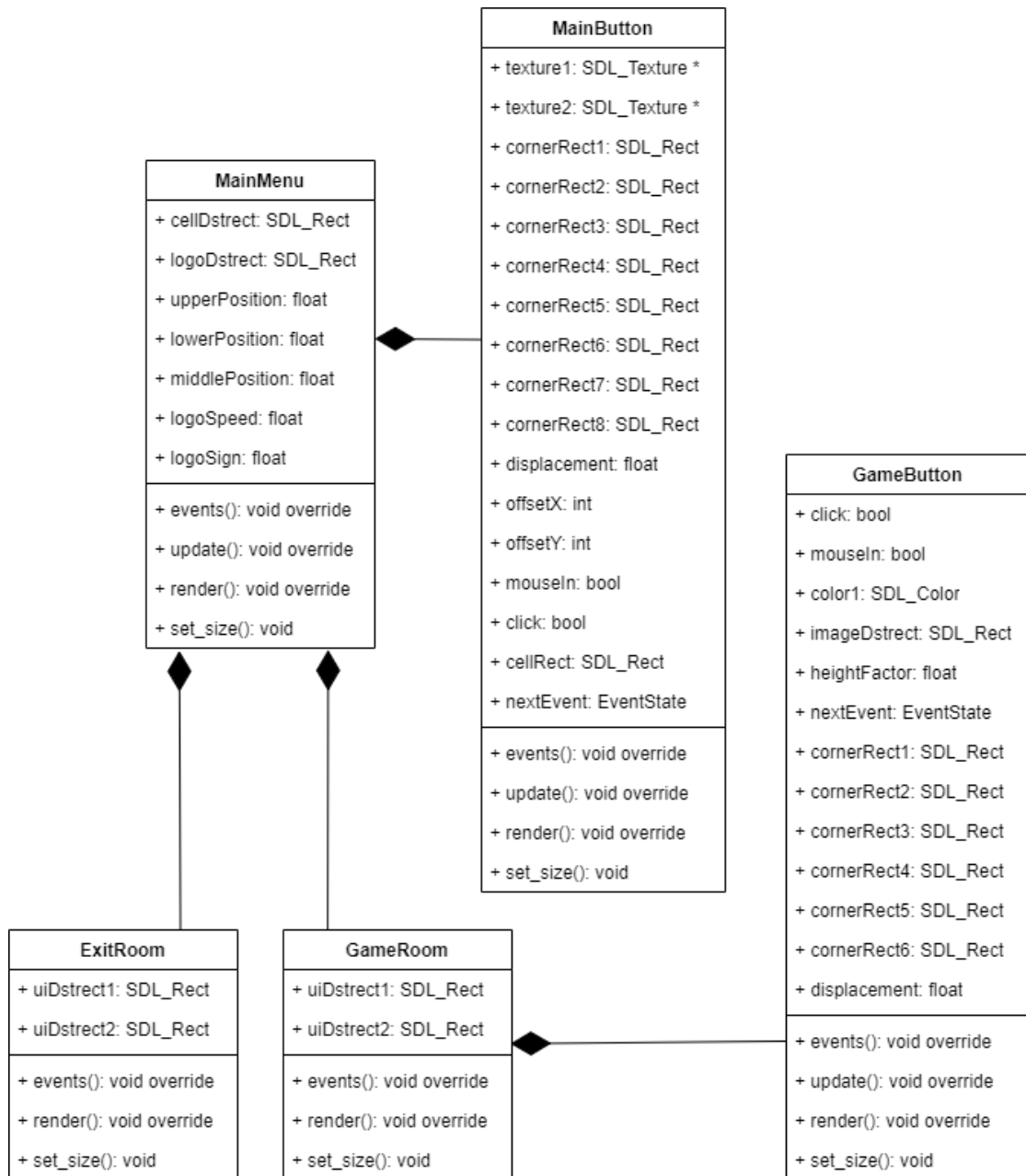


Рисунок 3.4 – Діаграма класів головного меню

Клас MainMenu – головне меню гри. Створюється за допомогою класу Root. Має чотири кнопки – об'єкти класу MainButton. Реалізує відображення та анімацію головного лого гри. Має два порожніх ігрових об'єкта, які використовують компонент Label для відображення заголовка гри. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- update() – перевизначає наслідуваний метод з класу Game Object для

власних потреб ігрової логіки;

- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;

- `set_size()` – адаптує головне лого під різні розміри моніторів.

Клас `MainButton` – призначений для створення головної кнопки. Після натискання, клас `MainMenu` створює один із чотирьох можливих класів: `GameRoom`, `TutorialRoom`, `InfoRoom`, `ExitRoom`. За допомогою бібліотек `SDL` та `GLM`, даний клас реалізовує просту анімацію отримання та втрати фокусу кнопки. Також даний клас використовує компонент `Image` для відображення спрайту кнопки. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрових подій;

- `update()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрової логіки;

- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;

- `set_size()` – адаптує кнопку під різні розміри моніторів.

Клас `GameRoom` – призначений для створення ігрової кімнати. Даний клас є відправною точкою для основної гри – режиму `Tower Defense`. Також даний клас використовує компонент `Label` для відображення заголовка. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрових подій;

- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;

- `set_size()` – адаптує елементи інтерфейсу ігрового об'єкта під різні монітори.

Клас `GameButton` – призначений для створення кнопки запуску гри. Після натискання, клас `Root` видаляє об'єкт класу `MainMenu` та створює клас `TowerDefense`. За допомогою бібліотек `SDL` та `GLM`, даний клас реалізовує

просту анімацію отримання та втрати фокусу кнопки. Також даний клас використовує компонент Label для відображення тексту кнопки. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- update() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрової логіки;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;
- set_size() – адаптує кнопку під різні розміри моніторів.

Клас ExitRoom – призначений для створення кімнати для виходу з гри. Даний клас реалізовує вихід та завершення гри. Також даний клас використовує компонент Label для відображення заголовка. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;
- set_size() – адаптує елементи інтерфейсу ігрового об’єкта під різні монітори.

3.2.4 Захист веж

У режимі Tower Defense основною механікою гри буде розміщення різних видів веж на мапі, які автоматично атакують ворогів, коли ті наближаються до них. Гра буде пропонувати гравцям стратегічне використання накопичених грошей та розміщення оборонних споруд для забезпечення ефективного захисту бази. Гравець буде сам вирішувати, коли йому викликати нову хвилю ворогів. Діаграма класів реалізації режиму Tower Defense зображена на рисунку 3.5.

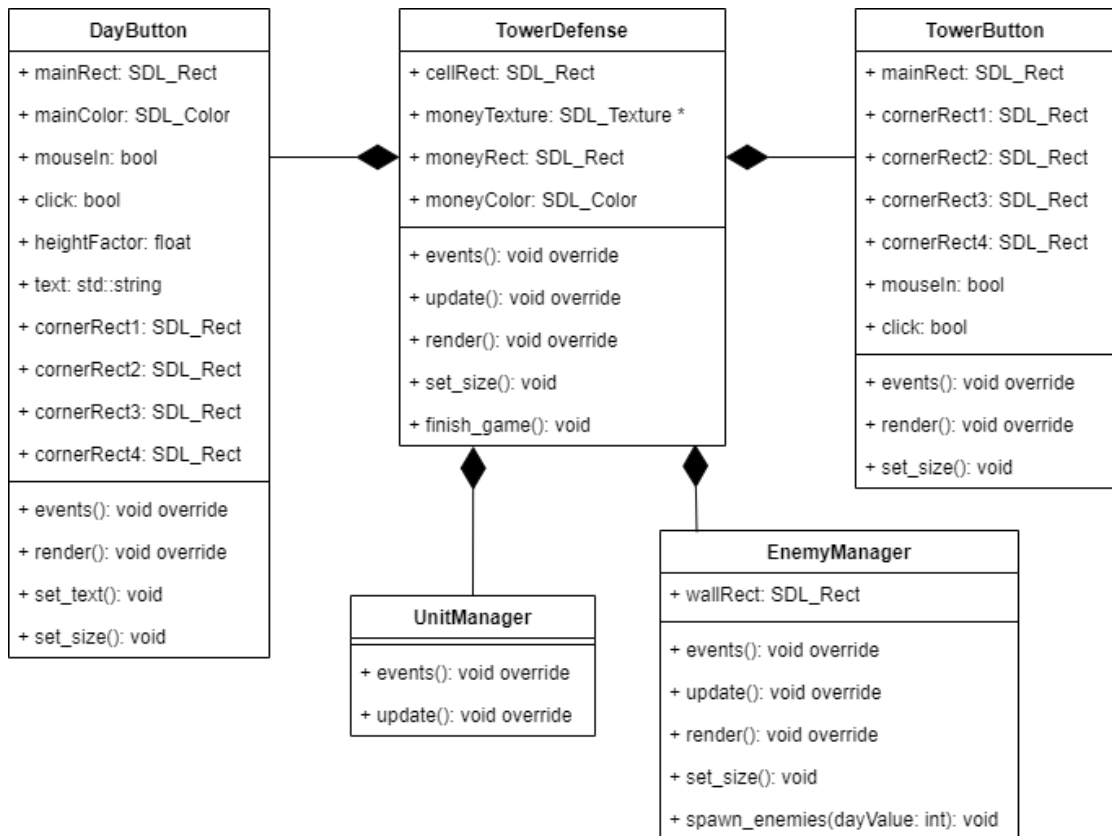


Рисунок 3.5 – Діаграма класів реалізації режиму Tower Defense

Клас TowerDefense – головний клас режиму Tower Defense. Реалізовує початок та завершення основної гри. Відображає кількість грошей, які заробив гравець вбиваючи ворогів. Також створює кордони ігрового світу використовуючи бібліотеку Box2d. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- update() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрової логіки;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;
- finish_game() – створює вікно для завершення гри;
- set_size() – адаптує елементи інтерфейсу ігрового об’єкта під різні монітори.

Клас TowerButton – кнопка магазину. Потрапляючи в магазин, гравець може купити одну з 4 веж на вибір, якщо в нього достатньо грошей.

Придбавши вежу, гравець може поставити її де завгодно для оборони головної бази. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрових подій;
- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;
- `set_size()` – адаптує кнопку під різні монітори.

Клас `DayButton` – кнопка для виклику ворогів. Якщо гравець впорався з хвилями ворогів, то лічильник дня збільшується на одиницю. Якщо гравець не впорався з ворогами, то це – кінець гри. У кінці гри лічильник пройдених днів заноситься до БД, якщо він відображає більш кращий результат, ніж значення у БД. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрових подій;
- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;
- `set_text()` – оновлює текст;
- `set_size()` – адаптує кнопку під різні монітори.

Клас `EnemyManager` – призначений для виклику ворогів. Кількість та тип кожного ворога є фіксованою змінною. Має такі методи:

- `events()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрових подій;
- `update()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб ігрової логіки;
- `render()` – перевизначає наслідуваний метод з класу `Game Object` для власних потреб відображення графіки;
- `spawn_enemies()` – створює хвилю ворогів.

Клас `UnitManager` – призначений для створення веж та головного персонажа. Всі союзні ігрові об'єкти мають своє фіксоване значення НР і якщо ворог зменшить його до нуля, то союзний ігровий об'єкт помре і не

буде здатен функціонувати далі. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- update() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрової логіки.

3.2.5 Вежі

У режимі Tower Defense вежі потрібні для того, щоб захистити хвилі ворогів. Діаграма класів основних веж зображена на рисунку 3.6.

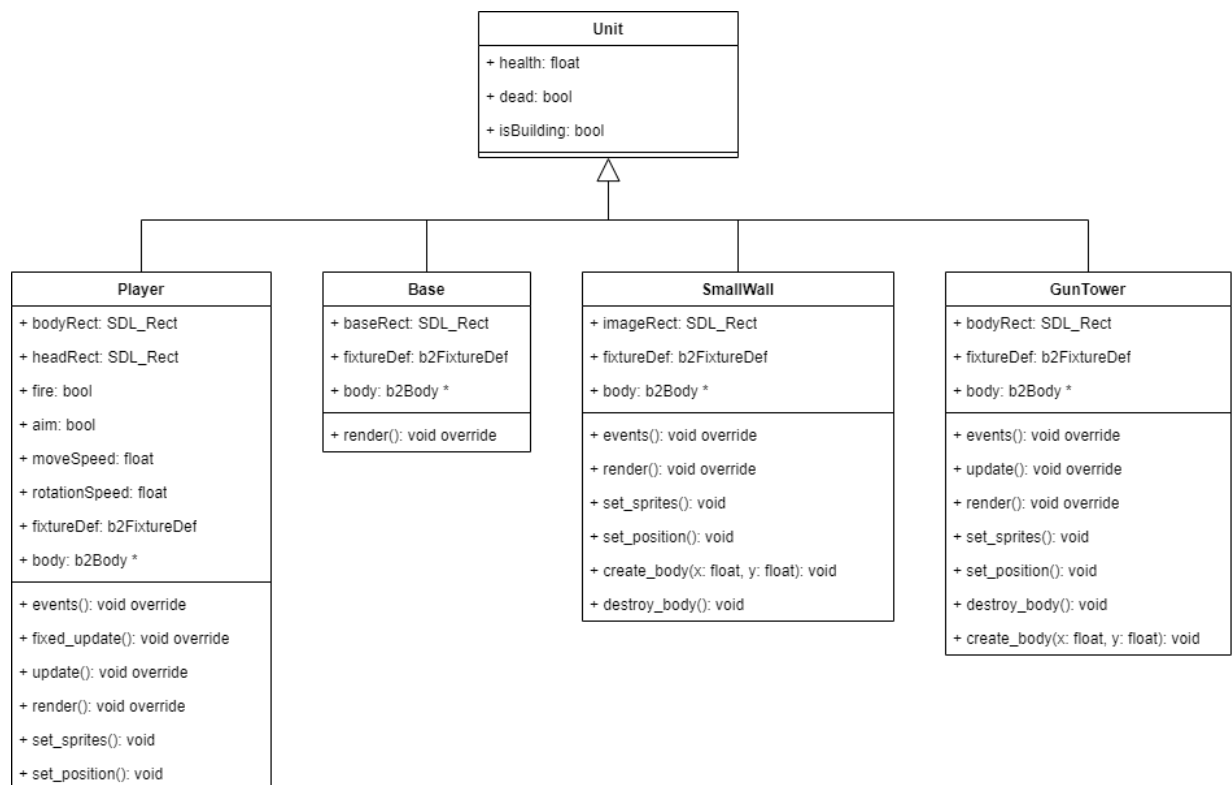


Рисунок 3.6 – Діаграма класів основних веж

Клас **Unit** – базовий клас для союзних ігрових об’єктів. Призначений для надання загальних параметрів союзним ігровим об’єктам.

Клас **Player** – головний персонаж, яким може керувати гравець. Реалізовує базову механіку переміщення та повороту. Здатен стріляти у ворогів. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- fixed_update() – перевизначає наслідуваний метод з класу Game Object для власних потреб фізичних симуляцій;
- update() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрової логіки;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;
- set_sprites() – надає нові координати спрайтам персонажа;
- set_position() – надає нові координати розміщення персонажа.

Клас Base – головна база, яку гравець має захищати. Якщо база буде знищена, то гра буде закінчена.

Клас SmallWall – захисна вежа. Відіграє роль стіни. Гравець здатен побудувати вежу у довільному просторі. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;
- set_sprites() – надає нові координати спрайтам вежі;
- set_position() – надає нові координати розміщення вежі;
- create_body() – створює нове фізичне тіло для вежі;
- destroy_body() – видаляє фізичне тіло вежі.

Клас GunTower – атакуюча вежа. Відіграє роль як стіни, так і зброї для атаки ворогів. Стріляє так само, як і головний персонаж. Має такі методи:

- events() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрових подій;
- update() – перевизначає наслідуваний метод з класу Game Object для власних потреб ігрової логіки;
- render() – перевизначає наслідуваний метод з класу Game Object для власних потреб відображення графіки;

- `set_sprites()` – надає нові координати спрайтам вежі;
- `set_position()` – надає нові координати розміщення вежі;
- `create_body()` – створює нове фізичне тіло для вежі;
- `destroy_body()` – видаляє фізичне тіло вежі.

3.3 Інтерфейс

Ігровий інтерфейс - це спосіб взаємодії гравця з грою через елементи, які відображаються на екрані. Це може включати такі елементи, як кнопки, меню, індикатори здоров'я та енергії, карти, характеристики персонажів та інше. Інтерфейс гри надає гравцеві необхідну інформацію і забезпечує зручність управління грою. Він є важливою частиною геймплею, оскільки може впливати на зручність гри та загальні враження від ігрового процесу.

3.3.1 Рендеринг

Відображення графіки реалізується за допомогою спрайтів та техніки Immediate drawing. За допомогою цих двох методів створюються всі графічні елементи у грі. Всі спрайти створюються в програмі Inkscape. Наприклад, головне лого гри було створено саме в цій програмі (рис. 3.7).

Inkscape — це безкоштовний редактор векторної графіки, який пропонує функціональні можливості, схожі з такими програмами, як Illustrator, Freehand, CorelDraw та Xara Xtreme. Програма ліцензована відповідно до GNU General Public License. Вона побудована на базі бібліотек для створення графічного інтерфейсу GTK+ і застосунку для розробки програм мовою C++ до цієї бібліотеки (gtkmm) [25].

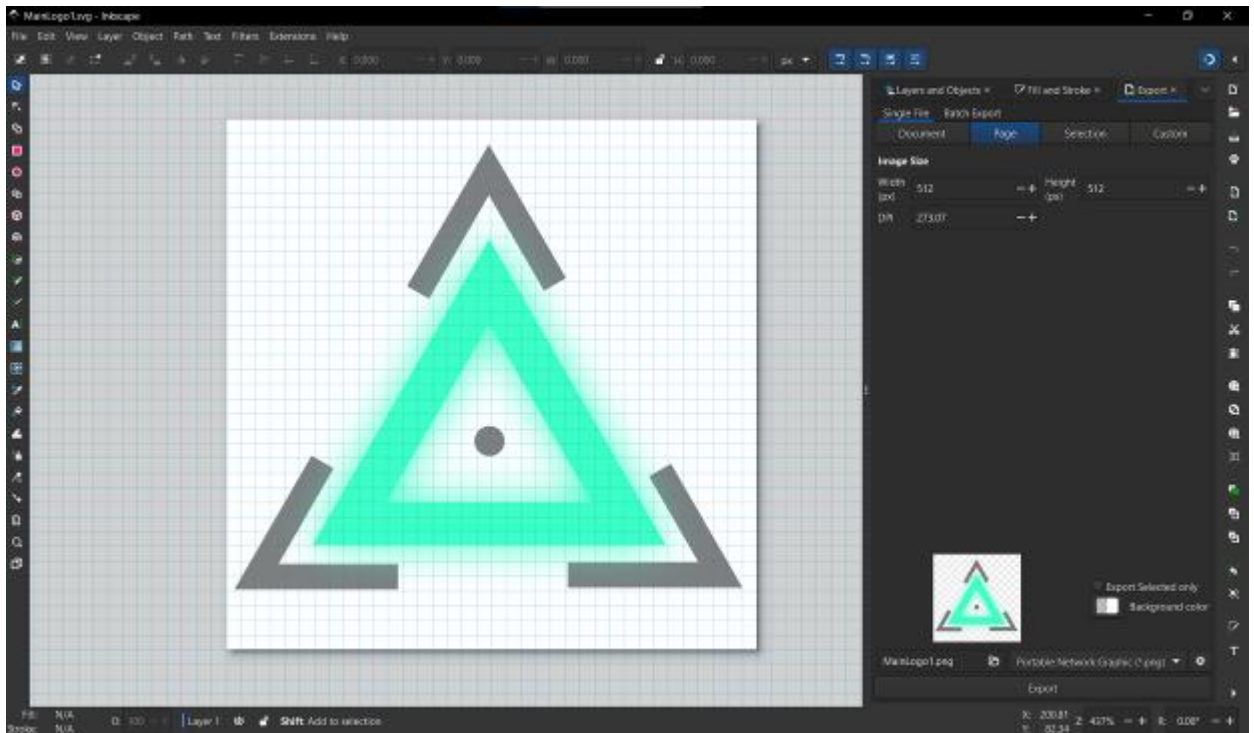


Рисунок 3.7 – Головне лого гри

В даній роботі не була поставлена ціль напряму використовувати графічний API, тому через це не можливо створювати будь-які візуальні ефекти за допомогою шейдерів. Щоб обійти цей недолік, було прийнято рішення створити несправжній візуальний ефект за допомогою програми Inkscape. За допомогою інструмента Blur можна зробити розмиття пікселів на екрані. Таким чином вдалося створити ефект світіння, але цей ефект фіксований, не має ніякої динамічної анімації, не реагує якимось чином на дії користувача, тощо.

Immediate drawing у програмуванні ігор - це підхід до рендерингу, при якому графічні об'єкти малюються безпосередньо в кожному кадрі. Це означає, що замість збереження об'єктів у структурі даних для подальшого використання, рендерингова функція безпосередньо викликає команди малювання кожного об'єкта в поточному кадрі. У даній роботі цей підхід має більше переваг, ніж використання спрайтів, бо Immediate drawing дуже легко реалізувати та його можна швидко змінити або видалити. Тому для створення інтерфейсу в цьому проєкті, цей підхід використовувався більше, ніж

використання спрайтів.

Бібліотека SDL надає декілька корисних функцій для реалізації Immediate drawing:

- `SDL_RenderDrawLine()` – малює лінію, яка включає обидві кінцеві точки;
- `SDL_RenderDrawLines()` – малює ряд з'єднаних ліній;
- `SDL_RenderDrawPoint()` – малює точку;
- `SDL_RenderDrawPoints()` – малює декілька точок;
- `SDL_RenderDrawRect()` – малює прямокутник;
- `SDL_RenderDrawRects()` – малює декілька прямокутників.

Використовуючи дані функції можна створити будь-які геометричні примітиви (рис. 3.8).

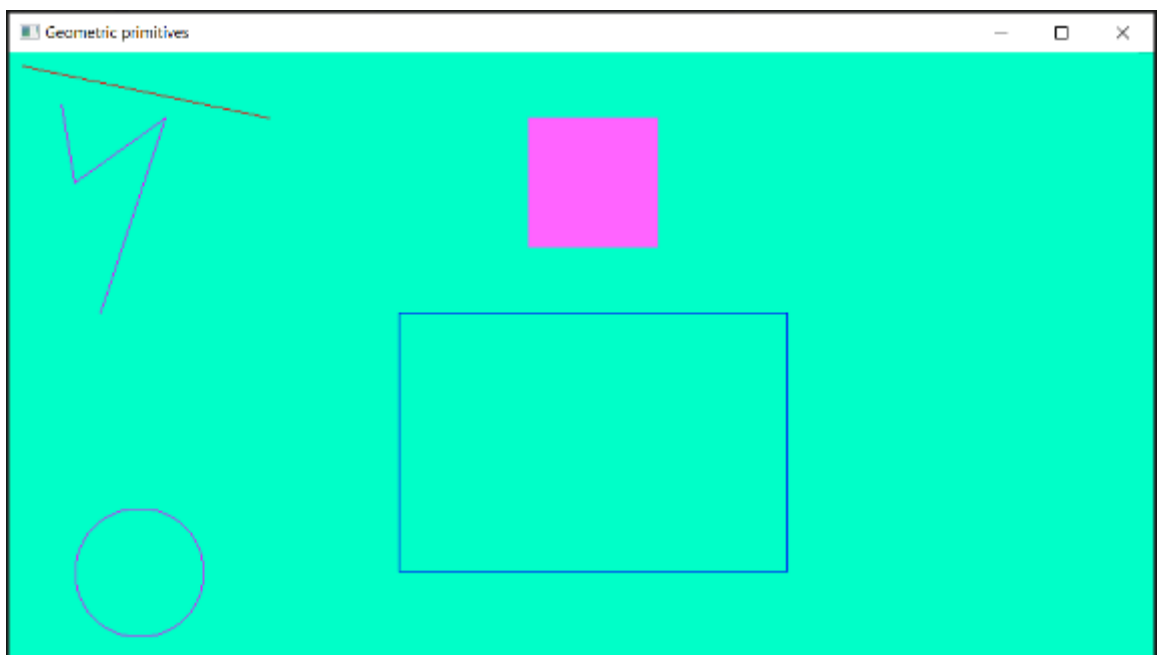


Рисунок 3.8 – Геометричні примітиви

3.3.2 Анімація

Анімація у програмуванні ігор — це процес створення ілюзії руху та зміни станів об'єктів у грі шляхом послідовного відображення серії зображень або кадрів. Вона використовується для додавання динаміки та

життєвості в ігровий світ, роблячи його більш реалістичним та захопливим.

Інтерфейс також не є виключенням. Анімація в ігровому інтерфейсі (UI-анімація) є важливим елементом, який покращує користувацький досвід, робить взаємодію з грою більш інтуїтивною і приємною, а також додає естетичної привабливості. UI-анімація може використовуватися для різних цілей, включаючи надання зворотного зв'язку користувачеві, привернення уваги до важливих елементів, та створення більш динамічного і живого інтерфейсу.

В даній роботі були реалізовані три види анімації, де з двома з них користувач може взаємодіяти.

Перша анімація відносно проста. Вона стимулює головне лого гри переміщуватися вниз та вгору. Таким чином дана анімація реалізує ефект пінг понгу. Створюється така анімація за допомогою трьох змінних: початковий поріг, кінцевий поріг та середнє значення, де середнє значення постійно змінюється від початкового порогу до кінцевого і навпаки (рис. 3.9). Всі математичні обчислення допомагає зробити бібліотека GLM.

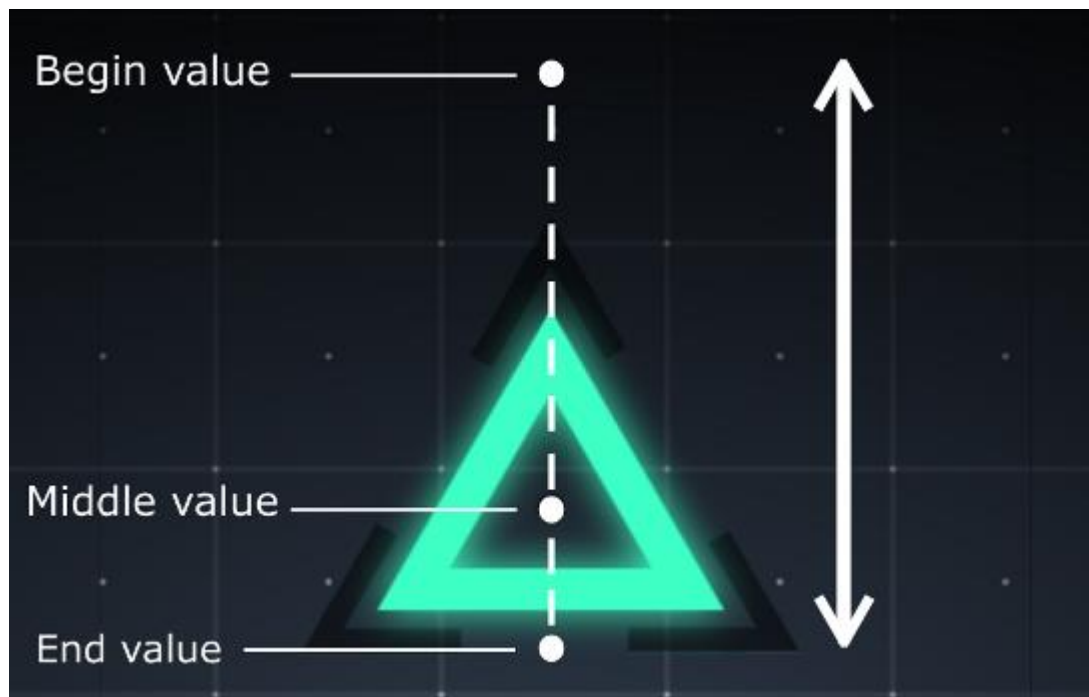


Рисунок 3.9 – Перша анімація

Друга анімація потребує взаємодії від користувача. Коли гравець наводить курсор миші на головну кнопку, то спрайт кнопки змінюється на інший, а її границі переміщуються на фіксовану відстань (рис. 3.10). Після того, як гравець прибрав мишу з кнопки, спрайт змінюється на початковий, а границі повертаються на початкове положення. Бібліотека SDL надає зручну функцію для перевірки наявності миші в середині кнопки - `SDL_PointInRect()`.

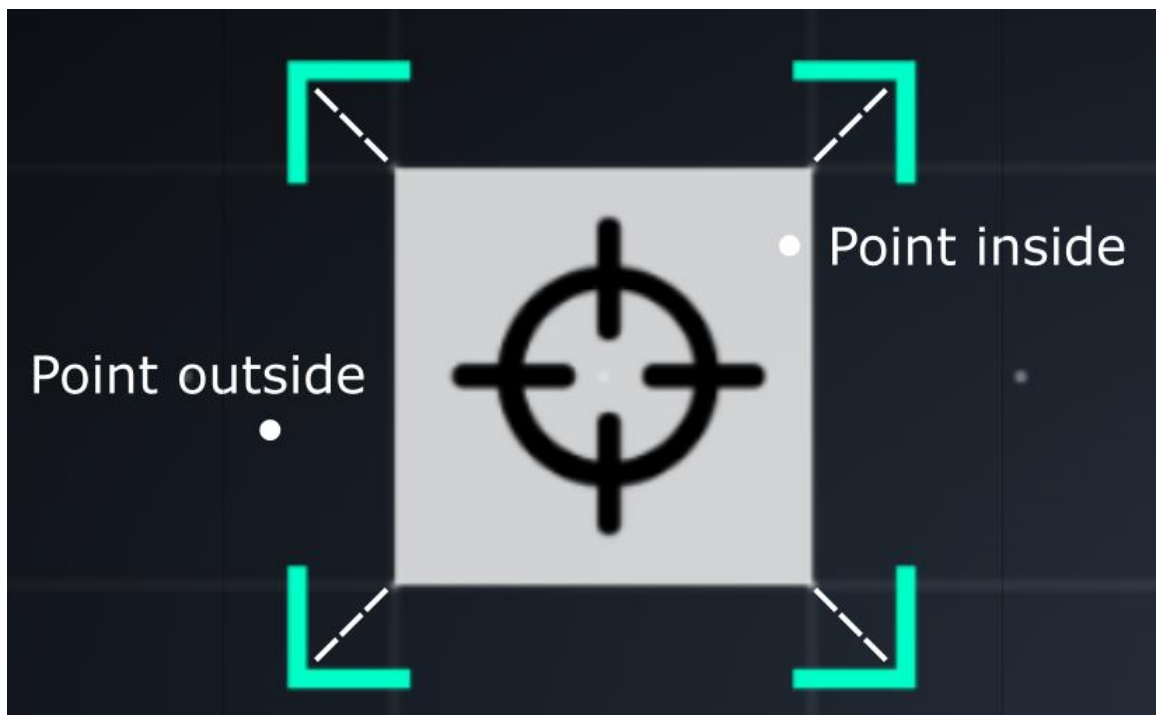


Рисунок 3.10 – Друга анімація

Третя анімація має комбінований стиль першої та другої. У кнопки Back Button відбувається обертання границі кнопки по осі z. Якщо гравець наводить курсор миші на кнопку, то обертання змінює свій напрямок на протилежний. Так само напрямок обертання змінюється коли гравець прибирає мишу з кнопки. Функція `SDL_GetMouseState()` з бібліотеки SDL допомагає отримати координати миші. Функція `clamp()` з бібліотеки GLM допомагає встановити границі обертання для того, щоб не відбувся overflow змінної.

3.3.3 Адаптація

Інтерфейс користувача може автоматично підлаштовуватися під різні типи пристроїв, екранів і роздільних здатностей, забезпечуючи зручний і оптимальний користувацький досвід незалежно від платформи. Адаптованість інтерфейсу включає зміну розміру, розташування та вигляду елементів UI для досягнення найкращої зручності та функціональності.

В даній роботі абсолютно всі UI елементи адаптовані під різні розміри екранів. Таке можливо завдяки масштабуванню та переміщенню елементів у відсотках, а не у фіксованій кількості пікселів. Бібліотека SDL дає можливість встановити фіксовані розміри вікна програми. За цими даними можна встановити необхідний розмір будь-якого UI елемента помноживши розміри вікна на відповідний відсоток (рис. 3.11).

Таким чином можна забезпечити зручність і доступність інтерфейсу для всіх користувачів, незалежно від розміру екранів. Створення адаптованого інтерфейсу у програмуванні ігор є ключовим аспектом, що забезпечує високоякісний користувацький досвід та робить гру доступною для максимально широкої аудиторії.

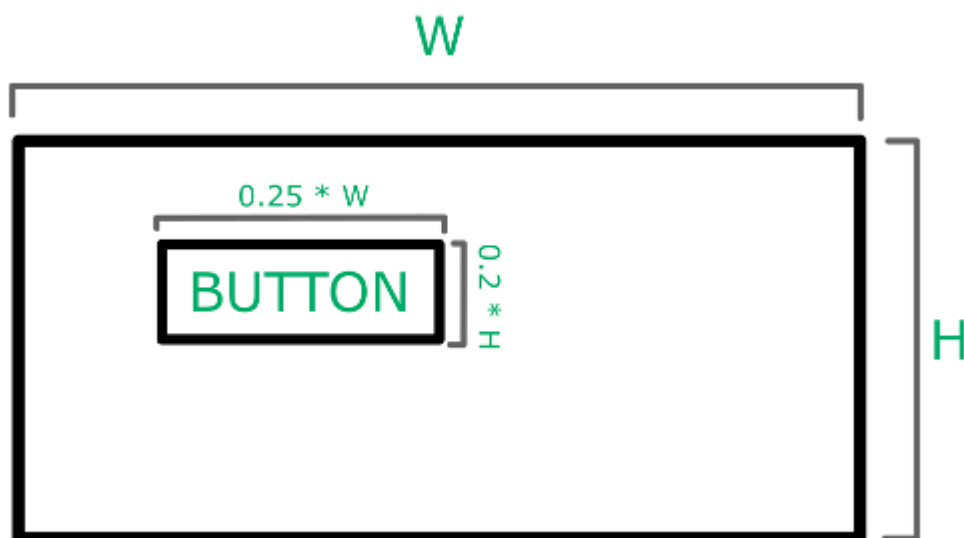


Рисунок 3.11 – Адаптованість інтерфейсу

3.4 Висновок

Розроблено просту структуру проєкту та інструменти розробки, за допомогою яких легко та швидко додавати або видаляти нові елементи системи. Makefile дозволяє швидко змінювати та компілювати новий код для комп'ютерної гри.

Створено три компоненти, які полегшують розробку тривіальних елементів гри. Більшість ігрових об'єктів використовують їх для власних потреб.

Розроблено головне меню, яке реалізує зручний інтерфейс для переходу із одного ігрового вікна до іншого.

Зроблено пояснення щодо тонкощів ігрового інтерфейсу.

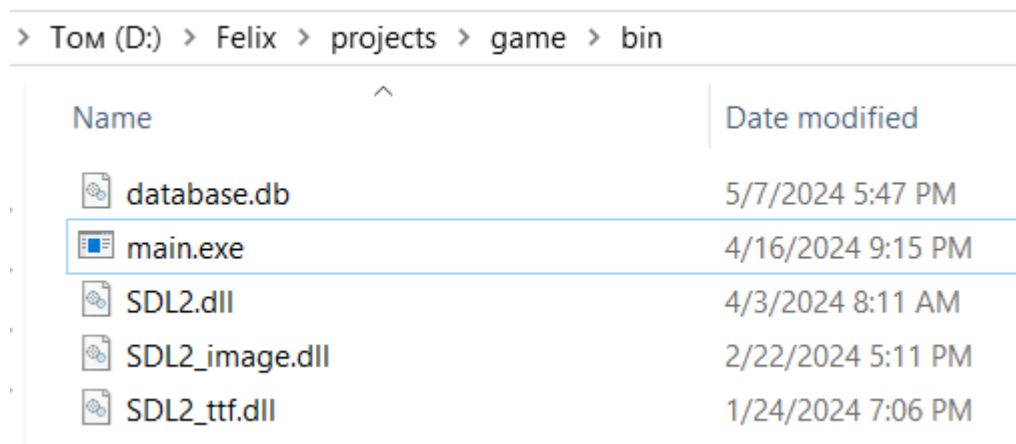
Реалізовано основний ігровий режим, де гравець може будувати захисні вежі, атакуючі вежі та керувати основним персонажем, який здатен знищувати ворогів.

Всі основні ігрові об'єкти були відображені в діаграмах за допомогою мови UML.

4 ЕКСПЛУАТАЦІЯ ПРОГРАМИ

4.1 Інструкція щодо запуску гри

Для того, щоб програма повноцінно функціонувала, потрібно зайти у директорію bin, та запустити файл main.exe (рис. 4.1). Якщо користувач грає в перший раз, то програма створить невелику локальну базу даних для зберігання результатів гри. Якщо не в перший, то програма автоматично відкриє існуючу базу даних для отримання результатів.



> Том (D:) > Felix > projects > game > bin	
Name	Date modified
database.db	5/7/2024 5:47 PM
main.exe	4/16/2024 9:15 PM
SDL2.dll	4/3/2024 8:11 AM
SDL2_image.dll	2/22/2024 5:11 PM
SDL2_ttf.dll	1/24/2024 7:06 PM

Рисунок 4.1 – Директорія bin

Також необхідно пам'ятати про динамічні бібліотеки(dll файли). Їх ні в якому разі не потрібно кудись переміщати або взагалі видаляти. Якщо видалити ці файли, то програма не буде функціонувати як треба або не буде функціонувати взагалі.

4.2 Системні вимоги

Мінімальні системні вимоги:

- операційна система: Windows 7;
- процесор: Intel Core i3;
- оперативна пам'ять: 4 Гб;
- відеопам'ять: 256 Мб;

- місце на жорсткому диску: 10 Кб.

Рекомендовані системні вимоги:

- операційна система: Windows 10;
- процесор: Intel Core i5;
- оперативна пам'ять: 8 Гб;
- відеопам'ять: 512 Мб;
- місце на жорсткому диску: 20 Кб.

4.3 Функціонал системи

Комп'ютерна гра призначена для розваги користувачів.

Програма має наступний функціонал:

- швидкий запуск гри;
- зручне головне меню;
- зручний та адаптований під різні монітори ігровий інтерфейс;
- управління головним персонажем, який здатен атакувати ворогів;
- захист основної бази гравця;
- будівництва захисних та атакуючих веж;
- хвилі ворогів;
- кінець гри, який відображає результати гравця.

4.4 Опис застосування програми

Коли користувач запускає комп'ютерну гру, то першим, що він побачить буде головне меню (рис. 4.2).



Рисунок 4.2 – Головне меню

У головному меню присутні 4 основні кнопки, заголовок гри та анімоване лого. Користувач може побачити анімацію кнопки, якщо він наведе курсор на неї (рис. 4.3).



Рисунок 4.3 – Анімація головної кнопки

Перша кнопка відкриває вікно під назвою Game Room (рис. 4.4). У

даному вікні можна побачити кнопку закриття вікна та кнопку запуску основного ігрового режиму – Tower Defense. Також ще посередині гравець може побачити опис даного вікна.

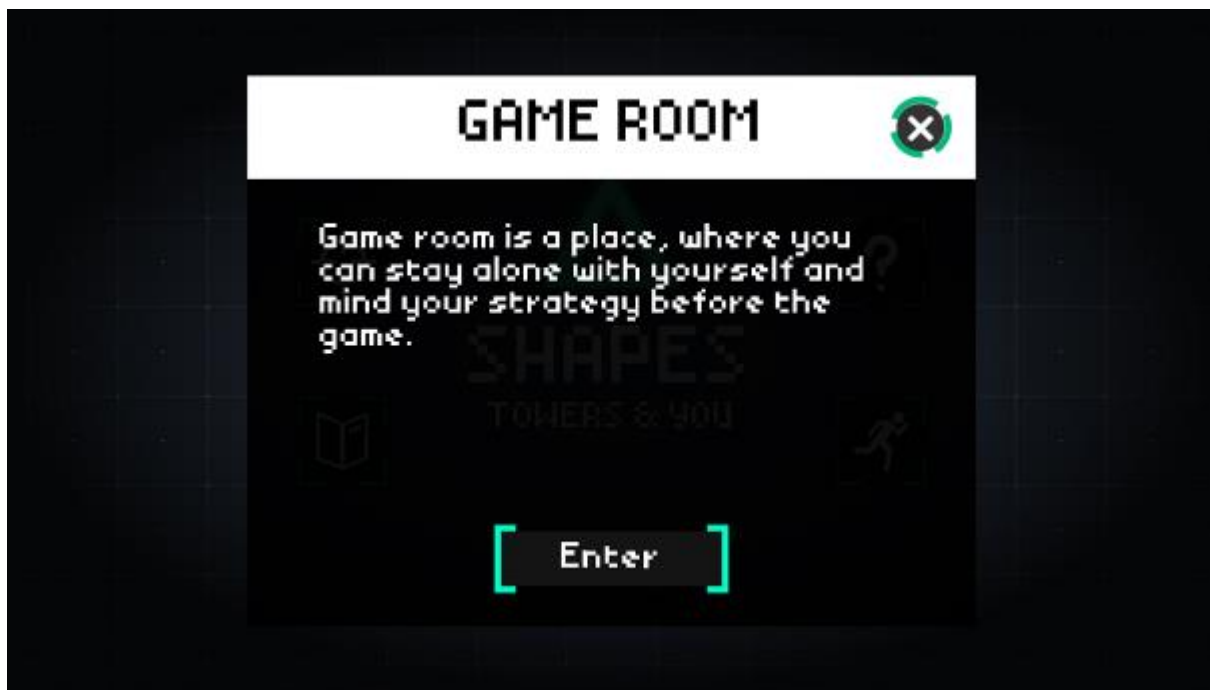


Рисунок 4.4 – Вікно Game Room

Друга кнопка відкриває вікно під назвою Tutorial (рис. 4.5). У даному вікні можна побачити основну інформацію, яка гравцю знадобиться для майбутніх ігор.



Рисунок 4.5 – Вікно Tutorial

Третя кнопка відкриває вікно під назвою Info (рис. 4.6). У даному вікні можна побачити основну інформацію щодо інструментів розробки програми та самого розробника.

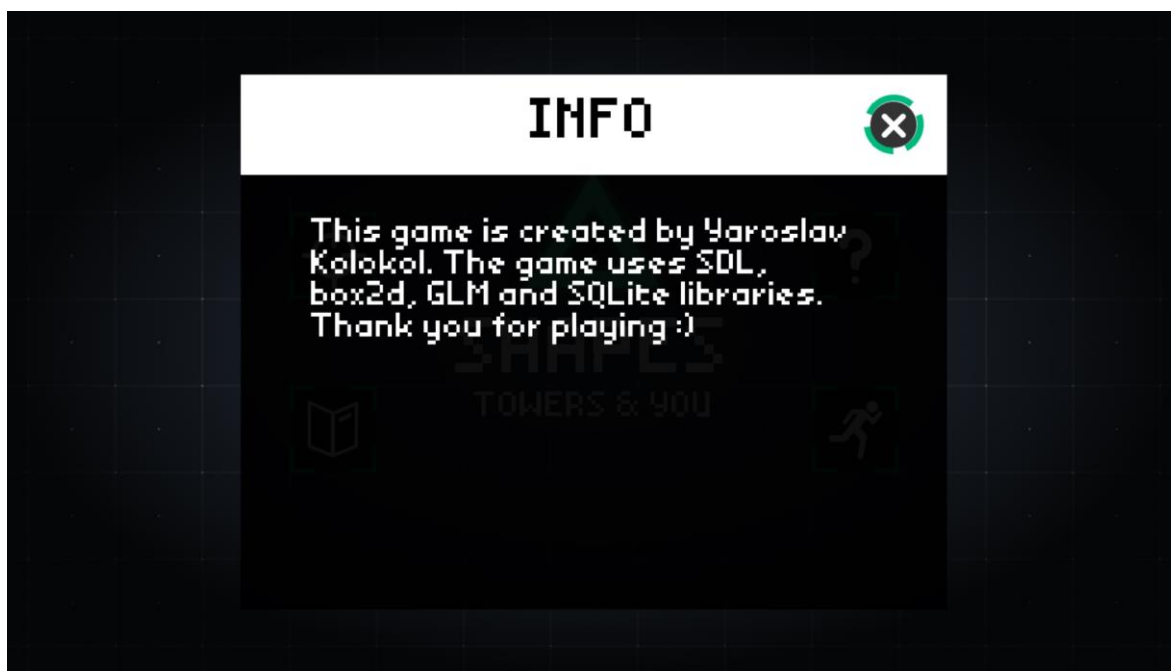


Рисунок 4.6 – Вікно Info

Четверта кнопка відкриває вікно під назвою Exit (рис. 4.7). Дане вікно

служує для безпечного виходу з гри. Користувач також може вийти з гри, натиснувши хрест в правому верхньому куті.

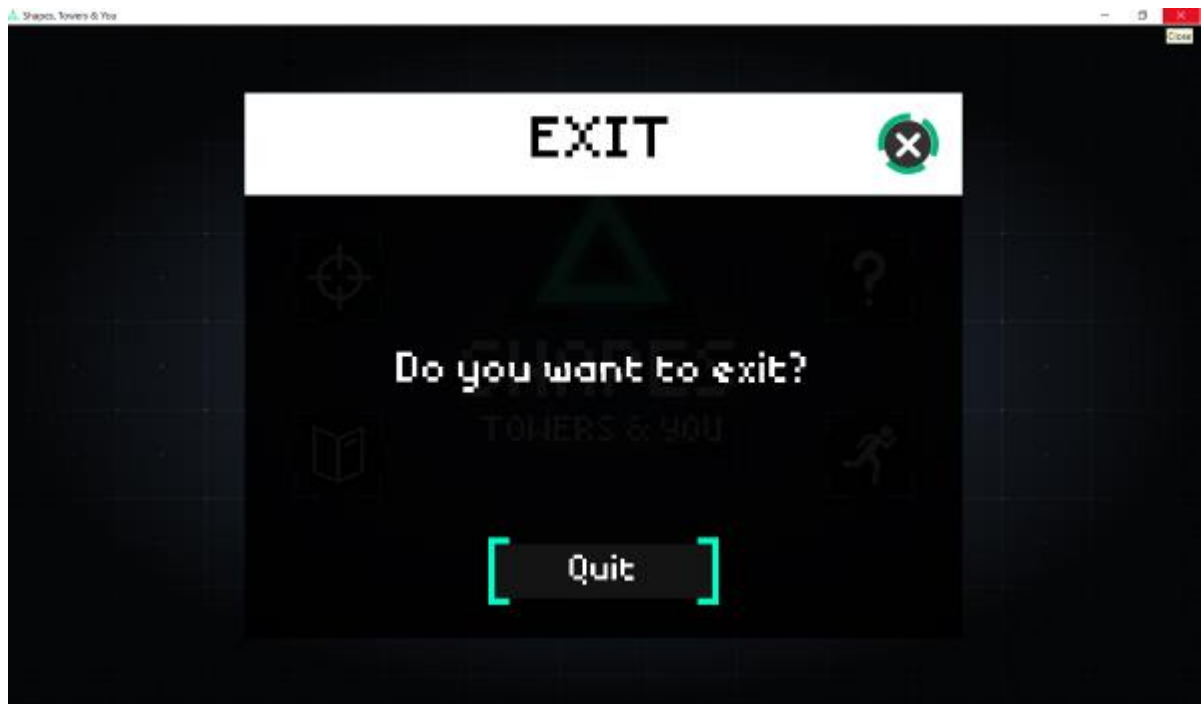


Рисунок 4.7 – Вікно Exit

Коли користувач натискає кнопку Enter у вікні Game Room, то головне меню повністю видаляється, всі ресурси для головного меню звільнюються і створюється основний ігровий режим (рис. 4.8). Основний режим має 2 кнопки інтерфейсу, ігрове поле, 2 союзні одиниці та кількість грошей. Одним із союзних одиниць може напряму керувати гравець. Це і є головний персонаж, який має захищати базу. Персонаж може ходити, якщо натискати клавіші W, A, S, D і цілитись та стріляти натискаючи праву та ліву кнопку миші відповідно.

Вся сцена має 4 стіни, які обмежують все ігрове поле. Таким чином ані союзні одиниці, ані ворожі не зможуть пройти крізь ці стіни. Також на сцені є червона зона, звідки будуть виходити ворожі одиниці у випадковому порядку.

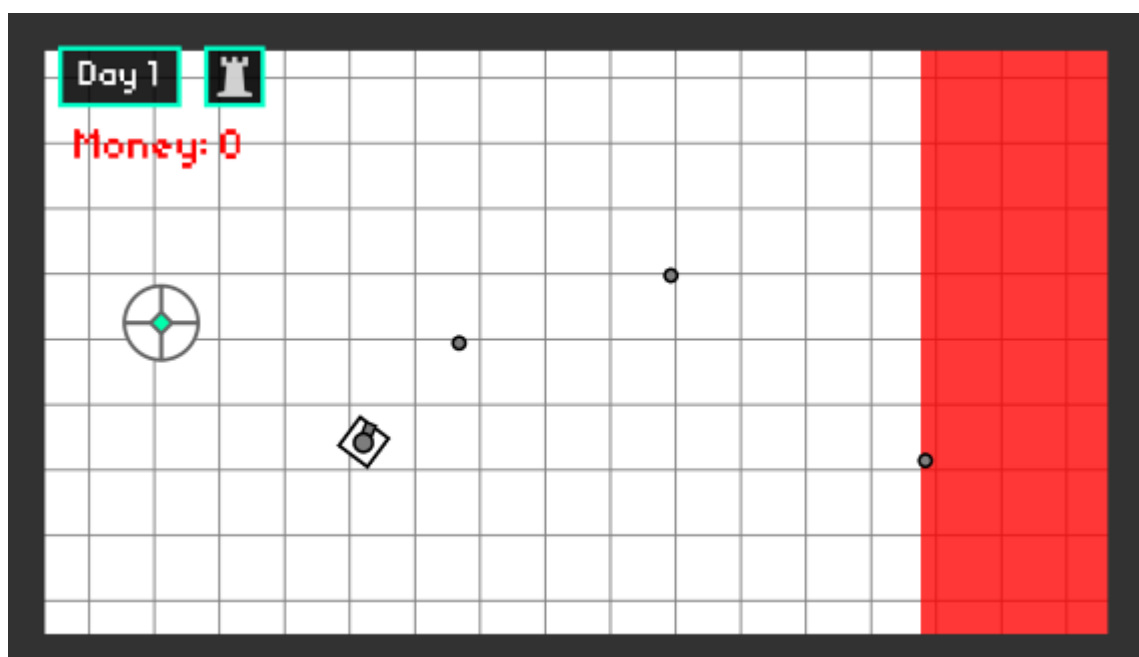


Рисунок 4.8 – Основний ігровий режим

Натискаючи на кнопку Day, гравець запускає хвилю ворогів (рис. 4.9). Тип та кількість ворогів обчислюється за поточним днем. Чим більше число, тим більше буде ворогів, тим сильніші ворожі одиниці будуть з'являтися.

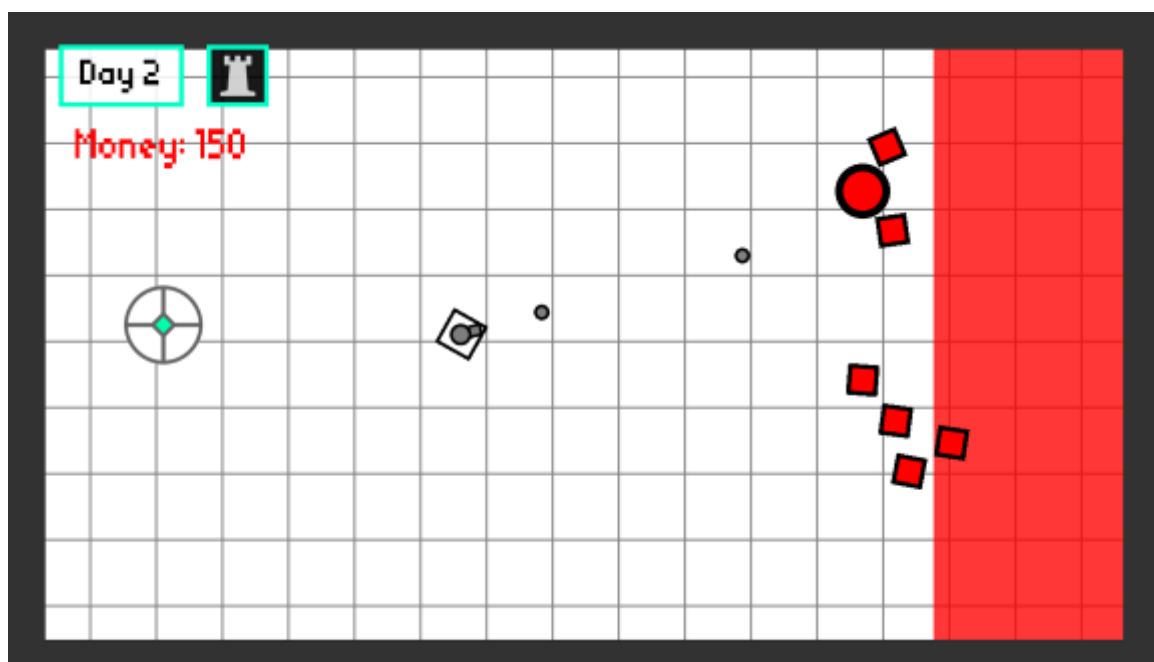


Рисунок 4.9 – Хвиля ворожих одиниць

Заробивши деяку кількість грошей, гравець може натиснути на другу

кнопку та відкрити магазин веж (рис. 4.10). За зароблені гроші можна купити захисну або атакуючу вежу. Атакуючі вежі коштують дорожче, так як окрім своєї ролі, вони ще виконують роль захисних веж.

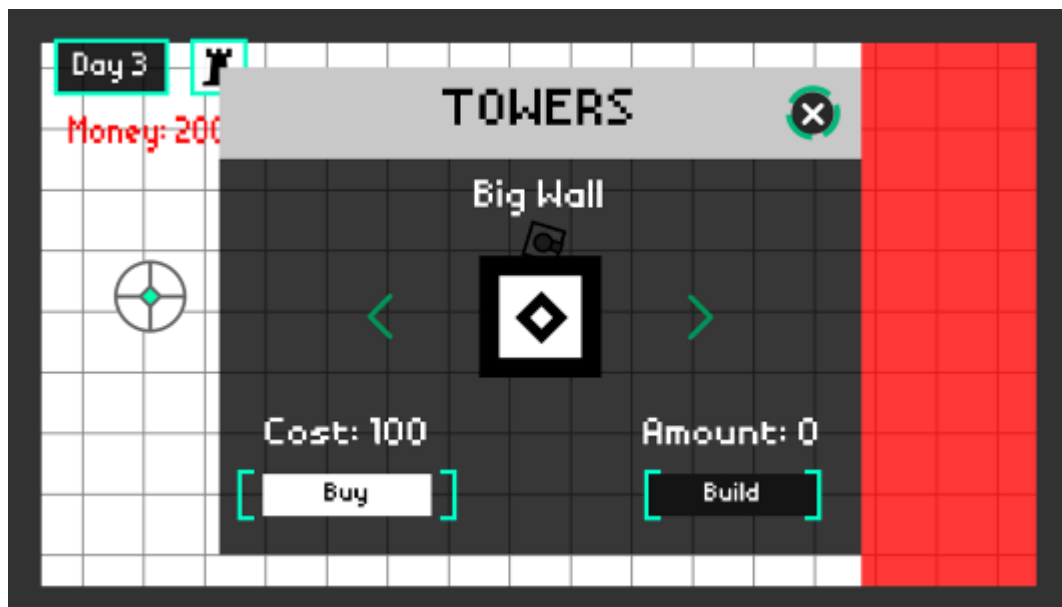


Рисунок 4.10 – Магазин веж

Після покупки вежі, її можна побудувати у довільному просторі (рис. 4.11).

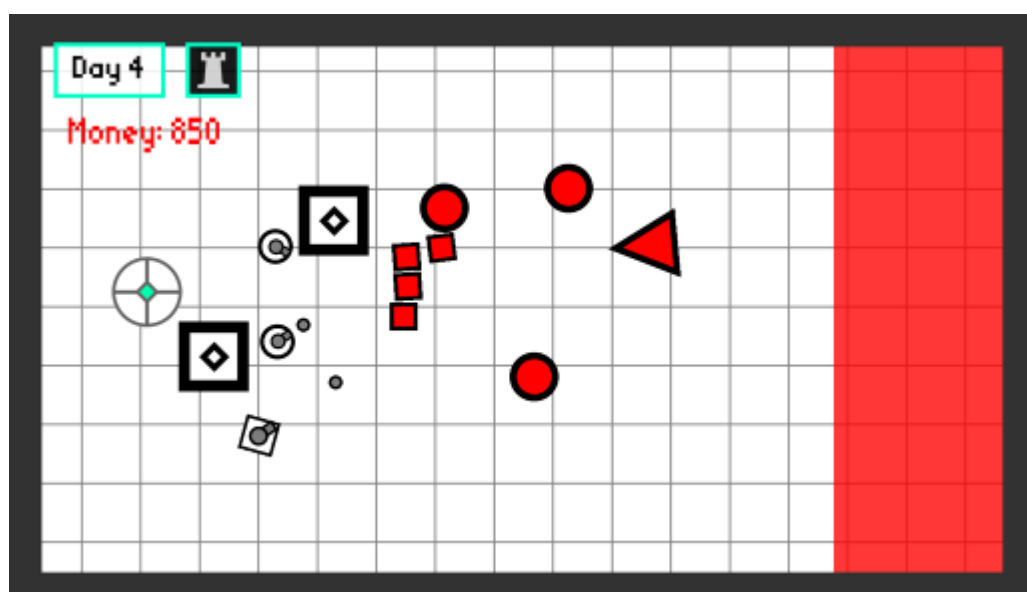


Рисунок 4.11 – Будівництво веж

Якщо гравець не зміг захистити базу у який-небудь день, то гра закінчується і відкривається вікно результатів (рис. 4.12). Кнопка Back слугує для того, щоб повернутися до головного меню.



Рисунок 4.12 – Результати гри

4.5 Висновок

Розроблена інструкція щодо запуску комп'ютерної гри, яка пояснює користувачу, що саме він може зробити, а що не може для запуску програми.

Були описані мінімальні та рекомендовані системні вимоги для обладнання користувача.

Описаний основний функціонал програми.

Було детально оглянуто кожний аспект застосування даної комп'ютерної гри із відображенням основних дій за допомогою скріншотів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи бакалавра було розроблено комп'ютерну гру у жанрі Tower Defense.

Було проведено аналіз предметної області, визначені ключові терміни, пов'язані з нею, а також з об'єктом дослідження й розробки. Розглянута проблема щодо створення комп'ютерної гри в жанрі Tower Defense. Вивчені основні патерни для розробки таких ігор і проведений порівняльний аналіз цих патернів. Також здійснено порівняльний аналіз існуючих аналогів і визначені ключові вимоги до системи та механіки гри.

Проведено аналіз та відбір інструментарію розробки програмного забезпечення, а також була розроблена архітектура майбутньої комп'ютерної гри у жанрі Tower Defense. Проведений порівняльний аналіз мов програмування для створення ПЗ, середовищ розробки та мультимедійних бібліотек.

Була розроблена проста структура проекту та засоби розробки, які дозволяють легко і швидко додавати або видаляти нові елементи системи. Створено три компоненти, які сприяють розробці простих елементів гри. Розроблено основне меню, яке забезпечує зручний інтерфейс для перемикання між ігровими вікнами. Реалізовано основний ігровий режим, де гравець може будувати захисні та атакуючі вежі та керувати основним персонажем, який здатен знищувати ворогів.

Створено посібник з інструкціями щодо запуску комп'ютерної гри, який пояснює користувачеві можливості та обмеження запуску програми. Описано основний функціонал програми та проведено детальний огляд кожного аспекту її використання, включаючи зображення основних дій за допомогою скріншотів.

Було розроблено комп'ютерну гру, де гравець може керувати головним персонажем і будувати різні вежі, які захищатимуть базу гравця. Дана гра може бути опублікована та використана у розважальних сферах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Відеогра [Електрон. ресурс]. - Режим доступу:
<https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%B5%D0%BE%D0%B3%D1%80%D0%B0>
2. Tower defense [Electronic resource]. - Access mode:
https://en.wikipedia.org/wiki/Tower_defense
3. Game Programming Patterns [Electronic resource]. - Access mode:
<https://medium.com/@apoorvmishra05/game-programming-patterns-9e4a877dc942>
4. Game objects [Electronic resource]. - Access mode:
<https://docs.decentraland.org/creator/development-guide/game-objects/>
5. Type Object [Electronic resource]. - Access mode:
<https://gameprogrammingpatterns.com/type-object.html>
6. Component [Electronic resource]. - Access mode:
<https://gameprogrammingpatterns.com/component.html>
7. Entity component system [Electronic resource]. - Access mode:
https://en.wikipedia.org/wiki/Entity_component_system
8. Data Locality [Electronic resource]. - Access mode:
<https://gameprogrammingpatterns.com/data-locality.html>
9. Mindustry [Електрон. ресурс]. - Режим доступу:
<https://uk.wikipedia.org/wiki/Mindustry>
10. Bloons Tower Defense [Електрон. ресурс]. - Режим доступу:
https://uk.wikipedia.org/wiki/Bloons_Tower_Defense
11. Plants vs. Zombies (series) [Electronic resource]. - Access mode:
[https://plantsvszombies.fandom.com/wiki/Plants_vs._Zombies_\(series\)](https://plantsvszombies.fandom.com/wiki/Plants_vs._Zombies_(series))
12. Thronefall [Electronic resource]. - Access mode:
<https://en.wikipedia.org/wiki/Thronefall>
13. Мова програмування [Електрон. ресурс]. - Режим доступу:
https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF

%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F

14. C++ [Електрон. ресурс]. - Режим доступу:
<https://uk.wikipedia.org/wiki/C%2B%2B>

15. C Sharp [Електрон. ресурс]. - Режим доступу:
https://uk.wikipedia.org/wiki/C_Sharp

16. Python [Електрон. ресурс]. - Режим доступу:
<https://uk.wikipedia.org/wiki/Python>

17. Інтегроване середовище розробки [Електрон. ресурс]. - Режим доступу:

https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%BE%D0%B2%D0%B8%D1%89%D0%B5_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8

18. Microsoft Visual Studio [Електрон. ресурс]. - Режим доступу:
https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

19. The Best C++ IDEs of 2023 [Electronic resource]. - Access mode:
<https://www.incredibuild.com/blog/best-c-ides>

20. Simple DirectMedia Layer [Electronic resource]. - Access mode:
https://en.wikipedia.org/wiki/Simple_DirectMedia_Layer

21. Simple and Fast Multimedia Library [Electronic resource]. - Access mode: https://en.wikipedia.org/wiki/Simple_and_Fast_Multimedia_Library

22. Allegro (software library) [Electronic resource]. - Access mode:
[https://en.wikipedia.org/wiki/Allegro_\(software_library\)](https://en.wikipedia.org/wiki/Allegro_(software_library))

23. Game Loop [Electronic resource]. - Access mode:
<https://gameprogrammingpatterns.com/game-loop.html>

24. Learn Makefiles with the tastiest examples [Electronic resource]. - Access mode: <https://makefiletutorial.com/>

25. Inkscape [Електрон. ресурс]. - Режим доступу:
<https://uk.wikipedia.org/wiki/Inkscape>

ДОДАТОК А
Текст програми

A.1 Файл main.cpp

```
#include "Game.h"

int main(int argc, char *argv[])
{
    Game game = Game();
    game.run();

    return 0;
}
```

A.2 Файл Game.h

```
#pragma once

#include "GameInit.h"
#include "GameData.h"

class Root;

class Game{
public:
    GameInit gameInit = GameInit();
    GameData gameData = GameData();

    Root *root;

    Game();
    ~Game();

    void run();
    void game_loop();

    void events();
    void fixed_update();
    void update();
    void render();
};
```

A.3 Файл Game.cpp

```
#include "Game.h"
#include "Root.h"

Game::Game() {

    // Set game icon
    SDL_Surface* icon = IMG_Load("../images/mainMenu/MainLogo1.png");
    SDL_SetWindowIcon(gameData.window, icon);
    SDL_FreeSurface(icon);

    root = new Root("Root", NULL_EVENT, &gameData, NULL);
}
```

```

}

Game::~Game() {

    if(root != NULL){
        delete root;
        root = NULL;
    }

    printf("The end of the game\n");
}

void Game::run() {

    game_loop();

}

void Game::game_loop() {

    Uint32 t1 = 0, t2 = 1;

    while(gameData.active == true){

        // Begin frame
        t1 = SDL_GetTicks();

        // Frame time which elapsed in seconds
        gameData.deltaTime = (float)t2 / 1000.0;
        gameData.fixedTime += gameData.deltaTime;

        events();
        while(gameData.fixedTime >= gameData.fixedDeltaTime){
            fixed_update();
            gameData.fixedTime -= gameData.fixedDeltaTime;
        }
        update();
        render();

        SDL_Delay(10);

        // End frame
        t2 = SDL_GetTicks() - t1;
    }

}

void Game::events() {

```

```

// Process every event, one by one
while(SDL_PollEvent(&gameData.event) == 1){

    // Check current event and decide what we can do with it
    switch(gameData.event.type){

        // Exit
        case SDL_QUIT:
            gameData.active = false;
            break;

        // Window events
        case SDL_WIDOWEVENT:
            switch(gameData.event.window.event){
                case SDL_WIDOWEVENT_RESIZED:
                    SDL_GetWindowSize(gameData.window,
&gameData.screenWidth, &gameData.screenHeight);
                    break;
            }
            break;

        // Test
        case SDL_KEYDOWN:
            switch (gameData.event.key.keysym.sym)
            {
                case SDLK_LEFT:

                    break;
                case SDLK_RIGHT:

                    break;
            }
            break;
    }

    // Other stuff also checks current event and decides what it
can do with it
    if(root != NULL){
        root->process_events();
        //printf("Process root\n");
    }

}

void Game::fixed_update(){

    gameData.world.Step(gameData.fixedDeltaTime,
gameData.velocityIterations, gameData.positionIterations);

    // Process all physics
    if(root != NULL){
        root->process_fixed_update();
    }

}

```

```

void Game::update() {
    // Process all updates
    if(root != NULL) {
        root->process_update();
    }
}

void Game::render() {
    // Choose green color
    SDL_SetRenderDrawBlendMode(gameData.renderer,
SDL_BLENDMODE_BLEND);
    SDL_SetRenderDrawColor(gameData.renderer, 0, 255, 200, 255);
    // Clear the whole screen
    SDL_RenderClear(gameData.renderer);

    // Render other stuff
    if(root != NULL) {
        root->process_render();
    }

    // Swap buffer
    SDL_RenderPresent(gameData.renderer);
}

```

A.4 Файл GameData.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <box2d/box2d.h>
#include <vector>
#include <iostream>

#include "UserData.h"
#include "TextureManager.h"
#include "ContactListener.h"
#include "Towers.h"

#include "Enumerations.h"

class GameData{
public:
    UserData userData = UserData();
    TextureManager textureManager = TextureManager(this);
    ContactListener contactListener = ContactListener();
    Towers towers = Towers(this);

    SDL_Window *window;
    SDL_Renderer *renderer;
    SDL_Event event;
}

```

```

        bool active;
        bool pause;
        bool dayActive;
        int enemyCounter;
        EventState currentEvent;
        int screenWidth;
        int screenHeight;

        float targetFPS;
        float deltaTime;
        float fixedDeltaTime;
        int32 velocityIterations;
        int32 positionIterations;

        b2World world = b2World(b2Vec2(0.0f, 0.0f));
        float fixedTime = 0.0;
        const float METER_TO_PIXEL = 100.0f;
        const float PIXEL_TO_METER = 0.01f;

        // Main game font
        TTF_Font *font;
        int fontSize;

        GameData();
        ~GameData();
};

```

A.5 Файл GameData.cpp

```

#include "GameData.h"

GameData::GameData() {

    active = true;
    pause = false;
    dayActive = false;
    enemyCounter = 0;

    currentEvent = MAIN_MENU;

    screenWidth = 900;
    screenHeight = 500;

    window = SDL_CreateWindow("Shapes, Towers & You", 400, 200,
screenWidth, screenHeight, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
    renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    deltaTime = 1.0 / 60.0;
    fixedDeltaTime = 1.0 / 60.0;

```

```

        velocityIterations = 6;
        positionIterations = 2;
        world.SetContactListener(&contactListener);

        fontSize = 100;
        font = TTF_OpenFont("../fonts/PixelFont.ttf", fontSize);

        // Antialiasing mode - level 1
        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1");

        printf("Data is initialised\n");
    }

```

```

GameData::~GameData() {

    textureManager.deallocate_tower_defense();
    textureManager.deallocate_main_menu();
    textureManager.deallocate_common();

    TTF_CloseFont(font);
    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);

    printf("Data is cleared\n");
}

```

A.6 Файл GameInit.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>

class GameInit{

public:

    GameInit();
    ~GameInit();

};

```

A.7 Файл GameInit.cpp

```

#include "GameInit.h"

GameInit::GameInit() {

    // Init all SDL stuff
    SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER);
    IMG_Init(IMG_INIT_PNG);
    TTF_Init();
}

```

```

        printf("SDL is initialised\n");
    }
    GameInit::~~GameInit() {

        TTF_Quit();
        IMG_Quit();
        SDL_Quit();

        printf("SDL is turned off\n");
        printf("The game is successfully ended!\n");
    }

```

A.8 Файл GameObject.h

```

#pragma once

#include "Transform.h"
#include <vector>

#include "Enumerations.h"

class Component;
class GameData;

class GameObject{
public:

    std::string name;
    EventState objectEvent;
    GameData *gameData;

    Transform transform = Transform();

    int index = 0;
    bool isAlive = true;
    GameObject *parent;

    std::vector<GameObject *> children;
    std::vector<Component *> components;

    GameObject(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    virtual ~GameObject();

    virtual void events();
    virtual void fixed_update();
    virtual void update();
    virtual void render();

    void process_events();
    void process_fixed_update();
    void process_update();
    void process_render();

```

```

        void add_child(GameObject *child);
        void delete_child(int indexValue);
        void delete_children();

        void add_component(Component *component);
        void delete_component(int indexValue);
        void delete_components();

        void destroy();
};

```

A.9 Файл GameObject.cpp

```

#include "GameObject.h"
#include "GameData.h"
#include "Component.h"

GameObject::GameObject(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue){

    name = nameValue;
    objectEvent = objectEventValue;
    gameData = gameDataValue;
    parent = parentValue;
}

GameObject::~GameObject() {

    delete_components();
    delete_children();
}

void GameObject::events() {

}

void GameObject::fixed_update() {

}

void GameObject::update() {

}

void GameObject::render() {

```

```

}

void GameObject::process_events() {
    events();

    for(int i = 0; i < components.size(); ++i){
        components[i]->events();
    }

    for(int i = 0; i < children.size(); ++i){
        children[i]->process_events();
    }

    if(isAlive == false){
        parent->delete_child(index);
    }
}

void GameObject::process_fixed_update() {
    fixed_update();

    for(int i = 0; i < components.size(); ++i){
        components[i]->fixed_update();
    }

    for(int i = 0; i < children.size(); ++i){
        children[i]->process_fixed_update();
    }

    if(isAlive == false){
        parent->delete_child(index);
    }
}

void GameObject::process_update() {
    update();

    for(int i = 0; i < components.size(); ++i){
        components[i]->update();
    }

    for(int i = 0; i < children.size(); ++i){
        children[i]->process_update();
    }

    if(isAlive == false){
        parent->delete_child(index);
    }
}

void GameObject::process_render() {
    render();

    for(int i = 0; i < components.size(); ++i){

```

```

        components[i]->render();
    }

    for(int i = 0; i < children.size(); ++i){
        children[i]->process_render();
    }

    if(isAlive == false){
        parent->delete_child(index);
    }
}

void GameObject::add_child(GameObject *child){
    children.push_back(child);
    children[children.size() - 1]->index = children.size() - 1;
}

void GameObject::delete_child(int indexValue){
    delete children[indexValue];
    children.erase(children.begin() + indexValue);
    for(int i = indexValue; i < children.size(); ++i){
        children[i]->index = i;
    }
}

void GameObject::delete_children(){
    for(int i = 0; i < children.size(); ++i){
        delete children[i];
    }
    children.clear();
}

void GameObject::add_component(Component *component){
    components.push_back(component);
    components[components.size() - 1]->parent = this;
}

void GameObject::delete_component(int indexValue){
    delete components[indexValue];
    components.erase(components.begin() + indexValue);
}

void GameObject::delete_components(){
    for(int i = 0; i < components.size(); ++i){
        delete components[i];
    }
    components.clear();
}

void GameObject::destroy(){
    isAlive = false;
}

```

```
}
```

A.10 Файл Component.h

```
#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <box2d/box2d.h>
#include <string>

class GameObject;

class Component{

public:

    std::string name;
    GameObject *parent;

    Component(std::string nameValue);
    virtual ~Component();

    virtual void events();
    virtual void fixed_update();
    virtual void update();
    virtual void render();

};
```

A.11 Файл Component.cpp

```
#include "Component.h"
#include "GameObject.h"

Component::Component(std::string nameValue){

    name = nameValue;
    parent = NULL;

}

Component::~~Component(){

}

void Component::events(){

}

void Component::fixed_update(){
```

```

}
void Component::update() {

}
void Component::render() {

}

```

A.12 Файл ContactListener.h

```

#pragma once

#include <box2d/box2d.h>
#include <stdio.h>
#include <iostream>

class ContactListener : public b2ContactListener{

public:

    ContactListener();
    ~ContactListener();

    void BeginContact(b2Contact* contact) override;
    void EndContact(b2Contact* contact) override;

};

```

A.13 Файл ContactListener.cpp

```

#include "ContactListener.h"
#include "UnitBullet.h"
#include "Enemy1.h"
#include "Enemy2.h"
#include "Enemy3.h"
#include "GameData.h"
#include "Base.h"
#include "Player.h"
#include "Gun.h"

ContactListener::ContactListener() {

}

ContactListener::~~ContactListener() {

    printf("ContactListener is destroyed\n");
}

void ContactListener::BeginContact(b2Contact* contact) {

    b2BodyUserData    data1    =    contact->GetFixtureA()->GetBody()-
>GetUserData();
    b2BodyUserData    data2    =    contact->GetFixtureB()->GetBody()-
>GetUserData();

```

```

        GameObject *temp1 = reinterpret_cast<GameObject *>(data1.pointer);
        GameObject *temp2 = reinterpret_cast<GameObject *>(data2.pointer);

        if(temp1->name.find("Unit") != std::string::npos && temp2->name.find("Enemy") != std::string::npos){
            Unit *unit = reinterpret_cast<Unit *>(data1.pointer);
            Enemy *enemy = reinterpret_cast<Enemy *>(data2.pointer);
            unit->health -= enemy->damage;
            enemy->dead = true;
            if((*unit).health <= 0){
                unit->dead = true;
            }
        }
        else if(temp2->name.find("Unit") != std::string::npos && temp1->name.find("Enemy") != std::string::npos){
            Unit *unit = reinterpret_cast<Unit *>(data2.pointer);
            Enemy *enemy = reinterpret_cast<Enemy *>(data1.pointer);
            unit->health -= enemy->damage;
            enemy->dead = true;
            if((*unit).health <= 0){
                unit->dead = true;
            }
        }
        else if(temp1->name.find("Bullet") != std::string::npos && temp2->name.find("Wall") != std::string::npos){
            UnitBullet *bullet = reinterpret_cast<UnitBullet *>(data1.pointer);
            bullet->dead = true;
        }
        else if(temp2->name.find("Bullet") != std::string::npos && temp1->name.find("Wall") != std::string::npos){
            UnitBullet *bullet = reinterpret_cast<UnitBullet *>(data2.pointer);
            bullet->dead = true;
        }
        else if(temp1->name.find("Bullet") != std::string::npos && temp2->name.find("Enemy") != std::string::npos){
            UnitBullet *bullet = reinterpret_cast<UnitBullet *>(data1.pointer);
            bullet->dead = true;
            Enemy *enemy = reinterpret_cast<Enemy *>(data2.pointer);
            enemy->health -= bullet->damage;
            if((*enemy).health <= 0){
                enemy->gameData->userData.money += enemy->money;
                enemy->dead = true;
            }
        }
        else if(temp2->name.find("Bullet") != std::string::npos && temp1->name.find("Enemy") != std::string::npos){
            UnitBullet *bullet = reinterpret_cast<UnitBullet *>(data2.pointer);
            bullet->dead = true;
            Enemy *enemy = reinterpret_cast<Enemy *>(data1.pointer);
            enemy->health -= bullet->damage;
            if((*enemy).health <= 0){
                enemy->gameData->userData.money += enemy->money;
                enemy->dead = true;
            }
        }
        else if(temp1->name.find("Gun") != std::string::npos && temp2->name.find("Enemy") != std::string::npos){
            Gun *gun = reinterpret_cast<Gun *>(data1.pointer);
            Enemy *enemy = reinterpret_cast<Enemy *>(data2.pointer);

```

```

        gun->fire = true;
        gun->target = &enemy->transform;
    }
    else if(temp2->name.find("Gun") != std::string::npos && temp1->name.find("Enemy") != std::string::npos){
        Gun *gun = reinterpret_cast<Gun *>(data2.pointer);
        Enemy *enemy = reinterpret_cast<Enemy *>(data1.pointer);
        gun->fire = true;
        gun->target = &enemy->transform;
    }
}

void ContactListener::EndContact(b2Contact* contact){

    b2BodyUserData data1 = contact->GetFixtureA()->GetBody()->GetUserData();
    b2BodyUserData data2 = contact->GetFixtureB()->GetBody()->GetUserData();

    GameObject *temp1 = reinterpret_cast<GameObject *>(data1.pointer);
    GameObject *temp2 = reinterpret_cast<GameObject *>(data2.pointer);

    if(temp1->name.find("Gun") != std::string::npos && temp2->name.find("Enemy") != std::string::npos){
        Gun *gun = reinterpret_cast<Gun *>(data1.pointer);
        Enemy *enemy = reinterpret_cast<Enemy *>(data2.pointer);
        gun->fire = false;
        gun->target = NULL;
    }
    else if(temp2->name.find("Gun") != std::string::npos && temp1->name.find("Enemy") != std::string::npos){
        Gun *gun = reinterpret_cast<Gun *>(data2.pointer);
        Enemy *enemy = reinterpret_cast<Enemy *>(data1.pointer);
        gun->fire = false;
        gun->target = NULL;
    }
}

```

A.14 Файл UserData.h

```

#pragma once

#include <sqlite3.h>
#include <stdio.h>

class UserData{

public:

    sqlite3 *db;

    int dayCounter = 0;
    int money = 0;

    int maxDays = 0;
    int maxMoney = 0;

    UserData();

```

```

        ~UserData();

};

```

A.15 Файл UserData.cpp

```

#include "UserData.h"

UserData::UserData() {

    sqlite3_open("database.db", &db);
    int rc;
    const char *sql = "CREATE TABLE IF NOT EXISTS MyTable (id INTEGER
PRIMARY KEY, day INTEGER, money INTEGER);";

    char *messageError;
    sqlite3_exec(db, sql, 0, 0, &messageError);

    const char *select_sql = "SELECT day, money FROM MyTable;";
    sqlite3_stmt *stmt;
    sqlite3_prepare_v2(db, select_sql, -1, &stmt, NULL);

    while ((rc = sqlite3_step(stmt)) == SQLITE_ROW) {
        // Retrieve values from the result row
        maxDays = sqlite3_column_int(stmt, 0);
        maxMoney = sqlite3_column_int(stmt, 1);

        // Output the retrieved values
        //std::cout << "Day: " << day << ", Money: " << money <<
std::endl;
    }

    sqlite3_finalize(stmt);
}

UserData::~UserData() {

    int rc;
    sqlite3_stmt *stmt;
    const char *insert_sql = "INSERT INTO MyTable (day, money) VALUES
(?, ?);";
    sqlite3_prepare_v2(db, insert_sql, -1, &stmt, NULL);

    // Bind the values to the placeholders
    sqlite3_bind_int(stmt, 1, maxDays);
    sqlite3_bind_int(stmt, 2, maxMoney);

    // Execute the statement
    rc = sqlite3_step(stmt);

    sqlite3_finalize(stmt);
    sqlite3_close(db);

    printf("UserData is destroyed\n");
}

```

A.16 Файл Root.h

```

#pragma once

```

```

#include "GameObject.h"

class Root : public GameObject{
public:
    Root(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~Root() override;

    void create_main_menu();
    void delete_main_menu();

    void create_tower_defense();
    void delete_tower_defense();
};

```

A.17 Файл Root.cpp

```

#include "Root.h"
#include "MainMenu.h"
#include "GameData.h"
#include "TowerDefense.h"

Root::Root(std::string nameValue, EventState objectEventValue, GameData
*gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    gameData->textureManager.allocate_common();

    create_main_menu();
    //create_tower_defense();
}
Root::~~Root(){

    gameData->currentEvent = NULL_EVENT;
}

void Root::create_main_menu(){

    gameData->textureManager.allocate_main_menu();
    gameData->currentEvent = MAIN_MENU;
    add_child(new MainMenu("Main Menu", MAIN_MENU, gameData, this));
}

void Root::delete_main_menu(){

    //delete_child(0);
    children[0]->destroy();
}

```

```

void Root::create_tower_defense() {
    gameData->textureManager.allocate_tower_defense();
    gameData->towers.set_images();
    gameData->currentEvent = TOWER_DEFENSE;
    add_child(new TowerDefense("Tower Defense", TOWER_DEFENSE,
gameData, this));
}

void Root::delete_tower_defense() {
    //delete_child(0);
    children[0]->destroy();
}

```

A.18 Файл Transform.h

```

#pragma once

#include <iostream>
#include <glm/glm.hpp>
#include <glm/vec2.hpp>
#include <glm/gtc/quaternion.hpp>
#include <glm/gtc/constants.hpp>

class Transform{
public:
    glm::vec2 position;
    float rotation;
    glm::vec2 scale;

    glm::vec2 velocity;

    int widthUI;
    int heightUI;
    float widthByHeightRatio;

    Transform();
    ~Transform();

    glm::vec2 normalize();
};

```

A.19 Файл Transform.cpp

```

#include "Transform.h"

Transform::Transform() {

```

```

        position.x = 0.0;
        position.y = 0.0;

        rotation = 0.0;

        scale.x = 1.0;
        scale.y = 1.0;

        velocity.x = 0.0;
        velocity.y = 0.0;

        widthUI = 0;
        heightUI = 0;
        widthByHeightRatio = 0.0;
    }

    Transform::~Transform() {

    }

    glm::vec2 Transform::normalize() {

        glm::vec2 velocityTemp = velocity;

        if(glm::length(velocity) > 1){
            velocityTemp = glm::normalize(velocity);
        }

        return velocityTemp;
    }

```

A.20 Файл TextureManager.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

class GameData;

class TextureManager{
public:

    // Parent
    GameData *gameData;

    // Main Menu
    SDL_Texture *cellTexture = NULL;
    SDL_Texture *logo = NULL;
    SDL_Texture *radialShade = NULL;

    SDL_Texture *waveButton1 = NULL;
    SDL_Texture *waveButton2 = NULL;

```

```
SDL_Texture *shopButton1 = NULL;
SDL_Texture *shopButton2 = NULL;
```

```
SDL_Texture *tutorialButton1 = NULL;
SDL_Texture *tutorialButton2 = NULL;
```

```
SDL_Texture *settingsButton1 = NULL;
SDL_Texture *settingsButton2 = NULL;
```

```
SDL_Texture *infoButton1 = NULL;
SDL_Texture *infoButton2 = NULL;
```

```
SDL_Texture *exitButton1 = NULL;
SDL_Texture *exitButton2 = NULL;
```

```
// Tower Defense
SDL_Texture *grid = NULL;
SDL_Texture *bodySprite1 = NULL;
SDL_Texture *headSprite1 = NULL;
SDL_Texture *base = NULL;
SDL_Texture *enemy1 = NULL;
SDL_Texture *enemy2 = NULL;
SDL_Texture *enemy3 = NULL;
SDL_Texture *bullet1 = NULL;
SDL_Texture *towerButton1 = NULL;
SDL_Texture *towerButton2 = NULL;
SDL_Texture *smallWall = NULL;
SDL_Texture *bigWall = NULL;
SDL_Texture *changeTower = NULL;
SDL_Texture *gunTowerHead1 = NULL;
SDL_Texture *gunTowerHead2 = NULL;
SDL_Texture *gunTowerBody1 = NULL;
SDL_Texture *gunTowerBody2 = NULL;
SDL_Texture *gunTower1 = NULL;
SDL_Texture *gunTower2 = NULL;
```

```
// Common
SDL_Texture *backButton1 = NULL;
SDL_Texture *backButton2 = NULL;
SDL_Texture *backButtonBounds = NULL;
```

```
TextureManager(GameData *gameDataValue);
~TextureManager();
```

```
void allocate_main_menu();
void deallocate_main_menu();
```

```
void allocate_tower_defense();
void deallocate_tower_defense();
```

```
void allocate_common();
void deallocate_common();
```

```
};
```

A.21 Файл TextureManager.cpp

```

#include "TextureManager.h"
#include "GameData.h"

TextureManager::TextureManager(GameData *gameDataValue) {

    gameData = gameDataValue;

}

TextureManager::~TextureManager() {

    deallocate_tower_defense();
    deallocate_main_menu();
    deallocate_common();
    printf("TextureManager is destroyed\n");
}

void TextureManager::allocate_tower_defense() {

    if(grid == NULL){
        grid = IMG_LoadTexture((*gameData).renderer,
"../images/towerDefense/CellGrid.png");
    }
    if(enemy1 == NULL){
        enemy1 = IMG_LoadTexture((*gameData).renderer,
"../images/enemies/Enemy1.png");
    }
    if(enemy2 == NULL){
        enemy2 = IMG_LoadTexture((*gameData).renderer,
"../images/enemies/Enemy2.png");
    }
    if(enemy3 == NULL){
        enemy3 = IMG_LoadTexture((*gameData).renderer,
"../images/enemies/Enemy3.png");
    }
    if(bullet1 == NULL){
        bullet1 = IMG_LoadTexture((*gameData).renderer,
"../images/units/bullet1.png");
    }
    if(bodySprite1 == NULL){
        bodySprite1 = IMG_LoadTexture((*gameData).renderer,
"../images/player/body/Body1.png");
    }
    if(headSprite1 == NULL){
        headSprite1 = IMG_LoadTexture((*gameData).renderer,
"../images/player/head/Head1.png");
    }
    if(base == NULL){
        base = IMG_LoadTexture((*gameData).renderer,
"../images/units/Base.png");
    }
    if(towerButton1 == NULL){
        towerButton1 = IMG_LoadTexture((*gameData).renderer,
"../images/towerDefense/TowerShop1.png");
    }
    if(towerButton2 == NULL){

```

```

        towerButton2 = IMG_LoadTexture((*gameData).renderer,
"../images/towerDefense/TowerShop2.png");
    }
    if(smallWall == NULL){
        smallWall = IMG_LoadTexture((*gameData).renderer,
"../images/units/SmallWall.png");
    }
    if(bigWall == NULL){
        bigWall = IMG_LoadTexture((*gameData).renderer,
"../images/units/BigWall.png");
    }
    if(changeTower == NULL){
        changeTower = IMG_LoadTexture((*gameData).renderer,
"../images/towerDefense/ChangeTower.png");
    }
    if(gunTowerHead1 == NULL){
        gunTowerHead1 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTowerHead1.png");
    }
    if(gunTowerHead2 == NULL){
        gunTowerHead2 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTowerHead2.png");
    }
    if(gunTowerBody1 == NULL){
        gunTowerBody1 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTowerBody1.png");
    }
    if(gunTowerBody2 == NULL){
        gunTowerBody2 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTowerBody2.png");
    }
    if(gunTower1 == NULL){
        gunTower1 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTower1.png");
    }
    if(gunTower2 == NULL){
        gunTower2 = IMG_LoadTexture((*gameData).renderer,
"../images/units/GunTower2.png");
    }
}

void TextureManager::deallocate_tower_defense(){

    if(grid != NULL){
        SDL_DestroyTexture(grid);
        grid = NULL;
    }
    if(enemy1 != NULL){
        SDL_DestroyTexture(enemy1);
        enemy1 = NULL;
    }
    if(enemy2 != NULL){
        SDL_DestroyTexture(enemy2);
        enemy2 = NULL;
    }
    if(enemy3 != NULL){
        SDL_DestroyTexture(enemy3);
        enemy3 = NULL;
    }
    if(bullet1 != NULL){
        SDL_DestroyTexture(bullet1);
        bullet1 = NULL;
    }
}

```

```

        if (bodySprite1 != NULL) {
            SDL_DestroyTexture(bodySprite1);
            bodySprite1 = NULL;
        }
        if (headSprite1 != NULL) {
            SDL_DestroyTexture(headSprite1);
            headSprite1 = NULL;
        }
        if (base != NULL) {
            SDL_DestroyTexture(base);
            base = NULL;
        }
        if (towerButton1 != NULL) {
            SDL_DestroyTexture(towerButton1);
            towerButton1 = NULL;
        }
        if (towerButton2 != NULL) {
            SDL_DestroyTexture(towerButton2);
            towerButton2 = NULL;
        }
        if (smallWall != NULL) {
            SDL_DestroyTexture(smallWall);
            smallWall = NULL;
        }
        if (bigWall != NULL) {
            SDL_DestroyTexture(bigWall);
            bigWall = NULL;
        }
        if (changeTower != NULL) {
            SDL_DestroyTexture(changeTower);
            changeTower = NULL;
        }
        if (gunTowerHead1 != NULL) {
            SDL_DestroyTexture(gunTowerHead1);
            gunTowerHead1 = NULL;
        }
        if (gunTowerHead2 != NULL) {
            SDL_DestroyTexture(gunTowerHead2);
            gunTowerHead2 = NULL;
        }
        if (gunTowerBody1 != NULL) {
            SDL_DestroyTexture(gunTowerBody1);
            gunTowerBody1 = NULL;
        }
        if (gunTowerBody2 != NULL) {
            SDL_DestroyTexture(gunTowerBody2);
            gunTowerBody2 = NULL;
        }
        if (gunTower1 != NULL) {
            SDL_DestroyTexture(gunTower1);
            gunTower1 = NULL;
        }
        if (gunTower2 != NULL) {
            SDL_DestroyTexture(gunTower2);
            gunTower2 = NULL;
        }
    }

}

void TextureManager::allocate_main_menu() {
    if (cellTexture == NULL) {

```

```

        cellTexture      =      IMG_LoadTexture ((*gameData).renderer,
"./images/CellGrid1.png");
    }
    if(radialShade == NULL){
        radialShade      =      IMG_LoadTexture ((*gameData).renderer,
"./images/RadialShade1.png");
    }
    if(logo == NULL){
        logo              =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/MainLogo1.png");
    }
    if(waveButton1 == NULL){
        waveButton1       =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/WaveButton1.png");
    }
    if(waveButton2 == NULL){
        waveButton2       =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/WaveButton2.png");
    }
    if(shopButton1 == NULL){
        shopButton1       =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/ShopButton1.png");
    }
    if(shopButton2 == NULL){
        shopButton2       =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/ShopButton2.png");
    }
    if(tutorialButton1 == NULL){
        tutorialButton1    =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/TutorialButton1.png");
    }
    if(tutorialButton2 == NULL){
        tutorialButton2    =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/TutorialButton2.png");
    }
    if(settingsButton1 == NULL){
        settingsButton1    =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/SettingsButton1.png");
    }
    if(settingsButton2 == NULL){
        settingsButton2    =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/SettingsButton2.png");
    }
    if(infoButton1 == NULL){
        infoButton1        =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/QuestionButton1.png");
    }
    if(infoButton2 == NULL){
        infoButton2        =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/QuestionButton2.png");
    }
    if(exitButton1 == NULL){
        exitButton1        =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/ExitButton1.png");
    }
    if(exitButton2 == NULL){
        exitButton2        =      IMG_LoadTexture ((*gameData).renderer,
"./images/mainMenu/mainButton/ExitButton2.png");
    }
}

void TextureManager::deallocate_main_menu() {

```

```

        if(cellTexture != NULL){
            SDL_DestroyTexture(cellTexture);
            cellTexture = NULL;
        }
        if(radialShade != NULL){
            SDL_DestroyTexture(radialShade);
            radialShade = NULL;
        }
        if(logo != NULL){
            SDL_DestroyTexture(logo);
            logo = NULL;
        }
        if(waveButton1 != NULL){
            SDL_DestroyTexture(waveButton1);
            waveButton1 = NULL;
        }
        if(waveButton2 != NULL){
            SDL_DestroyTexture(waveButton2);
            waveButton2 = NULL;
        }
        if(shopButton1 != NULL){
            SDL_DestroyTexture(shopButton1);
            shopButton1 = NULL;
        }
        if(shopButton2 != NULL){
            SDL_DestroyTexture(shopButton2);
            shopButton2 = NULL;
        }
        if(tutorialButton1 != NULL){
            SDL_DestroyTexture(tutorialButton1);
            tutorialButton1 = NULL;
        }
        if(tutorialButton2 != NULL){
            SDL_DestroyTexture(tutorialButton2);
            tutorialButton2 = NULL;
        }
        if(settingsButton1 != NULL){
            SDL_DestroyTexture(settingsButton1);
            settingsButton1 = NULL;
        }
        if(settingsButton2 != NULL){
            SDL_DestroyTexture(settingsButton2);
            settingsButton2 = NULL;
        }
        if(infoButton1 != NULL){
            SDL_DestroyTexture(infoButton1);
            infoButton1 = NULL;
        }
        if(infoButton2 != NULL){
            SDL_DestroyTexture(infoButton2);
            infoButton2 = NULL;
        }
        if(exitButton1 != NULL){
            SDL_DestroyTexture(exitButton1);
            exitButton1 = NULL;
        }
        if(exitButton2 != NULL){
            SDL_DestroyTexture(exitButton2);
            exitButton2 = NULL;
        }
    }

}

void TextureManager::allocate_common() {

```

```

        if (backButton1 == NULL) {
            backButton1 = IMG_LoadTexture((*gameData).renderer,
"../images/other/BackButton1.png");
        }
        if (backButton2 == NULL) {
            backButton2 = IMG_LoadTexture((*gameData).renderer,
"../images/other/BackButton2.png");
        }
        if (backButtonBounds == NULL) {
            backButtonBounds = IMG_LoadTexture((*gameData).renderer,
"../images/other/BackButtonBounds.png");
        }
    }
    void TextureManager::deallocate_common() {

        if (backButton1 != NULL) {
            SDL_DestroyTexture(backButton1);
            backButton1 = NULL;
        }
        if (backButton2 != NULL) {
            SDL_DestroyTexture(backButton2);
            backButton2 = NULL;
        }
        if (backButtonBounds != NULL) {
            SDL_DestroyTexture(backButtonBounds);
            backButtonBounds = NULL;
        }
    }
}

```

A.22 Файл Label.h

```

#pragma once

#include "Component.h"

class Label : public Component{
public:

    std::string text = "";
    SDL_Texture *texture = NULL;
    SDL_Rect destinationRect;

    float widthByHeightRatio = 0;
    float heightFactor = 0;

    Label();
    ~Label() override;

    void events() override;
    void render() override;

```

```

        void set_text(std::string textValue, SDL_Color colorValue);
        void set_size(float heightFactorValue);
        void delete_texture();
};

```

A.23 Файл Label.cpp

```

#include "Label.h"
#include "GameData.h"
#include "GameObject.h"

Label::Label() : Component("Label") {

}

Label::~Label() {

    delete_texture();

}

void Label::events() {

    // Resizing event
    switch(parent->gameData->event.type) {
        case SDL_WINDOWEVENT:
            switch(parent->gameData->event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size(heightFactor);
                    break;
            }
            break;
    }

}

void Label::render() {

    SDL_RenderCopy(parent->gameData->renderer, texture, NULL,
&destinationRect);

}

void Label::set_size(float heightFactorValue) {

    heightFactor = heightFactorValue;

    destinationRect.h = heightFactor * parent->gameData->screenHeight;
    destinationRect.w = destinationRect.h * widthByHeightRatio;
    destinationRect.x = parent->transform.position.x -
destinationRect.w / 2.0;
}

```

```

        destinationRect.y      =      parent->transform.position.y      -
destinationRect.h / 2.0;

    }

    void Label::set_text(std::string textValue, SDL_Color colorValue){

        text = textValue;

        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "0");
        SDL_Surface *surface = TTF_RenderText_Solid(parent->gameData-
>font, text.c_str(), colorValue);
        delete_texture();
        texture = SDL_CreateTextureFromSurface(parent->gameData->renderer,
surface);

        SDL_FreeSurface(surface);
        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1");

        int texW;
        int texH;
        SDL_QueryTexture(texture, NULL, NULL, &texW, &texH);
        widthByHeightRatio = ((float)texW) / ((float)texH);

    }

    void Label::delete_texture(){

        if(texture != NULL){
            SDL_DestroyTexture(texture);
        }

    }

```

A.24 Файл Image.h

```

#pragma once

#include "Component.h"

class Image : public Component{

    public:

        SDL_Texture *texture;
        SDL_Rect destinationRect;

        float heightFactor = 0;
        bool relative = true;

        Image();
        ~Image() override;

        void events() override;
        void render() override;

        void set_texture(SDL_Texture *textureValue);

```

```

        void set_size();
        void set_height_factor(float heightFactorValue);
};

```

A.25 Файл Image.cpp

```

#include "Image.h"
#include "GameData.h"
#include "GameObject.h"

Image::Image() : Component("Image") {
    texture = NULL;
}
Image::~Image() {
}

void Image::events() {
    // Resizing event
    switch(parent->gameData->event.type) {
        case SDL_WINDOWEVENT:
            switch(parent->gameData->event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }
}

void Image::render() {
    SDL_RenderCopy(parent->gameData->renderer, texture, NULL,
&destinationRect);
}

void Image::set_texture(SDL_Texture *textureValue) {
    texture = textureValue;
}
void Image::set_size() {
    if(relative == true){
        destinationRect.h = heightFactor * parent->gameData-
>screenHeight;
        destinationRect.w = destinationRect.h;
    }
}

```

```

        destinationRect.x      =      parent->transform.position.x      -
destinationRect.w / 2.0;
        destinationRect.y      =      parent->transform.position.y      -
destinationRect.h / 2.0;
    }
    else{
        // Do nothing
    }
}
void Image::set_height_factor(float heightFactorValue){
    heightFactor = heightFactorValue;
}

```

A.26 Файл Text.h

```

#pragma once

#include "Component.h"

class Text : public Component{
public:
    std::string text = "";
    SDL_Color color;

    SDL_Texture *texture = NULL;
    SDL_Rect destinationRect;
    SDL_Rect *parentDstRect;

    float heightByWidthRatio = 0.0f;
    float heightFactor = 0.0f;
    float width = 0.0f;
    Uint32 wrapLength = 0;

    Text();
    ~Text() override;

    void events() override;
    void render() override;

    void set_text(std::string textValue, SDL_Color colorValue);
    void set_size(float heightFactorValue);
    void delete_texture();
};

```

A.27 Файл Text.cpp

```

#include "Text.h"
#include "GameData.h"
#include "GameObject.h"

```

```

Text::Text() : Component("Text") {

}

Text::~Text() {
    delete_texture();
}

void Text::events() {
    // Resizing event
    switch(parent->gameData->event.type) {
        case SDL_WINDOWEVENT:
            switch(parent->gameData->event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size(heightFactor);
                    break;
            }
            break;
    }
}

void Text::render() {
    SDL_RenderCopy(parent->gameData->renderer, texture, NULL,
&destinationRect);
}

void Text::set_size(float heightFactorValue) {
    set_text(text, color);

    heightFactor = heightFactorValue;

    destinationRect.w = heightFactor * parentDstRect->w;
    destinationRect.h = destinationRect.w * heightByWidthRatio;

    destinationRect.x = parent->transform.position.x;
    destinationRect.y = parent->transform.position.y;
}

void Text::set_text(std::string textValue, SDL_Color colorValue) {
    text = textValue;
    color = colorValue;

    SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "0");
    SDL_Surface *surface = TTF_RenderUTF8_Blended_Wrapped(parent-
>gameData->font, text.c_str(), color, wrapLength);
    delete_texture();
    texture = SDL_CreateTextureFromSurface(parent->gameData->renderer,
surface);
    SDL_FreeSurface(surface);
}

```

```

        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1");

        int texW;
        int texH;
        SDL_QueryTexture(texture, NULL, NULL, &texW, &texH);
        heightByWidthRatio = ((float)texH) / ((float)texW);
    }

void Text::delete_texture() {
    if(texture != NULL) {
        SDL_DestroyTexture(texture);
    }
}

```

A.28 Файл BackButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class BackButton : public GameObject{
public:

    bool mouseIn;
    bool click;
    EventState nextEvent;
    float heightFactor = 0;

    SDL_Rect boundsRect;
    float rotationAngle = 0.0f;
    float rotationSpeed = 30.0f;
    int rotationSign = 1;

    BackButton(std::string nameValue, EventState
objectEventValue,   GameData *gameDataValue,   GameObject *parentValue,
EventState nextEventValue);
    ~BackButton() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
    bool is_mouse_inside_circle(int mouseX, int mouseY, int
circleX, int circleY, int radius);
};

```

A.29 Файл BackButton.cpp

```

#include "BackButton.h"
#include "Image.h"
#include "GameData.h"

BackButton::BackButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue,
EventState nextEventValue) : GameObject(nameValue, objectEventValue,
gameDataValue, parentValue){

    nextEvent = nextEventValue;
    mouseIn = false;
    click = false;

}

BackButton::~BackButton() {

}

void BackButton::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;

    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent){
        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(is_mouse_inside_circle(mousePoint.x,
mousePoint.y, boundsRect.x + boundsRect.w / 2.0f, boundsRect.y + boundsRect.h
/ 2.0f, boundsRect.w / 2.0f) == true){
                    mouseIn = true;
                    ((Image *)components[0])->texture =
gameData->textureManager.backButton2;

```

```

        rotationSign = -1;
    }
    else{
        mouseIn = false;
        ((Image *)components[0])->texture =
gameData->textureManager.backButton1;
        rotationSign = 1;
    }
    break;

    case SDL_MOUSEBUTTONDOWN:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(is_mouse_inside_circle(mousePoint.x,
mousePoint.y, boundsRect.x + boundsRect.w / 2.0f, boundsRect.y + boundsRect.h
/ 2.0f, boundsRect.w / 2.0f) == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){
            click = true;
        }
        break;

    case SDL_MOUSEBUTTONUP:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(is_mouse_inside_circle(mousePoint.x,
mousePoint.y, boundsRect.x + boundsRect.w / 2.0f, boundsRect.y + boundsRect.h
/ 2.0f, boundsRect.w / 2.0f) == true && click == true &&
(*gameData).event.button.button == SDL_BUTTON_LEFT){
            mouseIn = false;
            ((Image *)components[0])->texture =
gameData->textureManager.backButton1;
            gameData->currentEvent = nextEvent;
            parent->destroy();
        }
        click = false;
        break;
    }
}

}

void BackButton::update() {
    rotationAngle += rotationSign * rotationSpeed * gameData-
>deltaTime;
    rotationAngle = glm::clamp(rotationAngle, 0.0f, 360.0f);
    if(rotationAngle == 360.0f){
        rotationAngle -= 360.0f;
    }
    else if(rotationAngle == 0.0f){
        rotationAngle += 360.0f;
    }
}

void BackButton::render() {

```

```

        SDL_RenderCopyEx(parent->gameData->renderer,          gameData-
>textureManager.backButtonBounds,  NULL,  &boundsRect,  rotationAngle,  NULL,
SDL_FLIP_NONE);
    }

    void BackButton::set_size(){

        boundsRect.h = heightFactor * gameData->screenHeight;
        boundsRect.h *= 1.3f;
        boundsRect.w = boundsRect.h;
        boundsRect.x = transform.position.x - boundsRect.w / 2.0f;
        boundsRect.y = transform.position.y - boundsRect.h / 2.0f;
    }

    bool BackButton::is_mouse_inside_circle(int mouseX, int mouseY, int
circleX, int circleY, int radius){

        int distanceSquared = (mouseX - circleX) * (mouseX - circleX) +
(mouseY - circleY) * (mouseY - circleY);
        return distanceSquared <= (radius * radius);
    }

```

A.30 Файл TutorialRoom.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

class TutorialRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    TutorialRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~TutorialRoom() override;

    void events() override;
    void render() override;

    void set_size();
};

```

A.31 Файл TutorialRoom.cpp

```

#include "TutorialRoom.h"
#include "GameData.h"
#include "Label.h"
#include "Text.h"
#include "Image.h"
#include "BackButton.h"

TutorialRoom::TutorialRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    // Title
    add_child(new GameObject("Title", TUTORIAL_ROOM, gameData, this));
    children[0]->add_component(new Label());

    add_child(new GameObject("Main text", TUTORIAL_ROOM, gameData,
this));
    children[1]->add_component(new Text());

    add_child(new BackButton("Back button", TUTORIAL_ROOM, gameData,
this, MAIN_MENU));
    children[2]->add_component(new Image());

    set_size();

    ((BackButton *)children[2])->heightFactor = 0.07f;
    ((BackButton *)children[2])->set_size();

    ((Label *)children[0]->components[0])->set_text("TUTORIAL", {0, 0,
0, 255});
    ((Label *)children[0]->components[0])->set_size(0.08);

    ((Text *)children[1]->components[0])->parentDstRect = &uiDstrect2;
    ((Text *)children[1]->components[0])->set_text("You have a movable
character. It can shoot bullets. Kill enemies to earn some money. With this
money you can buy towers. Those towers will protect your base. Survive as
many days as you can. Good luck!", {255, 255, 255, 255});
    ((Text *)children[1]->components[0])->set_size(0.8);

    ((Image *)children[2]->components[0])->set_texture(gameData-
>textureManager.backButton1);
    ((Image *)children[2]->components[0])->set_height_factor(((BackButton *)children[2])->heightFactor);
    ((Image *)children[2]->components[0])->set_size();
}

TutorialRoom::~TutorialRoom(){

}

void TutorialRoom::events(){

    // Resizing event

```

```

        switch ((*gameData).event.type) {
            case SDL_WINDOWEVENT:
                switch ((*gameData).event.window.event) {
                    case SDL_WINDOWEVENT_RESIZED:

                        set_size();

                        break;

                }

                break;

        }

    }

void TutorialRoom::render() {

    SDL_SetRenderDrawColor(gameData->renderer, 255, 255, 255, 255);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 240);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect2);

}

void TutorialRoom::set_size() {

    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;
    uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

    uiDstrect2.h = 0.65f * gameData->screenHeight;
    uiDstrect2.w = uiDstrect1.w;
    uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
    uiDstrect2.x = uiDstrect1.x;

    children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
    children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

    children[1]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.1f;
    children[1]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.1f;

    ((Text *)children[1]->components[0])->wrapLength = uiDstrect2.w *
2;

    children[2]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.925f;
    children[2]->transform.position.y = uiDstrect1.y + uiDstrect1.h *
0.5f;

}

```

A.32 Файл InfoRoom.h

```
#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

class InfoRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    InfoRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~InfoRoom() override;

    void events() override;
    void render() override;

    void set_size();
};
```

A.33 Файл InfoRoom.cpp

```
#include "InfoRoom.h"
#include "GameData.h"
#include "Label.h"
#include "Text.h"
#include "Image.h"
#include "BackButton.h"

InfoRoom::InfoRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    // Title
    add_child(new GameObject("Title", INFO_ROOM, gameData, this));
    children[0]->add_component(new Label());

    add_child(new GameObject("Main text", INFO_ROOM, gameData, this));
    children[1]->add_component(new Text());

    add_child(new BackButton("Back button", INFO_ROOM, gameData, this,
MAIN_MENU));
    children[2]->add_component(new Image());
```

```

        set_size();

        ((BackButton *)children[2])->heightFactor = 0.07f;
        ((BackButton *)children[2])->set_size();

        ((Label *)children[0]->components[0])->set_text("INFO", {0, 0, 0,
255});
        ((Label *)children[0]->components[0])->set_size(0.08);

        ((Text *)children[1]->components[0])->parentDstRect = &uiDstrect2;
        ((Text *)children[1]->components[0])->set_text("This game is
created by Yaroslav Kolokol. The game uses SDL, box2d, GLM and SQLite
libraries. Thank you for playing :)", {255, 255, 255, 255});
        ((Text *)children[1]->components[0])->set_size(0.8);

        ((Image *)children[2]->components[0])->set_texture(gameData-
>textureManager.backButton1);
        ((Image *)children[2]->components[0])->
>set_height_factor(((BackButton *)children[2])->heightFactor);
        ((Image *)children[2]->components[0])->set_size();
    }
    InfoRoom::~InfoRoom() {

    }

    void InfoRoom::events() {

        // Resizing event
        switch((*gameData).event.type) {
            case SDL_WINDOWEVENT:
                switch((*gameData).event.window.event) {
                    case SDL_WINDOWEVENT_RESIZED:

                        set_size();

                        break;

                }

            break;
        }
    }

    void InfoRoom::render() {

        SDL_SetRenderDrawColor(gameData->renderer, 255, 255, 255, 255);
        SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

        SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 240);
        SDL_RenderFillRect(gameData->renderer, &uiDstrect2);
    }

```

```

void InfoRoom::set_size(){
    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;
    uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

    uiDstrect2.h = 0.65f * gameData->screenHeight;
    uiDstrect2.w = uiDstrect1.w;
    uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
    uiDstrect2.x = uiDstrect1.x;

    children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
    children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

    children[1]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.1f;
    children[1]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.1f;

    ((Text *)children[1]->components[0])->wrapLength = uiDstrect2.w *
2;

    children[2]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.925f;
    children[2]->transform.position.y = uiDstrect1.y + uiDstrect1.h *
0.5f;
}

```

A.34 Файл ExitRoom.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

class ExitRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    ExitRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);

```

```

        ~ExitRoom() override;

        void events() override;
        void render() override;

        void set_size();
};

```

A.35 Файл ExitRoom.cpp

```

#include "ExitRoom.h"
#include "GameData.h"
#include "Label.h"
#include "Text.h"
#include "Image.h"
#include "BackButton.h"
#include "ExitButton.h"

ExitRoom::ExitRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    // Title
    add_child(new GameObject("Title", EXIT_ROOM, gameData, this));
    children[0]->add_component(new Label());

    add_child(new GameObject("Main text", EXIT_ROOM, gameData, this));
    children[1]->add_component(new Label());

    add_child(new BackButton("Back button", EXIT_ROOM, gameData, this,
MAIN_MENU));
    children[2]->add_component(new Image());

    add_child(new ExitButton("Game button", EXIT_ROOM, gameData,
this));
    children[3]->add_component(new Label());

    set_size();

    ((BackButton *)children[2])->heightFactor = 0.07f;
    ((BackButton *)children[2])->set_size();

    ((ExitButton *)children[3])->heightFactor = 0.05f;
    ((ExitButton *)children[3])->set_size();

    ((Label *)children[0]->components[0])->set_text("EXIT", {0, 0, 0,
255});
    ((Label *)children[0]->components[0])->set_size(0.08);

    ((Label *)children[1]->components[0])->set_text("Do you want to
exit?", {255, 255, 255, 255});

```

```

        ((Label *)children[1]->components[0])->set_size(0.06);

        ((Image *)children[2]->components[0])->set_texture(gameData-
>textureManager.backButton1);
        ((Image *)children[2]->components[0])->
>set_height_factor(((BackButton *)children[2])->heightFactor);
        ((Image *)children[2]->components[0])->set_size();

        ((Label *)children[3]->components[0])->set_text("Quit", {255, 255,
255, 255});
        ((Label *)children[3]->components[0])->set_size(((ExitButton
*)children[3])->heightFactor);
    }
    ExitRoom::~ExitRoom() {

    }

void ExitRoom::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:

                    set_size();

                    break;

            }

            break;

    }

}

void ExitRoom::render() {

    SDL_SetRenderDrawColor(gameData->renderer, 255, 255, 255, 255);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 240);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect2);

}

void ExitRoom::set_size() {

    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;
    uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

    uiDstrect2.h = 0.65f * gameData->screenHeight;
    uiDstrect2.w = uiDstrect1.w;
    uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
    uiDstrect2.x = uiDstrect1.x;

```

```

        children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
        children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

        children[1]->transform.position.x = uiDstrect2.x + uiDstrect2.w /
2.0f;
        children[1]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.4;

        children[2]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.925f;
        children[2]->transform.position.y = uiDstrect1.y + uiDstrect1.h *
0.5f;

        children[3]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
        children[3]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.85f;
    }

```

A.36 Файл ExitButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class ExitButton : public GameObject{
public:

    bool mouseIn;
    bool click;
    SDL_Color color1 = {20, 20, 20, 255};
    SDL_Rect imageDstrect;
    float heightFactor = 0;
    EventState nextEvent;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;
    SDL_Rect cornerRect4;

    SDL_Rect cornerRect5;
    SDL_Rect cornerRect6;

    float displacement = 0.0f;

    ExitButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~ExitButton() override;

    void events() override;
    void update() override;

```

```

        void render() override;

        void set_size();
};

```

A.37 Файл ExitButton.cpp

```

#include "ExitButton.h"
#include "GameData.h"
#include "Label.h"

```

```

ExitButton::ExitButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

```

```

    nextEvent = TOWER_DEFENSE;
    mouseIn = false;
    click = false;

```

```

}

```

```

ExitButton::~ExitButton() {

```

```

}

```

```

void ExitButton::events() {

```

```

    // Resizing event
    switch ((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch ((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
        break;
    }

```

```

}

```

```

// Button clicking
if ((*gameData).currentEvent == objectEvent) {
    switch ((*gameData).event.type) {

```

```

        SDL_Point mousePoint;

```

```

        case SDL_MOUSEMOTION:

```

```

            mousePoint.x = (*gameData).event.motion.x;
            mousePoint.y = (*gameData).event.motion.y;

```

```

        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE){
            mouseIn = true;
            color1 = {255, 255, 255, 255};
            ((Label *)components[0])->set_text(((Label
*)components[0])->text, {0, 0, 0, 255});
        }
        else{
            mouseIn = false;
            color1 = {20, 20, 20, 255};
            ((Label *)components[0])->set_text(((Label
*)components[0])->text, {255, 255, 255, 255});
        }

        break;
    case SDL_MOUSEBUTTONDOWN:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT){
            click = true;
        }

        break;
    case SDL_MOUSEBUTTONUP:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

            gameData->active = false;

        }

        click = false;

        break;
    }
}

void ExitButton::update(){
    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, imageDstrect.h /
2.0f);

    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;

```

```

        cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f +
displacement;
        cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
        cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;

    }
    void ExitButton::render(){

        SDL_SetRenderDrawColor(gameData->renderer, color1.r, color1.g,
color1.b, color1.a);
        SDL_RenderFillRect(gameData->renderer, &imageDstrect);

        SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
        SDL_RenderFillRect(gameData->renderer, &cornerRect1);
        SDL_RenderFillRect(gameData->renderer, &cornerRect2);
        SDL_RenderFillRect(gameData->renderer, &cornerRect3);
        SDL_RenderFillRect(gameData->renderer, &cornerRect4);
        SDL_RenderFillRect(gameData->renderer, &cornerRect5);
        SDL_RenderFillRect(gameData->renderer, &cornerRect6);

    }

    void ExitButton::set_size(){

        imageDstrect.h = (heightFactor + 0.03) * gameData->screenHeight;
        imageDstrect.w = imageDstrect.h * 4;
        imageDstrect.x = transform.position.x - imageDstrect.w / 2.0f;
        imageDstrect.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect1.w = imageDstrect.w * 0.030f;
        cornerRect1.h = imageDstrect.h;
        cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect1.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect2.w = imageDstrect.w * 0.1f;
        cornerRect2.h = imageDstrect.w * 0.030f;
        cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect2.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect2.h;

        cornerRect3.w = imageDstrect.w * 0.1f;
        cornerRect3.h = imageDstrect.w * 0.030f;
        cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect3.y = transform.position.y + imageDstrect.h / 2.0f;

        cornerRect4.w = imageDstrect.w * 0.030f;
        cornerRect4.h = imageDstrect.h;
        cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f;
        cornerRect4.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect5.w = imageDstrect.w * 0.1f;
        cornerRect5.h = imageDstrect.w * 0.030f;
        cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w;
        cornerRect5.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect5.h;

        cornerRect6.w = imageDstrect.w * 0.1f;

```

```

        cornerRect6.h = imageDstrect.w * 0.030f;
        cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect6.w + cornerRect4.w;
        cornerRect6.y = transform.position.y + imageDstrect.h / 2.0f;
    }

```

A.38 Файл GameRoom.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"
// #include "BackButton.h"
// #include "EnterButton.h"

class GameRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    GameRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~GameRoom() override;

    void events() override;
    void render() override;

    void set_size();
};

```

A.39 Файл GameRoom.cpp

```

#include "GameRoom.h"
#include "GameData.h"
#include "Label.h"
#include "Text.h"
#include "Image.h"
#include "BackButton.h"
#include "GameButton.h"

GameRoom::GameRoom(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    // Title

```

```

        add_child(new GameObject("Title", GAME_ROOM, gameData, this));
        children[0]->add_component(new Label());

        add_child(new GameObject("Main text", GAME_ROOM, gameData, this));
        children[1]->add_component(new Text());

        add_child(new BackButton("Back button", GAME_ROOM, gameData, this,
MAIN_MENU));
        children[2]->add_component(new Image());

        add_child(new GameButton("Game button", GAME_ROOM, gameData,
this));
        children[3]->add_component(new Label());

        set_size();

        ((BackButton *)children[2])->heightFactor = 0.07f;
        ((BackButton *)children[2])->set_size();

        ((GameButton *)children[3])->heightFactor = 0.05f;
        ((GameButton *)children[3])->set_size();

        ((Label *)children[0]->components[0])->set_text("GAME ROOM", {0,
0, 0, 255});
        ((Label *)children[0]->components[0])->set_size(0.08);

        ((Text *)children[1]->components[0])->parentDstRect = &uiDstrect2;
        ((Text *)children[1]->components[0])->set_text("Game room is a
place, where you can stay alone with yourself and mind your strategy before
the game.", {255, 255, 255, 255});
        ((Text *)children[1]->components[0])->set_size(0.8);

        ((Image *)children[2]->components[0])->set_texture(gameData-
>textureManager.backButton1);
        ((Image *)children[2]->components[0])->
>set_height_factor(((BackButton *)children[2])->heightFactor);
        ((Image *)children[2]->components[0])->set_size();

        ((Label *)children[3]->components[0])->set_text("Enter", {255,
255, 255, 255});
        ((Label *)children[3]->components[0])->set_size(((GameButton
*)children[3])->heightFactor);
    }
    GameRoom::~GameRoom() {

    }

void GameRoom::events() {

    // Resizing event
    switch((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:

                    set_size();

```

```

        break;
    }
    break;
}

void GameRoom::render() {
    SDL_SetRenderDrawColor(gameData->renderer, 255, 255, 255, 255);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 240);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect2);
}

void GameRoom::set_size() {
    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;
    uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

    uiDstrect2.h = 0.65f * gameData->screenHeight;
    uiDstrect2.w = uiDstrect1.w;
    uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
    uiDstrect2.x = uiDstrect1.x;

    children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
    children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

    children[1]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.1f;
    children[1]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.1f;

    ((Text *)children[1]->components[0])->wrapLength = uiDstrect2.w *
2;

    children[2]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.925f;
    children[2]->transform.position.y = uiDstrect1.y + uiDstrect1.h *
0.5f;

    children[3]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
    children[3]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.85f;
}

```

A.40 Файл GameButton.h

```
#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class GameButton : public GameObject{
public:
    bool mouseIn;
    bool click;
    SDL_Color color1 = {20, 20, 20, 255};
    SDL_Rect imageDstrect;
    float heightFactor = 0;
    EventState nextEvent;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;
    SDL_Rect cornerRect4;

    SDL_Rect cornerRect5;
    SDL_Rect cornerRect6;

    float displacement = 0.0f;

    GameButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~GameButton() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
};
```

A.41 Файл GameButton.cpp

```
#include "GameButton.h"
#include "GameData.h"
#include "Label.h"
#include "Root.h"

GameButton::GameButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    nextEvent = TOWER_DEFENSE;
    mouseIn = false;
    click = false;
```

```

    }

    GameButton::~GameButton() {

    }

void GameButton::events() {

    // Resizing event
    switch((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent) {
        switch((*gameData).event.type) {

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE) {
                    mouseIn = true;
                    color1 = {255, 255, 255, 255};
                    ((Label *)components[0])->set_text(((Label
*)components[0])->text, {0, 0, 0, 255});
                }
                else{
                    mouseIn = false;
                    color1 = {20, 20, 20, 255};
                    ((Label *)components[0])->set_text(((Label
*)components[0])->text, {255, 255, 255, 255});
                }

                break;
            case SDL_MOUSEBUTTONDOWN:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT) {
                    click = true;
                }

                break;
            case SDL_MOUSEBUTTONUP:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);

```

```

        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

            gameData->currentEvent = nextEvent;
            mouseIn = false;

            ((Root *)parent->parent->parent)-
>delete_main_menu();
            ((Root *)parent->parent->parent)-
>create_tower_defense();

        }
        click = false;

        break;

    }

}

void GameButton::update() {

    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, imageDstrect.h /
2.0f);

    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f +
displacement;
    cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
    cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;

}

void GameButton::render() {

    SDL_SetRenderDrawColor(gameData->renderer, color1.r, color1.g,
color1.b, color1.a);
    SDL_RenderFillRect(gameData->renderer, &imageDstrect);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &cornerRect1);
    SDL_RenderFillRect(gameData->renderer, &cornerRect2);
    SDL_RenderFillRect(gameData->renderer, &cornerRect3);
    SDL_RenderFillRect(gameData->renderer, &cornerRect4);

```

```

        SDL_RenderFillRect(gameData->renderer, &cornerRect5);
        SDL_RenderFillRect(gameData->renderer, &cornerRect6);
    }

    void GameButton::set_size() {
        imageDstrect.h = (heightFactor + 0.03) * gameData->screenHeight;
        imageDstrect.w = imageDstrect.h * 4;
        imageDstrect.x = transform.position.x - imageDstrect.w / 2.0f;
        imageDstrect.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect1.w = imageDstrect.w * 0.030f;
        cornerRect1.h = imageDstrect.h;
        cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect1.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect2.w = imageDstrect.w * 0.1f;
        cornerRect2.h = imageDstrect.w * 0.030f;
        cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect2.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect2.h;

        cornerRect3.w = imageDstrect.w * 0.1f;
        cornerRect3.h = imageDstrect.w * 0.030f;
        cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect3.y = transform.position.y + imageDstrect.h / 2.0f;

        cornerRect4.w = imageDstrect.w * 0.030f;
        cornerRect4.h = imageDstrect.h;
        cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f;
        cornerRect4.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect5.w = imageDstrect.w * 0.1f;
        cornerRect5.h = imageDstrect.w * 0.030f;
        cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w;
        cornerRect5.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect5.h;

        cornerRect6.w = imageDstrect.w * 0.1f;
        cornerRect6.h = imageDstrect.w * 0.030f;
        cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect6.w + cornerRect4.w;
        cornerRect6.y = transform.position.y + imageDstrect.h / 2.0f;
    }
}

```

A.42 Файл MainMenu.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

```

```

class MainMenu : public GameObject{
public:

    // Background cell
    SDL_Rect cellDstrect = {0, 0, 0, 0};

    // Main logo
    SDL_Rect logoDstrect = {0, 0, 0, 0};
    float upperPosition = 0.0f;
    float lowerPosition = 0.0f;

    float middlePosition = 0.0f;
    float logoSpeed = 0.5f;
    float logoSign = 1.0f;

    MainMenu(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~MainMenu() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();

};

```

A.43 Файл MainMenu.cpp

```

#include "MainMenu.h"
#include "GameData.h"
#include "Label.h"
#include "Image.h"
#include "MainButton.h"
// #include "GameRoom.h"

MainMenu::MainMenu(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    // Main Titles
    add_child(new GameObject("Main Title 1", MAIN_MENU, gameData,
this));
    children[0]->add_component(new Label());
    add_child(new GameObject("Main Title 2", MAIN_MENU, gameData,
this));

```

```

        children[1]->add_component(new Label());

        // Main buttons
        add_child(new MainButton("Game button", MAIN_MENU, gameData, this,
4, 2, GAME_ROOM));
        children[2]->add_component(new Image());
        add_child(new MainButton("Tutorial button", MAIN_MENU, gameData,
this, 4, 0, TUTORIAL_ROOM));
        children[3]->add_component(new Image());
        add_child(new MainButton("Info button", MAIN_MENU, gameData, this,
-3, 2, INFO_ROOM));
        children[4]->add_component(new Image());
        add_child(new MainButton("Exit button", MAIN_MENU, gameData, this,
-3, 0, EXIT_ROOM));
        children[5]->add_component(new Image());

        // Set up
        set_size();

        ((MainButton *)children[2])->set_size();
        ((MainButton *)children[2])->texture1 = gameData-
>textureManager.waveButton1;
        ((MainButton *)children[2])->texture2 = gameData-
>textureManager.waveButton2;

        ((MainButton *)children[3])->set_size();
        ((MainButton *)children[3])->texture1 = gameData-
>textureManager.tutorialButton1;
        ((MainButton *)children[3])->texture2 = gameData-
>textureManager.tutorialButton2;

        ((MainButton *)children[4])->set_size();
        ((MainButton *)children[4])->texture1 = gameData-
>textureManager.infoButton1;
        ((MainButton *)children[4])->texture2 = gameData-
>textureManager.infoButton2;

        ((MainButton *)children[5])->set_size();
        ((MainButton *)children[5])->texture1 = gameData-
>textureManager.exitButton1;
        ((MainButton *)children[5])->texture2 = gameData-
>textureManager.exitButton2;

        ((Label *)children[0]->components[0])->set_text("SHAPES", {255,
255, 255, 255});
        ((Label *)children[0]->components[0])->set_size(0.11);
        ((Label *)children[1]->components[0])->set_text("TOWERS & YOU",
{255, 255, 255, 255});
        ((Label *)children[1]->components[0])->set_size(0.05);

        ((Image *)children[2]->components[0])->set_texture(((MainButton
*)children[2])->texture1);
        ((Image *)children[2]->components[0])->set_height_factor(0.13);
        ((Image *)children[2]->components[0])->set_size();

        ((Image *)children[3]->components[0])->set_texture(((MainButton
*)children[3])->texture1);

```

```

        ((Image *)children[3]->components[0])->set_height_factor(0.13);
        ((Image *)children[3]->components[0])->set_size();

        ((Image *)children[4]->components[0])->set_texture(((MainButton
*)children[4])->texture1);
        ((Image *)children[4]->components[0])->set_height_factor(0.13);
        ((Image *)children[4]->components[0])->set_size();

        ((Image *)children[5]->components[0])->set_texture(((MainButton
*)children[5])->texture1);
        ((Image *)children[5]->components[0])->set_height_factor(0.13);
        ((Image *)children[5]->components[0])->set_size();

    }

MainMenu::~MainMenu() {

    printf("MainMenu is closed\n");

}

void MainMenu::update() {

    middlePosition += logoSpeed * logoSign * gameData->deltaTime;
    middlePosition = glm::clamp(middlePosition, 0.0f, 1.0f);
    if(middlePosition == 1.0f || middlePosition == 0.0f){
        logoSign *= -1;
    }

    logoDstrect.y = middlePosition * (lowerPosition - upperPosition) +
upperPosition;

}

void MainMenu::render(){

    for(int j = 0; j < parent->gameData->screenHeight / cellDstrect.h
+ 2; ++j){
        for(int i = 0; i < parent->gameData->screenWidth /
cellDstrect.w + 2; ++i){
            SDL_RenderCopy(gameData->renderer, gameData-
>textureManager.cellTexture, NULL, &cellDstrect);
            cellDstrect.x += cellDstrect.w;
        }
        cellDstrect.y += cellDstrect.h;

        cellDstrect.x = (parent->gameData->screenWidth -
cellDstrect.w * (parent->gameData->screenWidth / cellDstrect.w + 2)) / 2;

    }

    cellDstrect.y = (parent->gameData->screenHeight - cellDstrect.h *
(parent->gameData->screenHeight / cellDstrect.h + 2)) / 2;

```

```

        SDL_RenderCopy((*gameData).renderer, gameData-
>textureManager.radialShade, NULL, NULL);
        SDL_RenderCopy((*gameData).renderer, gameData-
>textureManager.logo, NULL, &logoDstrect);
    }

    void MainMenu::events() {

        // Resizing event
        switch((*gameData).event.type) {
            case SDL_WINDOWEVENT:
                switch((*gameData).event.window.event) {
                    case SDL_WINDOWEVENT_RESIZED:
                        set_size();

                        break;
                }

                break;
        }
    }

    void MainMenu::set_size() {

        cellDstrect.h = gameData->screenHeight * 0.13;
        cellDstrect.w = cellDstrect.h;
        cellDstrect.x = gameData->screenWidth / 2 - (gameData->screenWidth
+ cellDstrect.w * 2) / 2;
        cellDstrect.y = gameData->screenHeight / 2 - (gameData-
>screenHeight + cellDstrect.h * 2) / 2;

        logoDstrect.h = 0.3 * gameData->screenHeight;
        logoDstrect.w = logoDstrect.h;
        logoDstrect.x = gameData->screenWidth / 2 - (logoDstrect.w) / 2;
        upperPosition = gameData->screenHeight / 3.5 - (logoDstrect.h) /
2;
        lowerPosition = gameData->screenHeight / 3 - (logoDstrect.h) / 2;

        children[0]->transform.position.x = gameData->screenWidth / 2.0f;
        children[0]->transform.position.y = gameData->screenHeight /
1.93f;

        children[1]->transform.position.x = gameData->screenWidth / 2.0f;
        children[1]->transform.position.y = gameData->screenHeight /
1.67f;

        ((MainButton *)children[2])->cellRect = cellDstrect;
        ((MainButton *)children[3])->cellRect = cellDstrect;
        ((MainButton *)children[4])->cellRect = cellDstrect;
        ((MainButton *)children[5])->cellRect = cellDstrect;
    }

```

A.44 Файл MainButton.h

```
#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class MainButton : public GameObject{
public:

    SDL_Texture *texture1;
    SDL_Texture *texture2;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;
    SDL_Rect cornerRect4;

    SDL_Rect cornerRect5;
    SDL_Rect cornerRect6;

    SDL_Rect cornerRect7;
    SDL_Rect cornerRect8;

    float displacement = 0.0f;

    int offsetX;
    int offsetY;

    bool mouseIn;
    bool click;
    SDL_Rect cellRect;

    EventState nextEvent;

    MainButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, int
offsetXValue, int offsetYValue, EventState nextEventValue);
    ~MainButton() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();

};
```

A.45 Файл MainButton.cpp

```

#include "MainButton.h"
#include "Image.h"
#include "GameData.h"
#include "GameRoom.h"
#include "TutorialRoom.h"
#include "InfoRoom.h"
#include "ExitRoom.h"

MainButton::MainButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, int
offsetXValue, int offsetYValue, EventState nextEventValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    nextEvent = nextEventValue;
    offsetX = offsetXValue;
    offsetY = offsetYValue;

    mouseIn = false;
    click = false;

}

MainButton::~MainButton() {

}

void MainButton::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent){
        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;

```

```

        if(SDL_PointInRect(&mousePoint, &((Image
*)components[0])->destinationRect) == SDL_TRUE){
            mouseIn = true;
            ((Image *)components[0])->texture =
texture2;
        }
        else{
            mouseIn = false;
            ((Image *)components[0])->texture =
texture1;
        }
        break;

        case SDL_MOUSEBUTTONDOWN:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if(SDL_PointInRect(&mousePoint, &((Image
*)components[0])->destinationRect) == SDL_TRUE &&
(*gameData).event.button.button == SDL_BUTTON_LEFT){
                click = true;
            }
            break;

        case SDL_MOUSEBUTTONUP:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if(SDL_PointInRect(&mousePoint, &((Image
*)components[0])->destinationRect) == SDL_TRUE && click == true &&
(*gameData).event.button.button == SDL_BUTTON_LEFT){

                if(nextEvent == GAME_ROOM){
                    parent->add_child(new GameRoom("Game
Room", GAME_ROOM, gameData, parent));
                }
                else if(nextEvent == TUTORIAL_ROOM){
                    parent->add_child(new
TutorialRoom("Tutorial Room", TUTORIAL_ROOM, gameData, parent));
                }
                else if(nextEvent == INFO_ROOM){
                    parent->add_child(new InfoRoom("Info
Room", INFO_ROOM, gameData, parent));
                }
                else if(nextEvent == EXIT_ROOM){
                    parent->add_child(new ExitRoom("Exit
Room", EXIT_ROOM, gameData, parent));
                }

                gameData->currentEvent = nextEvent;
                mouseIn = false;
                ((Image *)components[0])->texture =
texture1;
            }
            click = false;
            break;
    }

}
}

```

```

    }

void MainButton::update() {
    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, cellRect.h / 4.0f);

    cornerRect1.x = transform.position.x - cellRect.w / 2.0f -
displacement;
    cornerRect1.y = transform.position.y - cellRect.h / 2.0f -
displacement;

    cornerRect2.x = transform.position.x - cellRect.w / 2.0f -
displacement;
    cornerRect2.y = transform.position.y - cellRect.h / 2.0f -
displacement;

    cornerRect3.x = transform.position.x + cellRect.w / 2.0f +
displacement - cornerRect3.w;
    cornerRect3.y = transform.position.y - cellRect.h / 2.0f -
displacement;

    cornerRect4.x = transform.position.x + cellRect.w / 2.0f +
displacement - cornerRect4.w;
    cornerRect4.y = transform.position.y - cellRect.h / 2.0f -
displacement;

    cornerRect5.x = transform.position.x + cellRect.w / 2.0f +
displacement - cornerRect5.w;
    cornerRect5.y = transform.position.y + cellRect.h / 2.0f +
displacement - cornerRect5.h;

    cornerRect6.x = transform.position.x + cellRect.w / 2.0f +
displacement - cornerRect6.w;
    cornerRect6.y = transform.position.y + cellRect.h / 2.0f +
displacement - cornerRect6.h;

    cornerRect7.x = transform.position.x - cellRect.w / 2.0f -
displacement;
    cornerRect7.y = transform.position.y + cellRect.h / 2.0f +
displacement - cornerRect7.h;

    cornerRect8.x = transform.position.x - cellRect.w / 2.0f -
displacement;
    cornerRect8.y = transform.position.y + cellRect.h / 2.0f +
displacement - cornerRect8.h;
}

void MainButton::render() {
    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &cornerRect1);
    SDL_RenderFillRect(gameData->renderer, &cornerRect2);
    SDL_RenderFillRect(gameData->renderer, &cornerRect3);

```

```

        SDL_RenderFillRect(gameData->renderer, &cornerRect4);
        SDL_RenderFillRect(gameData->renderer, &cornerRect5);
        SDL_RenderFillRect(gameData->renderer, &cornerRect6);
        SDL_RenderFillRect(gameData->renderer, &cornerRect7);
        SDL_RenderFillRect(gameData->renderer, &cornerRect8);
    }

    void MainButton::set_size(){

        int tempX = (gameData->screenWidth - cellRect.w * (gameData-
>screenWidth / cellRect.w + 2)) / 2;

        while(tempX + cellRect.w * offsetX < gameData->screenWidth / 2){
            tempX += cellRect.w;
        }

        int tempY = (gameData->screenHeight - cellRect.h * (gameData-
>screenHeight / cellRect.h + 2)) / 2;

        while(tempY + cellRect.h * offsetY < gameData->screenHeight / 2){
            tempY += cellRect.h;
        }

        if((gameData->screenWidth / cellRect.w) % 2 != 0 && offsetX < 0){
            tempX -= cellRect.w;
        }

        transform.position.x = tempX + cellRect.w / 2.0f;
        transform.position.y = tempY + cellRect.h / 2.0f;

        cornerRect1.w = cellRect.w * 0.050f;
        cornerRect1.h = cellRect.h * 0.30f;
        cornerRect1.x = transform.position.x - cellRect.w / 2.0f;
        cornerRect1.y = transform.position.y - cellRect.h / 2.0f;

        cornerRect2.w = cellRect.w * 0.30f;
        cornerRect2.h = cellRect.h * 0.050f;
        cornerRect2.x = transform.position.x - cellRect.w / 2.0f;
        cornerRect2.y = transform.position.y - cellRect.h / 2.0f;

        cornerRect3.w = cellRect.w * 0.30f;
        cornerRect3.h = cellRect.h * 0.050f;
        cornerRect3.x = transform.position.x + cellRect.w / 2.0f -
cornerRect3.w;
        cornerRect3.y = transform.position.y - cellRect.h / 2.0f;

        cornerRect4.w = cellRect.w * 0.050f;
        cornerRect4.h = cellRect.h * 0.30f;
        cornerRect4.x = transform.position.x + cellRect.w / 2.0f -
cornerRect4.w;
        cornerRect4.y = transform.position.y - cellRect.h / 2.0f;

        cornerRect5.w = cellRect.w * 0.050f;
        cornerRect5.h = cellRect.h * 0.30f;
        cornerRect5.x = transform.position.x + cellRect.w / 2.0f -
cornerRect5.w;
        cornerRect5.y = transform.position.y + cellRect.h / 2.0f -
cornerRect5.h;

        cornerRect6.w = cellRect.w * 0.30f;
        cornerRect6.h = cellRect.h * 0.050f;

```

```

        cornerRect6.x = transform.position.x + cellRect.w / 2.0f -
cornerRect6.w;
        cornerRect6.y = transform.position.y + cellRect.h / 2.0f -
cornerRect6.h;

        cornerRect7.w = cellRect.w * 0.30f;
        cornerRect7.h = cellRect.h * 0.050f;
        cornerRect7.x = transform.position.x - cellRect.w / 2.0f;
        cornerRect7.y = transform.position.y + cellRect.h / 2.0f -
cornerRect7.h;

        cornerRect8.w = cellRect.w * 0.050f;
        cornerRect8.h = cellRect.h * 0.30f;
        cornerRect8.x = transform.position.x - cellRect.w / 2.0f;
        cornerRect8.y = transform.position.y + cellRect.h / 2.0f -
cornerRect8.h;
    }

```

A.46 Файл BuildButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class BuildButton : public GameObject{
public:

    bool mouseIn;
    bool click;
    SDL_Color color1 = {20, 20, 20, 255};
    SDL_Rect imageDstrect;
    float heightFactor = 0;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;
    SDL_Rect cornerRect4;

    SDL_Rect cornerRect5;
    SDL_Rect cornerRect6;

    float displacement = 0.0f;

    BuildButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~BuildButton() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
};

```

A.47 Файл BuildButton.cpp

```

#include "BuildButton.h"
#include "GameData.h"
#include "Label.h"
#include "TowerRoom.h"
#include "SmallWall.h"
#include "BigWall.h"
#include "GunTower1.h"
#include "GunTower2.h"

BuildButton::BuildButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    mouseIn = false;
    click = false;

}

BuildButton::~~BuildButton() {

}

void BuildButton::events() {

    // Resizing event
    switch ((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch ((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }

    // Button clicking
    if ((*gameData).currentEvent == objectEvent) {
        switch ((*gameData).event.type) {

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if (SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE) {

```

```

        mouseIn = true;
        color1 = {255, 255, 255, 255};
        ((Label *)components[0])->set_text(((Label
*)components[0])->text, {0, 0, 0, 255});
    }
    else{
        mouseIn = false;
        color1 = {20, 20, 20, 255};
        ((Label *)components[0])->set_text(((Label
*)components[0])->text, {255, 255, 255, 255});
    }

    break;
    case SDL_MOUSEBUTTONDOWN:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT){
            click = true;
        }

        break;
    case SDL_MOUSEBUTTONUP:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

            if(gameData->towers.amount[ ((TowerRoom
*)parent)->currentIndex] > 0){

                gameData->currentEvent =
TOWER_DEFENSE;

                --gameData->
>towers.amount[ ((TowerRoom *)parent)->currentIndex];
                parent->destroy();
                if(gameData->towers.name[ ((TowerRoom
*)parent)->currentIndex] == "Small Wall"){
                    parent->parent->children[0]-
>add_child(new SmallWall("Unit Small wall", TOWER_DEFENSE, gameData, parent-
>parent->children[0], true));
                }
                else if(gameData-
>towers.name[ ((TowerRoom *)parent)->currentIndex] == "Big Wall"){
                    parent->parent->children[0]-
>add_child(new BigWall("Unit Big wall", TOWER_DEFENSE, gameData, parent-
>parent->children[0], true));
                }
                else if(gameData-
>towers.name[ ((TowerRoom *)parent)->currentIndex] == "Gun Tower 1"){
                    parent->parent->children[0]-
>add_child(new GunTower1("Unit Gun Tower 1", TOWER_DEFENSE, gameData, parent-
>parent->children[0], true));
                }
                else if(gameData-
>towers.name[ ((TowerRoom *)parent)->currentIndex] == "Gun Tower 2"){
                    parent->parent->children[0]-
>add_child(new GunTower2("Unit Gun Tower 2", TOWER_DEFENSE, gameData, parent-
>parent->children[0], true));
                }
            }
        }
    }
}

```

```

        click = false;

        break;
    }

}

}

void BuildButton::update() {
    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, imageDstrect.h /
2.0f);

    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f +
displacement;
    cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
    cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
}
void BuildButton::render() {
    SDL_SetRenderDrawColor(gameData->renderer, color1.r, color1.g,
color1.b, color1.a);
    SDL_RenderFillRect(gameData->renderer, &imageDstrect);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &cornerRect1);
    SDL_RenderFillRect(gameData->renderer, &cornerRect2);
    SDL_RenderFillRect(gameData->renderer, &cornerRect3);
    SDL_RenderFillRect(gameData->renderer, &cornerRect4);
    SDL_RenderFillRect(gameData->renderer, &cornerRect5);
    SDL_RenderFillRect(gameData->renderer, &cornerRect6);
}

void BuildButton::set_size() {
    imageDstrect.h = (heightFactor + 0.03) * gameData->screenHeight;
    imageDstrect.w = imageDstrect.h * 4;
    imageDstrect.x = transform.position.x - imageDstrect.w / 2.0f;
    imageDstrect.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect1.w = imageDstrect.w * 0.030f;

```

```

        cornerRect1.h = imageDstrect.h;
        cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect1.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect2.w = imageDstrect.w * 0.1f;
        cornerRect2.h = imageDstrect.w * 0.030f;
        cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect2.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect2.h;

        cornerRect3.w = imageDstrect.w * 0.1f;
        cornerRect3.h = imageDstrect.w * 0.030f;
        cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect3.y = transform.position.y + imageDstrect.h / 2.0f;

        cornerRect4.w = imageDstrect.w * 0.030f;
        cornerRect4.h = imageDstrect.h;
        cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f;
        cornerRect4.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect5.w = imageDstrect.w * 0.1f;
        cornerRect5.h = imageDstrect.w * 0.030f;
        cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w;
        cornerRect5.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect5.h;

        cornerRect6.w = imageDstrect.w * 0.1f;
        cornerRect6.h = imageDstrect.w * 0.030f;
        cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect6.w + cornerRect4.w;
        cornerRect6.y = transform.position.y + imageDstrect.h / 2.0f;
    }

```

A.48 Файл BuyButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class BuyButton : public GameObject{
public:

    bool mouseIn;
    bool click;
    SDL_Color color1 = {20, 20, 20, 255};
    SDL_Rect imageDstrect;
    float heightFactor = 0;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;

```

```

        SDL_Rect cornerRect4;

        SDL_Rect cornerRect5;
        SDL_Rect cornerRect6;

        float displacement = 0.0f;

        BuyButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
        ~BuyButton() override;

        void events() override;
        void update() override;
        void render() override;

        void set_size();
};

```

A.49 Файл BuyButton.cpp

```

#include "BuyButton.h"
#include "GameData.h"
#include "Label.h"
#include "TowerRoom.h"

BuyButton::BuyButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    mouseIn = false;
    click = false;

}

BuyButton::~~BuyButton(){

}

void BuyButton::events(){

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }
}

```

```

        // Button clicking
        if((*gameData).currentEvent == objectEvent){
            switch((*gameData).event.type){

                SDL_Point mousePoint;

                case SDL_MOUSEMOTION:

                    mousePoint.x = (*gameData).event.motion.x;
                    mousePoint.y = (*gameData).event.motion.y;
                    if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE){

                        mouseIn = true;
                        color1 = {255, 255, 255, 255};
                        ((Label *)components[0])->set_text(((Label
*)components[0])->text, {0, 0, 0, 255});
                    }
                    else{
                        mouseIn = false;
                        color1 = {20, 20, 20, 255};
                        ((Label *)components[0])->set_text(((Label
*)components[0])->text, {255, 255, 255, 255});
                    }

                    break;
                case SDL_MOUSEBUTTONDOWN:
                    SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                    if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT){
                        click = true;
                    }

                    break;
                case SDL_MOUSEBUTTONUP:
                    SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                    if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

                        if(gameData->userData.money >= gameData-
>towers.cost[((TowerRoom *)parent)->currentIndex]){
                            ++gameData-
>towers.amount[((TowerRoom *)parent)->currentIndex];
                            ((Label *)parent->children[7]-
>components[0])->set_text("Amount: " + std::to_string(gameData-
>towers.amount[((TowerRoom *)parent)->currentIndex]), {255, 255, 255, 255});
                            ((Label *)parent->children[7]-
>components[0])->set_size(0.06);
                            gameData->userData.money -=
gameData->towers.cost[((TowerRoom *)parent)->currentIndex];
                        }

                    }

                    click = false;

                    break;
            }
        }
    }
}

```

```

void BuyButton::update() {
    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, imageDstrect.h /
2.0f);

    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f +
displacement;
    cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
    cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;

}
void BuyButton::render(){

    SDL_SetRenderDrawColor(gameData->renderer, color1.r, color1.g,
color1.b, color1.a);
    SDL_RenderFillRect(gameData->renderer, &imageDstrect);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &cornerRect1);
    SDL_RenderFillRect(gameData->renderer, &cornerRect2);
    SDL_RenderFillRect(gameData->renderer, &cornerRect3);
    SDL_RenderFillRect(gameData->renderer, &cornerRect4);
    SDL_RenderFillRect(gameData->renderer, &cornerRect5);
    SDL_RenderFillRect(gameData->renderer, &cornerRect6);

}

void BuyButton::set_size(){

    imageDstrect.h = (heightFactor + 0.03) * gameData->screenHeight;
    imageDstrect.w = imageDstrect.h * 4;
    imageDstrect.x = transform.position.x - imageDstrect.w / 2.0f;
    imageDstrect.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect1.w = imageDstrect.w * 0.030f;
    cornerRect1.h = imageDstrect.h;
    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
    cornerRect1.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect2.w = imageDstrect.w * 0.1f;
    cornerRect2.h = imageDstrect.w * 0.030f;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
    cornerRect2.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect2.h;

    cornerRect3.w = imageDstrect.w * 0.1f;

```

```

        cornerRect3.h = imageDstrect.w * 0.030f;
        cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
        cornerRect3.y = transform.position.y + imageDstrect.h / 2.0f;

        cornerRect4.w = imageDstrect.w * 0.030f;
        cornerRect4.h = imageDstrect.h;
        cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f;
        cornerRect4.y = transform.position.y - imageDstrect.h / 2.0f;

        cornerRect5.w = imageDstrect.w * 0.1f;
        cornerRect5.h = imageDstrect.w * 0.030f;
        cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w;
        cornerRect5.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect5.h;

        cornerRect6.w = imageDstrect.w * 0.1f;
        cornerRect6.h = imageDstrect.w * 0.030f;
        cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect6.w + cornerRect4.w;
        cornerRect6.y = transform.position.y + imageDstrect.h / 2.0f;
    }

```

A.50 Файл ChangeTower.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

class ChangeTower : public GameObject{
public:
    SDL_Rect imageRect = {0, 0, 0, 0};
    bool isLeft = true;
    float angle = 0;
    bool mouseIn = false;
    bool click = false;

    int *targetIndex = NULL;

    ChangeTower(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, int
*indexValue);
    ~ChangeTower() override;

    void events() override;
    void render() override;

    void set_size();
};

```

A.51 Файл ChangeTower.cpp

```

#include "ChangeTower.h"
#include "GameData.h"
#include "Label.h"
#include "Image.h"

ChangeTower::ChangeTower(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, int
*targetIndexValue) : GameObject(nameValue, objectEventValue, gameDataValue,
parentValue){

    targetIndex = targetIndexValue;

}

ChangeTower::~ChangeTower() {

}

void ChangeTower::events() {

    // Resizing event
    switch(gameData->event.type){
        case SDL_WINDOWEVENT:
            switch(gameData->event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }

            break;

    }

    if(gameData->currentEvent == objectEvent){
        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(SDL_PointInRect(&mousePoint, &imageRect) ==

SDL_TRUE){

                    mouseIn = true;
                }
                else{
                    mouseIn = false;
                }

                break;

            case SDL_MOUSEBUTTONDOWN:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);

```

```

        if(SDL_PointInRect(&mousePoint, &imageRect) ==
SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT){
            click = true;
        }

        break;

        case SDL_MOUSEBUTTONDOWN:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if(SDL_PointInRect(&mousePoint, &imageRect) ==
SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

                if((*targetIndex) > 0 && isLeft == true){
                    --(*targetIndex);
                    ((Label *)parent->children[2]-
>components[0])->set_text(gameData->towers.name[*targetIndex], {255, 255,
255, 255});
                    ((Label *)parent->children[2]-
>components[0])->set_size(0.06);

                    ((Image *)parent->children[5]-
>components[0])->set_texture(gameData->towers.image[*targetIndex]);
                    ((Image *)parent->children[5]-
>components[0])->set_height_factor(0.2);
                    ((Image *)parent->children[5]-
>components[0])->set_size();

                    ((Label *)parent->children[6]-
>components[0])->set_text("Cost: " + std::to_string(gameData-
>towers.cost[*targetIndex]), {255, 255, 255, 255});
                    ((Label *)parent->children[6]-
>components[0])->set_size(0.06);

                    ((Label *)parent->children[7]-
>components[0])->set_text("Amount: " + std::to_string(gameData-
>towers.amount[*targetIndex]), {255, 255, 255, 255});
                    ((Label *)parent->children[7]-
>components[0])->set_size(0.06);
                }
                else if((*targetIndex) < gameData-
>towers.AMOUNT - 1 && isLeft == false){
                    ++(*targetIndex);
                    ((Label *)parent->children[2]-
>components[0])->set_text(gameData->towers.name[*targetIndex], {255, 255,
255, 255});
                    ((Label *)parent->children[2]-
>components[0])->set_size(0.06);

                    ((Image *)parent->children[5]-
>components[0])->set_texture(gameData->towers.image[*targetIndex]);
                    ((Image *)parent->children[5]-
>components[0])->set_height_factor(0.2);
                    ((Image *)parent->children[5]-
>components[0])->set_size();

                    ((Label *)parent->children[6]-
>components[0])->set_text("Cost: " + std::to_string(gameData-
>towers.cost[*targetIndex]), {255, 255, 255, 255});
                    ((Label *)parent->children[6]-
>components[0])->set_size(0.06);

```

```

        ((Label *)parent->children[7]-
>components[0])->set_text("Amount: " + std::to_string(gameData-
>towers.amount[*targetIndex]), {255, 255, 255, 255});
        ((Label *)parent->children[7]-
>components[0])->set_size(0.06);
    }
    }
    click = false;
    break;
}
}

void ChangeTower::render() {
    SDL_RenderCopyEx((*gameData).renderer, gameData-
>textureManager.changeTower, NULL, &imageRect, angle, NULL, SDL_FLIP_NONE);
}

void ChangeTower::set_size() {
    imageRect.h = 0.1f * gameData->screenHeight;
    imageRect.w = imageRect.h;
    imageRect.x = transform.position.x - imageRect.w / 2.0f;
    imageRect.y = transform.position.y - imageRect.h / 2.0f;
}

```

A.52 Файл DayButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class DayButton : public GameObject{
public:
    // Graphics
    SDL_Rect mainRect = {0, 0, 0, 0};
    SDL_Color mainColor = {0, 0, 0, 220};

    SDL_Rect textRect = {0, 0, 0, 0};
    SDL_Color textColor = {255, 255, 255, 255};
    SDL_Texture *textTexture = NULL;
    std::string text = "Day ";

    SDL_Rect border1 = {0, 0, 0, 0};
    SDL_Rect border2 = {0, 0, 0, 0};
    SDL_Rect border3 = {0, 0, 0, 0};
    SDL_Rect border4 = {0, 0, 0, 0};

    // Common
    bool mouseIn = false;
    bool click = false;
}

```

```

        float heightFactor = 0.05f;
        float widthByHeightRatio = 0;

        DayButton(std::string      nameValue,      EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
        ~DayButton() override;

        void events() override;
        void render() override;

        void set_size();
        void set_text();
        void delete_texture();
};

```

A.53 Файл DayButton.cpp

```

#include "DayButton.h"
#include "GameData.h"
#include "EnemyManager.h"

DayButton::DayButton(std::string      nameValue,      EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    set_text();
    set_size();

    printf("DayButton is created\n");
}
DayButton::~~DayButton() {

    delete_texture();

}

void DayButton::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_text();
                    set_size();
                    break;
            }
            break;

    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent){

```

```

        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE){

                    mainColor = {255, 255, 255, 220};
                    textColor = {0, 0, 0, 255};
                    set_text();

                }
                else if((*gameData).dayActive == false){
                    mainColor = {0, 0, 0, 220};
                    textColor = {255, 255, 255, 255};
                    set_text();

                }

                break;

            case SDL_MOUSEBUTTONDOWN:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                if(SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_RIGHT){
                    click = true;

                }

                break;

            case SDL_MOUSEBUTTONUP:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                if(SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_RIGHT && (*gameData).dayActive == false){

                    (*gameData).dayActive = true;
                    mainColor = {255, 255, 255, 220};
                    textColor = {0, 0, 0, 255};
                    set_text();
                    ((EnemyManager *)parent->children[1])->
>spawn_enemies(gameData->userData.dayCounter + 1);

                }

                click = false;
                break;

        }

    }

}

void DayButton::render(){

    SDL_SetRenderDrawColor((*gameData).renderer, mainColor.r,
mainColor.g, mainColor.b, mainColor.a);
    SDL_RenderFillRect((*gameData).renderer, &mainRect);
    SDL_RenderCopy((*gameData).renderer, textTexture, NULL,
&textRect);

```

```

        SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
        SDL_RenderFillRect(gameData->renderer, &border1);
        SDL_RenderFillRect(gameData->renderer, &border2);
        SDL_RenderFillRect(gameData->renderer, &border3);
        SDL_RenderFillRect(gameData->renderer, &border4);
    }

void DayButton::set_size(){

    transform.position.x = 0.1f * gameData->screenWidth;
    transform.position.y = 0.1f * gameData->screenHeight;

    textRect.h = heightFactor * gameData->screenHeight;
    textRect.w = textRect.h * widthByHeightRatio;
    textRect.x = transform.position.x - textRect.w / 2.0;
    textRect.y = transform.position.y - textRect.h / 2.0;

    mainRect.w = textRect.w + textRect.w * 0.4f;
    mainRect.h = textRect.h + textRect.h * 0.6f;
    mainRect.x = transform.position.x - mainRect.w / 2.0;
    mainRect.y = transform.position.y - mainRect.h / 2.0;

    border1.w = mainRect.w;
    border1.h = mainRect.h * 0.1f;
    border1.x = mainRect.x;
    border1.y = mainRect.y - border1.h;

    border2.w = mainRect.w;
    border2.h = mainRect.h * 0.1f;
    border2.x = mainRect.x;
    border2.y = mainRect.y + mainRect.h;

    border3.w = mainRect.h * 0.1f;
    border3.h = mainRect.h + border2.h * 2.0f;
    border3.x = mainRect.x - border3.w;
    border3.y = mainRect.y - border2.h;

    border4.w = mainRect.h * 0.1f;
    border4.h = mainRect.h + border2.h * 2.0f;
    border4.x = mainRect.x + mainRect.w;
    border4.y = mainRect.y - border2.h;
}

void DayButton::set_text(){

    SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "0");
    SDL_Surface *surface = TTF_RenderText_Solid(gameData->font, (text
+ std::to_string((*gameData).userData.dayCounter + 1)).c_str(), textColor);
    delete_texture();
    textTexture = SDL_CreateTextureFromSurface(gameData->renderer,
surface);

    SDL_FreeSurface(surface);
    SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1");

    int texW;
    int texH;
    SDL_QueryTexture(textTexture, NULL, NULL, &texW, &texH);
    widthByHeightRatio = ((float)texW) / ((float)texH);

```

```

}
void DayButton::delete_texture() {
    if(textTexture != NULL) {
        SDL_DestroyTexture(textTexture);
    }
}

```

A.54 Файл FinishButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class FinishButton : public GameObject{
public:
    bool mouseIn;
    bool click;
    SDL_Color color1 = {20, 20, 20, 255};
    SDL_Rect imageDstrect;
    float heightFactor = 0;
    EventState nextEvent;

    SDL_Rect cornerRect1;
    SDL_Rect cornerRect2;

    SDL_Rect cornerRect3;
    SDL_Rect cornerRect4;

    SDL_Rect cornerRect5;
    SDL_Rect cornerRect6;

    float displacement = 0.0f;

    FinishButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~FinishButton() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
};

```

A.55 Файл FinishButton.cpp

```

#include "FinishButton.h"
#include "GameData.h"
#include "Label.h"

```

```

#include "Root.h"

FinishButton::FinishButton(std::string      nameValue,      EventState
objectEventValue,      GameData *gameDataValue,      GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    nextEvent = MAIN_MENU;
    mouseIn = false;
    click = false;

}

FinishButton::~FinishButton() {

}

void FinishButton::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;

    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent){
        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(SDL_PointInRect(&mousePoint,      &imageDstrect)
== SDL_TRUE){

                    mouseIn = true;
                    color1 = {255, 255, 255, 255};
                    ((Label *)components[0])->set_text(((Label
*)components[0])->text, {0, 0, 0, 255});
                }
                else{
                    mouseIn = false;
                    color1 = {20, 20, 20, 255};
                    ((Label *)components[0])->set_text(((Label
*)components[0])->text, {255, 255, 255, 255});
                }

            break;

        }
    }
}

```

```

        case SDL_MOUSEBUTTONDOWN:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_LEFT){
                click = true;
            }

            break;
        case SDL_MOUSEBUTTONUP:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if(SDL_PointInRect(&mousePoint, &imageDstrect)
== SDL_TRUE && click == true && (*gameData).event.button.button ==
SDL_BUTTON_LEFT){

                gameData->currentEvent = nextEvent;
                gameData->userData.dayCounter = 0;
                mouseIn = false;
                (*gameData).dayActive = false;
                ((Root *)parent->parent->parent)-
>delete_tower_defense();
                ((Root *)parent->parent->parent)-
>create_main_menu();

            }
            click = false;

            break;
        }
    }
}

void FinishButton::update() {
    if(mouseIn == true){
        displacement += 200.0f * gameData->deltaTime;
    }
    else{
        displacement -= 200.0f * gameData->deltaTime;
    }
    displacement = glm::clamp(displacement, 0.0f, imageDstrect.h /
2.0f);

    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w - displacement;
    cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f +
displacement;
    cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
    cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w + displacement;
}

```

```

void FinishButton::render(){
    SDL_SetRenderDrawColor(gameData->renderer, color1.r, color1.g,
color1.b, color1.a);
    SDL_RenderFillRect(gameData->renderer, &imageDstrect);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &cornerRect1);
    SDL_RenderFillRect(gameData->renderer, &cornerRect2);
    SDL_RenderFillRect(gameData->renderer, &cornerRect3);
    SDL_RenderFillRect(gameData->renderer, &cornerRect4);
    SDL_RenderFillRect(gameData->renderer, &cornerRect5);
    SDL_RenderFillRect(gameData->renderer, &cornerRect6);
}

void FinishButton::set_size(){
    imageDstrect.h = (heightFactor + 0.03) * gameData->screenHeight;
    imageDstrect.w = imageDstrect.h * 4;
    imageDstrect.x = transform.position.x - imageDstrect.w / 2.0f;
    imageDstrect.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect1.w = imageDstrect.w * 0.030f;
    cornerRect1.h = imageDstrect.h;
    cornerRect1.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
    cornerRect1.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect2.w = imageDstrect.w * 0.1f;
    cornerRect2.h = imageDstrect.w * 0.030f;
    cornerRect2.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
    cornerRect2.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect2.h;

    cornerRect3.w = imageDstrect.w * 0.1f;
    cornerRect3.h = imageDstrect.w * 0.030f;
    cornerRect3.x = transform.position.x - imageDstrect.w / 2.0f -
cornerRect1.w;
    cornerRect3.y = transform.position.y + imageDstrect.h / 2.0f;

    cornerRect4.w = imageDstrect.w * 0.030f;
    cornerRect4.h = imageDstrect.h;
    cornerRect4.x = transform.position.x + imageDstrect.w / 2.0f;
    cornerRect4.y = transform.position.y - imageDstrect.h / 2.0f;

    cornerRect5.w = imageDstrect.w * 0.1f;
    cornerRect5.h = imageDstrect.w * 0.030f;
    cornerRect5.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect5.w + cornerRect4.w;
    cornerRect5.y = transform.position.y - imageDstrect.h / 2.0f -
cornerRect5.h;

    cornerRect6.w = imageDstrect.w * 0.1f;
    cornerRect6.h = imageDstrect.w * 0.030f;
    cornerRect6.x = transform.position.x + imageDstrect.w / 2.0f -
cornerRect6.w + cornerRect4.w;
    cornerRect6.y = transform.position.y + imageDstrect.h / 2.0f;
}

```

A.56 Файл FinishRoom.h

```
#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

class FinishRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    FinishRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~FinishRoom() override;

    void events() override;
    void render() override;

    void set_size();
};
```

A.57 Файл FinishRoom.cpp

```
#include "FinishRoom.h"
#include "GameData.h"
#include "Label.h"
#include "FinishButton.h"

FinishRoom::FinishRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){
    if ((*gameData).userData.dayCounter >
(*gameData).userData.maxDays){
        (*gameData).userData.maxDays =
(*gameData).userData.dayCounter;
    }
    if ((*gameData).userData.money > (*gameData).userData.maxMoney){
        (*gameData).userData.maxMoney = (*gameData).userData.money;
    }

    // Title
    add_child(new GameObject("Title", TOWER_DEFENSE, gameData, this));
```

```

        children[0]->add_component(new Label());

        add_child(new FinishButton("Finish button", TOWER_DEFENSE,
gameData, this));
        children[1]->add_component(new Label());

        // Current day
        add_child(new GameObject("Current day", TOWER_DEFENSE, gameData,
this));
        children[2]->add_component(new Label());

        // Current money
        add_child(new GameObject("Current money", TOWER_DEFENSE, gameData,
this));
        children[3]->add_component(new Label());

        // Max days
        add_child(new GameObject("Max days", TOWER_DEFENSE, gameData,
this));
        children[4]->add_component(new Label());

        // Max money
        add_child(new GameObject("Max money", TOWER_DEFENSE, gameData,
this));
        children[5]->add_component(new Label());

        set_size();

        ((FinishButton *)children[1])->heightFactor = 0.05f;
        ((FinishButton *)children[1])->set_size();

        ((Label *)children[0]->components[0])->set_text("RESULTS", {0, 0,
0, 255});
        ((Label *)children[0]->components[0])->set_size(0.08);

        ((Label *)children[1]->components[0])->set_text("Back", {255, 255,
255, 255});
        ((Label *)children[1]->components[0])->set_size(((FinishButton
*)children[1])->heightFactor);

        ((Label *)children[2]->components[0])->set_text("Survived days: "
+ std::to_string((*gameData).userData.dayCounter), {255, 255, 255, 255});
        ((Label *)children[2]->components[0])->set_size(0.06);

        ((Label *)children[3]->components[0])->set_text("Earned money: " +
std::to_string((*gameData).userData.money), {255, 255, 255, 255});
        ((Label *)children[3]->components[0])->set_size(0.06);

        ((Label *)children[4]->components[0])->set_text("Max survived
days: " + std::to_string((*gameData).userData.maxDays), {255, 255, 255,
255});
        ((Label *)children[4]->components[0])->set_size(0.06);

        ((Label *)children[5]->components[0])->set_text("Max earned money:
" + std::to_string((*gameData).userData.maxMoney), {255, 255, 255, 255});
        ((Label *)children[5]->components[0])->set_size(0.06);
    }
    FinishRoom::~FinishRoom() {

```

```

    }

void FinishRoom::events() {
    // Resizing event
    switch((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
        break;
    }
}

void FinishRoom::render() {
    SDL_SetRenderDrawColor(gameData->renderer, 200, 200, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 240);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect2);
}

void FinishRoom::set_size() {
    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;
    uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

    uiDstrect2.h = 0.65f * gameData->screenHeight;
    uiDstrect2.w = uiDstrect1.w;
    uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
    uiDstrect2.x = uiDstrect1.x;

    children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
    children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

    children[1]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
    children[1]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.85f;

    children[2]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
    children[2]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.15f;

    children[3]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;

```

```

        children[3]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.3f;
        children[4]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
        children[4]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.45f;
        children[5]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.5f;
        children[5]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.6f;
    }

```

A.58 Файл TowerButton.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class TowerButton : public GameObject{
public:
    // Graphics
    SDL_Rect mainRect = {0, 0, 0, 0};

    SDL_Rect border1 = {0, 0, 0, 0};
    SDL_Rect border2 = {0, 0, 0, 0};
    SDL_Rect border3 = {0, 0, 0, 0};
    SDL_Rect border4 = {0, 0, 0, 0};

    // Common
    bool mouseIn = false;
    bool click = false;
    float heightFactor = 0.08f;

    TowerButton(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~TowerButton() override;

    void events() override;
    void render() override;

    void set_size();
};

```

A.59 Файл TowerButton.cpp

```

#include "TowerButton.h"

```

```

#include "GameData.h"
#include "TowerRoom.h"

TowerButton::TowerButton(std::string      nameValue,      EventState
objectEventValue,   GameData *gameDataValue,   GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    set_size();

}

TowerButton::~~TowerButton() {

}

void TowerButton::events() {

    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }

    // Button clicking
    if((*gameData).currentEvent == objectEvent){
        switch((*gameData).event.type){

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if(SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE){

                    mouseIn = true;
                }
                else{
                    mouseIn = false;
                }

                break;

            case SDL_MOUSEBUTTONDOWN:
                SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                if(SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE && (*gameData).event.button.button == SDL_BUTTON_RIGHT){
                    click = true;
                }
            }
        }
    }
}

```

```

        break;

        case SDL_MOUSEBUTTONDOWN:
            SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
            if (SDL_PointInRect(&mousePoint, &mainRect) ==
SDL_TRUE    &&    click    ==    true    &&    (*gameData).event.button.button ==
SDL_BUTTON_RIGHT && (*gameData).dayActive == false){

                gameData->currentEvent = TOWER_ROOM;
                parent->add_child(new      TowerRoom("Tower
Room", TOWER_ROOM, gameData, parent));

            }
            click = false;
            break;

        }

    }

}

void TowerButton::render() {

    if(mouseIn == false){
        SDL_RenderCopy((*gameData).renderer,          gameData-
>textureManager.towerButton1, NULL, &mainRect);
    }
    else{
        SDL_RenderCopy((*gameData).renderer,          gameData-
>textureManager.towerButton2, NULL, &mainRect);
    }

    SDL_SetRenderDrawColor(gameData->renderer, 0, 255, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &border1);
    SDL_RenderFillRect(gameData->renderer, &border2);
    SDL_RenderFillRect(gameData->renderer, &border3);
    SDL_RenderFillRect(gameData->renderer, &border4);

}

void TowerButton::set_size() {

    transform.position.x = 0.2f * gameData->screenWidth;
    transform.position.y = 0.1f * gameData->screenHeight;

    mainRect.h = heightFactor * gameData->screenHeight;
    mainRect.w = mainRect.h;
    mainRect.x = transform.position.x - mainRect.w / 2.0;
    mainRect.y = transform.position.y - mainRect.h / 2.0;

    border1.w = mainRect.w;
    border1.h = mainRect.h * 0.1f;

```

```

border1.x = mainRect.x;
border1.y = mainRect.y - border1.h;

border2.w = mainRect.w;
border2.h = mainRect.h * 0.1f;
border2.x = mainRect.x;
border2.y = mainRect.y + mainRect.h;

border3.w = mainRect.h * 0.1f;
border3.h = mainRect.h + border2.h * 2.0f;
border3.x = mainRect.x - border3.w;
border3.y = mainRect.y - border2.h;

border4.w = mainRect.h * 0.1f;
border4.h = mainRect.h + border2.h * 2.0f;
border4.x = mainRect.x + mainRect.w;
border4.y = mainRect.y - border2.h;
}

```

A.60 Файл TowerDefense.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class TowerDefense : public GameObject{
public:
    // Background cell
    SDL_Rect cellRect = {0, 0, 0, 0};

    // Statistics
    SDL_Texture *moneyTexture = NULL;
    SDL_Texture *fpsTexture = NULL;
    SDL_Rect moneyRect = {0, 0, 0, 0};
    SDL_Color moneyColor = {255, 0, 0, 255};
    SDL_Rect fpsRect = {0, 0, 0, 0};
    float widthByHeightRatioMoney = 0;
    float widthByHeightRatioFPS = 0;
    float heightFactorMoney = 0.06;

    TowerDefense(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~TowerDefense() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
    void finish_game();

    void delete_texture(SDL_Texture **textureValue);

```

```

        void set_text(std::string textValue, SDL_Color colorValue,
SDL_Texture **textureValue, float *widthByHeightRatio);

};

```

A.61 Файл TowerDefense.cpp

```

#include "TowerDefense.h"
#include "GameData.h"
#include "UnitManager.h"
#include "Wall.h"
#include "DayButton.h"
#include "TowerButton.h"
#include "EnemyManager.h"
#include "UnitBulletManager.h"
#include "FinishRoom.h"
#include "Label.h"

TowerDefense::TowerDefense(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    printf("TowerDefense is started\n");

    // Unit spawner
    add_child(new UnitManager("Unit manager", TOWER_DEFENSE, gameData,
this));

    // Enemy spawner
    add_child(new EnemyManager("Enemy manager", TOWER_DEFENSE,
gameData, this));

    // Bullet spawner
    add_child(new UnitBulletManager("Bullet manager", TOWER_DEFENSE,
gameData, this));

    // Walls
    add_child(new Wall("Wall", TOWER_DEFENSE, gameData, this, 0, 4, 1,
9));
    add_child(new Wall("Wall", TOWER_DEFENSE, gameData, this, 14.7, 4,
1, 9));
    add_child(new Wall("Wall", TOWER_DEFENSE, gameData, this, 7, 0,
15, 1));
    add_child(new Wall("Wall", TOWER_DEFENSE, gameData, this, 7, 8.5,
15, 1));

    // Day button
    add_child(new DayButton("Day button", TOWER_DEFENSE, gameData,
this));

    // Tower button
    add_child(new TowerButton("Tower button", TOWER_DEFENSE, gameData,
this));

    set_text("Money: " + std::to_string((*gameData).userData.money),
moneyColor, &moneyTexture, &widthByHeightRatioMoney);
    set_size();

```

```

    }
    TowerDefense::~TowerDefense() {

        printf("TowerDefense is ended\n");
    }

    void TowerDefense::render() {
        for(int j = 0; j < parent->gameData->screenHeight / cellRect.h +
2; ++j){
            for(int i = 0; i < parent->gameData->screenWidth /
cellRect.w + 2; ++i){
                SDL_RenderCopy(gameData->renderer,          gameData-
>textureManager.grid, NULL, &cellRect);
                cellRect.x += cellRect.w;
            }
            cellRect.y += cellRect.h;
            cellRect.x = (parent->gameData->screenWidth - cellRect.w *
(parent->gameData->screenWidth / cellRect.w + 2)) / 2;
        }
        cellRect.y = (parent->gameData->screenHeight - cellRect.h *
(parent->gameData->screenHeight / cellRect.h + 2)) / 2;

        SDL_RenderCopy(gameData->renderer,          moneyTexture,          NULL,
&moneyRect);
    }

    void TowerDefense::events() {

        // Resizing event
        switch(parent->gameData->event.type) {
            case SDL_WINDOWEVENT:
                switch(parent->gameData->event.window.event) {
                    case SDL_WINDOWEVENT_RESIZED:
                        set_size();
                        break;
                }
                break;
        }
    }

    void TowerDefense::update() {

        set_text("Money: " + std::to_string((*gameData).userData.money),
moneyColor, &moneyTexture, &widthByHeightRatioMoney);
    }

    void TowerDefense::set_size() {

        cellRect.h = gameData->screenHeight * 0.10;
        cellRect.w = cellRect.h;
    }

```

```

        cellRect.x = gameData->screenWidth / 2 - (gameData->screenWidth +
cellRect.w * 2) / 2;
        cellRect.y = gameData->screenHeight / 2 - (gameData->screenHeight
+ cellRect.h * 2) / 2;

        moneyRect.h = heightFactorMoney * gameData->screenHeight;
        moneyRect.w = moneyRect.h * widthByHeightRatioMoney;
        moneyRect.x = gameData->screenWidth * 0.06;
        moneyRect.y = gameData->screenHeight * 0.18;
    }

    void TowerDefense::finish_game() {

        printf("FINISH!\n");
        delete_children();
        moneyColor = {0, 0, 0, 0};
        moneyRect.x = -1000;

        // Results
        add_child(new    FinishRoom("Results",    TOWER_DEFENSE,    gameData,
this));
    }

    void TowerDefense::delete_texture(SDL_Texture **textureValue) {

        if(*(textureValue) != NULL){
            SDL_DestroyTexture(*(textureValue));
        }
    }

    void    TowerDefense::set_text(std::string    textValue,    SDL_Color
colorValue, SDL_Texture **textureValue, float *widthByHeightRatio){

        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "0");
        SDL_Surface    *surface    =    TTF_RenderText_Solid(gameData->font,
textValue.c_str(), colorValue);
        delete_texture(textureValue);
        *(textureValue) = SDL_CreateTextureFromSurface(gameData->renderer,
surface);

        SDL_FreeSurface(surface);
        SDL_SetHint(SDL_HINT_RENDER_SCALE_QUALITY, "1");

        int texW;
        int texH;
        SDL_QueryTexture(*(textureValue), NULL, NULL, &texW, &texH);
        *(widthByHeightRatio) = ((float)texW) / ((float)texH);
    }
}

```

A.62 Файл TowerRoom.h

```

#pragma once

#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

#include "GameObject.h"

```

```

class TowerRoom : public GameObject{
public:
    // UI
    SDL_Rect uiDstrect1;
    SDL_Rect uiDstrect2;

    int currentIndex = 0;

    TowerRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~TowerRoom() override;

    void events() override;
    void render() override;

    void set_size();

};

```

A.63 Файл TowerRoom.cpp

```

#include "TowerRoom.h"
#include "GameData.h"
#include "Label.h"
#include "Image.h"
#include "BackButton.h"
#include "ChangeTower.h"
#include "BuyButton.h"
#include "BuildButton.h"

TowerRoom::TowerRoom(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    // Title
    add_child(new GameObject("Title", TOWER_ROOM, gameData, this));
    children[0]->add_component(new Label());

    // Back button
    add_child(new BackButton("Back button", TOWER_ROOM, gameData,
this, TOWER_DEFENSE));
    children[1]->add_component(new Image());

    // Tower name
    add_child(new GameObject("Tower name", TOWER_ROOM, gameData,
this));
    children[2]->add_component(new Label());

    // Left changer
    add_child(new ChangeTower("Left button", TOWER_ROOM, gameData,
this, &currentIndex));

    // Right changer

```

```

        add_child(new ChangeTower("Right button", TOWER_ROOM, gameData,
this, &currentIndex));

        // Tower image
        add_child(new GameObject("Tower image", TOWER_ROOM, gameData,
this));
        children[5]->add_component(new Image());

        // Cost of the tower
        add_child(new GameObject("Cost", TOWER_ROOM, gameData, this));
        children[6]->add_component(new Label());

        // Amount of towers
        add_child(new GameObject("Amount", TOWER_ROOM, gameData, this));
        children[7]->add_component(new Label());

        // Buy button
        add_child(new BuyButton("Buy button", TOWER_ROOM, gameData,
this));
        children[8]->add_component(new Label());

        // Build button
        add_child(new BuildButton("Build button", TOWER_ROOM, gameData,
this));
        children[9]->add_component(new Label());

        set_size();

        ((BackButton *)children[1])->heightFactor = 0.07f;
        ((BackButton *)children[1])->set_size();

        ((ChangeTower *)children[4])->isLeft = false;
        ((ChangeTower *)children[4])->angle = 180;

        ((ChangeTower *)children[3])->set_size();
        ((ChangeTower *)children[4])->set_size();

        ((BuyButton *)children[8])->heightFactor = 0.04f;
        ((BuyButton *)children[8])->set_size();

        ((BuildButton *)children[9])->heightFactor = 0.04f;
        ((BuildButton *)children[9])->set_size();

        ((Label *)children[0]->components[0])->set_text("TOWERS", {0, 0,
0, 255});
        ((Label *)children[0]->components[0])->set_size(0.08);

        ((Image *)children[1]->components[0])->set_texture(gameData-
>textureManager.backButton1);
        ((Image *)children[1]->components[0])->set_height_factor(((BackButton *)children[1])->heightFactor);
        ((Image *)children[1]->components[0])->set_size();

        ((Label *)children[2]->components[0])->set_text(gameData-
>towers.name[currentIndex], {255, 255, 255, 255});
        ((Label *)children[2]->components[0])->set_size(0.06);

        ((Image *)children[5]->components[0])->set_texture(gameData-
>towers.image[currentIndex]);
        ((Image *)children[5]->components[0])->set_height_factor(0.2);
        ((Image *)children[5]->components[0])->set_size();

```

```

        ((Label *)children[6]->components[0])->set_text("Cost: " +
std::to_string(gameData->towers.cost[currentIndex]), {255, 255, 255, 255});
        ((Label *)children[6]->components[0])->set_size(0.06);

        ((Label *)children[7]->components[0])->set_text("Amount: " +
std::to_string(gameData->towers.amount[currentIndex]), {255, 255, 255, 255});
        ((Label *)children[7]->components[0])->set_size(0.06);

        ((Label *)children[8]->components[0])->set_text("Buy", {255, 255,
255, 255});
        ((Label *)children[8]->components[0])->set_size(((BuyButton
*)children[8])->heightFactor);

        ((Label *)children[9]->components[0])->set_text("Build", {255,
255, 255, 255});
        ((Label *)children[9]->components[0])->set_size(((BuyButton
*)children[9])->heightFactor);
    }
    TowerRoom::~TowerRoom() {

    }

void TowerRoom::events() {

    // Resizing event
    switch((*gameData).event.type) {
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event) {
                case SDL_WINDOWEVENT_RESIZED:

                    set_size();

                    break;

            }

            break;
    }
}

void TowerRoom::render() {

    SDL_SetRenderDrawColor(gameData->renderer, 200, 200, 200, 255);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect1);

    SDL_SetRenderDrawColor(gameData->renderer, 0, 0, 0, 200);
    SDL_RenderFillRect(gameData->renderer, &uiDstrect2);

}

void TowerRoom::set_size() {

    uiDstrect1.h = 0.15f * gameData->screenHeight;
    uiDstrect1.w = 0.6f * gameData->screenWidth;
    uiDstrect1.y = 0.1f * gameData->screenHeight;

```

```

        uiDstrect1.x = gameData->screenWidth / 2.0f - uiDstrect1.w / 2.0f;

        uiDstrect2.h = 0.65f * gameData->screenHeight;
        uiDstrect2.w = uiDstrect1.w;
        uiDstrect2.y = uiDstrect1.y + uiDstrect1.h;
        uiDstrect2.x = uiDstrect1.x;

        children[0]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
        children[0]->transform.position.y = uiDstrect1.y + uiDstrect1.h /
2.0f;

        children[1]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.925f;
        children[1]->transform.position.y = uiDstrect1.y + uiDstrect1.h *
0.5f;

        children[2]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
        children[2]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.1f;

        children[3]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.25f;
        children[3]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.4f;

        children[4]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.75f;
        children[4]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.4f;

        children[5]->transform.position.x = uiDstrect1.x + uiDstrect1.w /
2.0f;
        children[5]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.4f;

        children[6]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.20f;
        children[6]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.7f;

        children[7]->transform.position.x = uiDstrect1.x + uiDstrect1.w *
0.80f;
        children[7]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.7f;

        children[8]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.20f;
        children[8]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.85f;

        children[9]->transform.position.x = uiDstrect2.x + uiDstrect2.w *
0.80f;
        children[9]->transform.position.y = uiDstrect2.y + uiDstrect2.h *
0.85f;
    }

```

A.64 Файл Towers.h

```
#pragma once

#include <string>
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>

class GameData;

class Towers{
public:
    // Parent
    GameData *gameData;

    // Data
    static const int AMOUNT = 4;
    std::string name[AMOUNT];
    SDL_Texture *image[AMOUNT];
    int cost[AMOUNT];
    int amount[AMOUNT];

    Towers(GameData *gameDataValue);
    ~Towers();

    void set_images();
};
```

A.65 Файл Towers.cpp

```
#include "Towers.h"
#include "GameData.h"

Towers::Towers(GameData *gameDataValue){
    gameData = gameDataValue;

    name[0] = "Small Wall";
    image[0] = gameData->textureManager.smallWall;
    cost[0] = 50;
    amount[0] = 0;

    name[1] = "Big Wall";
    image[1] = gameData->textureManager.bigWall;
    cost[1] = 100;
    amount[1] = 0;

    name[2] = "Gun Tower 1";
    image[2] = gameData->textureManager.gunTower1;
    cost[2] = 200;
    amount[2] = 0;

    name[3] = "Gun Tower 2";
    image[3] = gameData->textureManager.gunTower2;
```

```

        cost[3] = 300;
        amount[3] = 0;
    }

Towers::~Towers() {

}

void Towers::set_images() {

    image[0] = gameData->textureManager.smallWall;
    image[1] = gameData->textureManager.bigWall;
    image[2] = gameData->textureManager.gunTower1;
    image[3] = gameData->textureManager.gunTower2;

}

```

A.66 Файл UnitBullet.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class UnitBullet : public GameObject{

public:

    // Graphics
    SDL_Rect bulletRect = {0, 0, 20, 20};

    // Common
    float force = 2.0f;
    bool dead = false;
    int damage = 1;

    // Physics
    b2FixtureDef fixtureDef;
    b2Body* body;

    UnitBullet(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, Transform
startPosition, float angleValue, int damageValue);
    ~UnitBullet() override;

    void update() override;
    void render() override;

    void set_sprites();
    void set_position();

```

```
};
```

A.67 Файл UnitBullet.cpp

```
#include "UnitBullet.h"
#include "GameData.h"

UnitBullet::UnitBullet(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, Transform
startPosition, float angleValue, int damageValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

    transform = startPosition;
    damage = damageValue;

    // Physics
    b2CircleShape shape;
    shape.m_p.Set(0.0f, 0.0f);
    shape.m_radius = 0.1f;

    fixtureDef.shape = &shape;
    fixtureDef.density = 0.1;
    fixtureDef.filter.categoryBits = UNIT_BULLET;
    fixtureDef.filter.maskBits = ENEMIES | OTHER;

    b2BodyDef bodyDef;
    bodyDef.position.Set(transform.position.x * gameData-
>PIXEL_TO_METER, transform.position.y * gameData->PIXEL_TO_METER);
    bodyDef.type = b2_dynamicBody;
    bodyDef.bullet = true;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&fixtureDef);

    b2Vec2 vec1(glm::cos(angleValue), glm::sin(angleValue));
    vec1.Normalize();
    body->ApplyForceToCenter(vec1, true);

    set_position();
    set_sprites();
}

UnitBullet::~UnitBullet() {

    gameData->world.DestroyBody(body);
}

void UnitBullet::update() {

    set_position();
```

```

        set_sprites();
    }
    void UnitBullet::render() {
        SDL_RenderCopyEx((*gameData).renderer, gameData->textureManager.bullet1, NULL, &bulletRect, glm::degrees(body->GetAngle()) + 90, NULL, SDL_FLIP_NONE);
    }

    void UnitBullet::set_sprites() {
        bulletRect.x = transform.position.x - bulletRect.w / 2.0;
        bulletRect.y = transform.position.y - bulletRect.h / 2.0;
    }

    void UnitBullet::set_position() {
        transform.position.x = body->GetPosition().x * gameData->METER_TO_PIXEL;
        transform.position.y = body->GetPosition().y * gameData->METER_TO_PIXEL;
    }
}

```

A.68 Файл UnitBulletManager.h

```

#pragma once

#include "GameObject.h"

class UnitBulletManager : public GameObject{
public:
    UnitBulletManager(std::string nameValue, EventState objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~UnitBulletManager() override;

    void events() override;
    void update() override;
};

```

A.69 Файл UnitBulletManager.cpp

```

#include "UnitBulletManager.h"
#include "UnitBullet.h"
#include "GameData.h"

UnitBulletManager::UnitBulletManager(std::string nameValue, EventState objectEventValue, GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

```

```

}
UnitBulletManager::~UnitBulletManager() {

}

void UnitBulletManager::update() {

    int j = children.size() - 1;
    while(j >= 0){
        if(((UnitBullet *)children[j])->dead == true){
            //children[j]->destroy();
            delete_child(j);
        }
        --j;
    }
}

void UnitBulletManager::events() {

    // Check current event and decide what we can do with it
    switch(gameData->event.type) {

        // Test
        case SDLK_KEYDOWN:
            switch (gameData->event.key.keysym.sym)
            {
                case SDLK_LEFT:
                    //children[0]->destroy();
                    //delete_child(0);
                    break;
                case SDLK_RIGHT:
                    break;
            }
            break;
    }
}

```

A.70 Файл Wall.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

```

```

class Wall : public GameObject{
public:
    // Graphics
    SDL_Rect wallRect;

    // Physics
    b2FixtureDef fixtureDef;
    b2Body* body;

    Wall(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue, float x, float y, float w,
float h);
    ~Wall() override;

    void render() override;
};

```

A.71 Файл Wall.cpp

```

#include "Wall.h"
#include "GameData.h"

Wall::Wall(std::string nameValue, EventState objectEventValue, GameData
*gameDataValue, GameObject *parentValue, float x, float y, float w, float h)
: GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    b2PolygonShape shape;
    shape.SetAsBox(w / 2.0f, h / 2.0f);

    fixtureDef.shape = &shape;
    fixtureDef.density = 0.1;
    fixtureDef.filter.categoryBits = OTHER;
    fixtureDef.filter.maskBits = UNITS | ENEMIES | UNIT_BULLET;

    b2BodyDef bodyDef;
    bodyDef.position.Set(x, y);
    bodyDef.type = b2_staticBody;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&fixtureDef);

    wallRect.w = w * (*gameData).METER_TO_PIXEL;
    wallRect.h = h * (*gameData).METER_TO_PIXEL;
    wallRect.x = (body->GetPosition()).x * (*gameData).METER_TO_PIXEL)
- wallRect.w / 2.0f;
    wallRect.y = (body->GetPosition()).y * (*gameData).METER_TO_PIXEL)
- wallRect.h / 2.0f;

    printf("Wall is created\n");
}

Wall::~~Wall(){

```

```

}

void Wall::render() {
    SDL_SetRenderDrawColor((*gameData).renderer, 50, 50, 50, 255);
    SDL_RenderFillRect((*gameData).renderer, &wallRect);
}

```

A.72 Файл Base.h

```

#pragma once

#include "Unit.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class Base : public Unit{
public:
    // Graphics
    SDL_Rect baseRect;

    // Physics
    b2FixtureDef fixtureDef;
    b2Body *body;

    Base(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~Base() override;

    void render() override;
};

```

A.73 Файл Base.cpp

```

#include "Base.h"
#include "GameData.h"

Base::Base(std::string nameValue, EventState objectEventValue, GameData
*gameDataValue, GameObject *parentValue) : Unit(nameValue, objectEventValue,
gameDataValue, parentValue){

    // Physics
    b2CircleShape shape;

```

```

        shape.m_p.Set(0.0f, 0.0f);
        shape.m_radius = 0.5f;

        fixtureDef.density = 0.1;
        fixtureDef.shape = &shape;
        fixtureDef.filter.categoryBits = UNITS;
        fixtureDef.filter.maskBits = UNITS | ENEMIES | OTHER;

        b2BodyDef bodyDef;
        bodyDef.position.Set(2.0f, 4.0f);
        bodyDef.type = b2_staticBody;
        bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

        body = (*gameData).world.CreateBody(&bodyDef);
        body->CreateFixture(&fixtureDef);

        transform.position.x = body->GetPosition().x * gameData-
>METER_TO_PIXEL;
        transform.position.y = body->GetPosition().y * gameData-
>METER_TO_PIXEL;

        health = 10;

        baseRect.w = 100;
        baseRect.h = 100;
        baseRect.x = transform.position.x - baseRect.w / 2.0;
        baseRect.y = transform.position.y - baseRect.h / 2.0;

        printf("Base is created\n");
    }

    Base::~Base() {

        gameData->world.DestroyBody(body);

    }

    void Base::render() {

        SDL_RenderCopy((*gameData).renderer, gameData-
>textureManager.base, NULL, &baseRect);

    }

```

A.74 Файл BigWall.h

```

#pragma once

#include "Unit.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class BigWall : public Unit{

```

```

public:

    // Graphics
    SDL_Rect imageRect = {0, 0, 100, 100};

    // Physics
    b2FixtureDef fixtureDef;
    b2Body* body;

    BigWall(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue, bool isBuildingValue);
    ~BigWall() override;

    void events() override;
    void render() override;

    void set_sprites();
    void set_position();
    void create_body(float x, float y, bool isSensorValue);
    void destroy_body();

};

```

A.75 Файл BigWall.cpp

```

#include "BigWall.h"
#include "GameData.h"

BigWall::BigWall(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue, bool isBuildingValue) :
Unit(nameValue, objectEventValue, gameDataValue, parentValue){

    health = 5;
    isBuilding = isBuildingValue;

    create_body(0, 0, true);
}

BigWall::~BigWall(){

    destroy_body();
}

void BigWall::render(){

    SDL_RenderCopy((*gameData).renderers, gameData->textureManager.bigWall, NULL, &imageRect);

}

```

```

void BigWall::set_sprites() {
    imageRect.x = transform.position.x - imageRect.w / 2.0;
    imageRect.y = transform.position.y - imageRect.h / 2.0;
}
void BigWall::set_position() {
    transform.position.x = body->GetPosition().x * gameData->METER_TO_PIXEL;
    transform.position.y = body->GetPosition().y * gameData->METER_TO_PIXEL;
}

void BigWall::create_body(float x, float y, bool isSensorValue) {
    b2PolygonShape shape;
    shape.SetAsBox(0.5, 0.5);

    fixtureDef.density = 0.1;
    fixtureDef.shape = &shape;
    fixtureDef.filter.categoryBits = UNITS;
    fixtureDef.filter.maskBits = UNITS | ENEMIES | OTHER;
    fixtureDef.isSensor = isSensorValue;

    b2BodyDef bodyDef;
    bodyDef.position.Set(x, y);
    bodyDef.type = b2_staticBody;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&fixtureDef);

    set_position();
    set_sprites();
}

void BigWall::destroy_body() {
    gameData->world.DestroyBody(body);
}

void BigWall::events() {
    // Button clicking
    if ((*gameData).currentEvent == objectEvent) {
        switch ((*gameData).event.type) {

            SDL_Point mousePoint;

            case SDL_MOUSEMOTION:

                mousePoint.x = (*gameData).event.motion.x;
                mousePoint.y = (*gameData).event.motion.y;
                if (isBuilding == true) {
                    body->SetTransform(b2Vec2(mousePoint.x *
gameData->PIXEL_TO_METER, mousePoint.y * gameData->PIXEL_TO_METER), 0);
                }
            }
        }
    }
}

```



```

        float aimAngle = 0;
        bool fire = false;
        bool fire_delay = false;

        int damage = 1;
        Uint32 timeDelay = 1;

        // Timer
        SDL_TimerID timerID1;

        Gun(std::string nameValue, EventState objectEventValue,
        GameData *gameDataValue, GameObject *parentValue, SDL_Texture *imageValue);
        ~Gun() override;

        void update() override;
        void render() override;

        void set_sprites();
        void set_position();
        void create_body(float x, float y);
        void destroy_body();
        static Uint32 delay_callback(Uint32 interval, void* param);
};

```

A.77 Файл Gun.cpp

```

#include "Gun.h"
#include "GameData.h"
#include "UnitBullet.h"

Gun::Gun(std::string nameValue, EventState objectEventValue, GameData
*gameDataValue, GameObject *parentValue, SDL_Texture *imageValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    image = imageValue;
    create_body(0, 0);
}

Gun::~~Gun() {

    SDL_RemoveTimer(timerID1);
    destroy_body();
}

void Gun::update() {

    if(target != NULL){
        aimAngle = glm::atan(target->position.y -
transform.position.y, target->position.x - transform.position.x);
    }

    if(fire == true && fire_delay == false){

        fire_delay = true;
    }
}

```

```

        parent->parent->parent->children[2]->add_child(new
UnitBullet("Bullet",    TOWER_DEFENSE,    gameData,    parent->parent->parent-
>children[2], transform, aimAngle, damage));
        timerID1      =      SDL_AddTimer(timeDelay,      delay_callback,
&fire_delay);

    }

}

void Gun::render() {

    SDL_Point headPoint;
    headPoint.x = headRect.w / 2;
    headPoint.y = headRect.h / 2 + 10;
    SDL_RenderCopyEx((*gameData).renderer, image, NULL, &headRect,
glm::degrees(aimAngle) + 90, &headPoint, SDL_FLIP_NONE);

}

void Gun::set_sprites() {

    headRect.x = transform.position.x - headRect.w / 2.0;
    headRect.y = transform.position.y - headRect.h / 2.0 - 10;

}

void Gun::set_position() {

    transform.position.x    =    body->GetPosition().x    *    gameData-
>METER_TO_PIXEL;
    transform.position.y    =    body->GetPosition().y    *    gameData-
>METER_TO_PIXEL;

}

void Gun::create_body(float x, float y){

    b2CircleShape shape;
    shape.m_p.Set(0.0f, 0.0f);
    shape.m_radius = 2.0f;

    sensorFixture.density = 0.1;
    sensorFixture.shape = &shape;
    sensorFixture.filter.categoryBits = UNITS;
    sensorFixture.filter.maskBits = ENEMIES;
    sensorFixture.isSensor = true;

    b2BodyDef bodyDef;
    bodyDef.position.Set(x, y);
    bodyDef.type = b2_staticBody;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&sensorFixture);

    set_position();
    set_sprites();

}

void Gun::destroy_body() {

    gameData->world.DestroyBody(body);

```

```

}

Uint32 Gun::delay_callback(Uint32 interval, void* param){

    (*(bool *)param) = false;
    return 0;
}

```

A.78 Файл GunTower1.h

```

#pragma once

#include "Unit.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class GunTower1 : public Unit{

public:

    // Graphics
    SDL_Rect bodyRect = {0, 0, 50, 50};

    // Physics
    b2FixtureDef fixtureDef;
    b2Body* body;

    GunTower1(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, bool
isBuildingValue);

    ~GunTower1() override;

    void events() override;
    void update() override;
    void render() override;

    void set_sprites();
    void set_position();
    void create_body(float x, float y, bool isSensorValue);
    void destroy_body();

};

```

A.79 Файл GunTower1.cpp

```

#include "GunTower1.h"
#include "GameData.h"
#include "Gun.h"

GunTower1::GunTower1(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue, bool

```

```

isBuildingValue)      :      Unit(nameValue,      objectEventValue,      gameDataValue,
parentValue){

    health = 5;
    isBuilding = isBuildingValue;

    create_body(0, 0, true);

    add_child(new Gun("Gun", TOWER_DEFENSE, gameData, this, gameData-
>textureManager.gunTowerHead1));
    ((Gun *)children[0])->damage = 2;
    ((Gun *)children[0])->timeDelay = 500;
}

GunTower1::~~GunTower1(){

    destroy_body();
}

void GunTower1::update(){

}

void GunTower1::render(){

    SDL_RenderCopyEx((*gameData).renderer,      gameData-
>textureManager.gunTowerBody1,      NULL,      &bodyRect,      glm::degrees(body-
>GetAngle()) + 90, NULL, SDL_FLIP_NONE);

}

void GunTower1::set_sprites(){

    bodyRect.x = transform.position.x - bodyRect.w / 2.0;
    bodyRect.y = transform.position.y - bodyRect.h / 2.0;

}

void GunTower1::set_position(){

    transform.position.x      =      body->GetPosition().x      *      gameData-
>METER_TO_PIXEL;
    transform.position.y      =      body->GetPosition().y      *      gameData-
>METER_TO_PIXEL;

}

void GunTower1::create_body(float x, float y, bool isSensorValue){

    b2CircleShape shape;
    shape.m_p.Set(0.0f, 0.0f);
    shape.m_radius = 0.25f;

```

```

        fixtureDef.density = 0.1;
        fixtureDef.shape = &shape;
        fixtureDef.filter.categoryBits = UNITS;
        fixtureDef.filter.maskBits = UNITS | ENEMIES | OTHER;
        fixtureDef.isSensor = isSensorValue;

        b2BodyDef bodyDef;
        bodyDef.position.Set(x, y);
        bodyDef.type = b2_staticBody;
        bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

        body = (*gameData).world.CreateBody(&bodyDef);
        body->CreateFixture(&fixtureDef);

        set_position();
        set_sprites();
    }

    void GunTower1::destroy_body() {
        gameData->world.DestroyBody(body);
    }

    void GunTower1::events() {
        // Button clicking
        if((*gameData).currentEvent == objectEvent){
            switch((*gameData).event.type){

                SDL_Point mousePoint;

                case SDL_MOUSEMOTION:

                    mousePoint.x = (*gameData).event.motion.x;
                    mousePoint.y = (*gameData).event.motion.y;
                    if(isBuilding == true){

                        body->SetTransform(b2Vec2(mousePoint.x *
gameData->PIXEL_TO_METER, mousePoint.y * gameData->PIXEL_TO_METER), 0);
                        set_position();
                        set_sprites();

                        ((Gun *)children[0])->body->
>SetTransform(b2Vec2(body->GetPosition().x, body->GetPosition().y), 0);
                        ((Gun *)children[0])->set_position();
                        ((Gun *)children[0])->set_sprites();

                    }

                    break;
                case SDL_MOUSEBUTTONDOWN:
                    SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
                    if((*gameData).event.button.button ==
SDL_BUTTON_RIGHT){

                        click = true;

                    }
            }
        }
    }

```

```

        break;
    case SDL_MOUSEBUTTONDOWN:
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        if(click == true && isBuilding == true &&
(*gameData).event.button.button == SDL_BUTTON_RIGHT){

            isBuilding = false;
            destroy_body();
            create_body(mousePoint.x * gameData->PIXEL_TO_METER,
mousePoint.y * gameData->PIXEL_TO_METER, false);
            //((Gun *)children[0])->create_body(mousePoint.x * gameData->PIXEL_TO_METER,
mousePoint.y * gameData->PIXEL_TO_METER);
        }
        click = false;

        break;
    }
}
}
}

```

A.80 Файл Player.h

```

#pragma once

#include "Unit.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class Player : public Unit{
public:

    // Graphics
    SDL_Rect bodyRect;
    SDL_Rect headRect;

    // Aiming
    bool aim;
    float aimAngle;

    // Fire
    bool fire;
    bool fire_delay;

    // Data
    float moveSpeed = 200;
    float rotationSpeed = 200;

    // Physics
    b2FixtureDef fixtureDef;
    b2Body* body;

```

```

        // Timer
        SDL_TimerID timerID1;

        Player(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
        ~Player() override;

        void events() override;
        void fixed_update() override;
        void update() override;
        void render() override;

        void set_sprites();
        void set_position();
        static Uint32 delay_callback(Uint32 interval, void* param);
};

```

A.81 Файл Player.cpp

```

#include "Player.h"
#include "GameData.h"
#include "UnitBullet.h"

Player::Player(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : Unit(nameValue,
objectEventValue, gameDataValue, parentValue){

    b2PolygonShape shape;
    shape.SetAsBox(0.25f, 0.25f);

    fixtureDef.shape = &shape;
    fixtureDef.density = 0.1;
    fixtureDef.filter.categoryBits = UNITS;
    fixtureDef.filter.maskBits = UNITS | ENEMIES | OTHER;

    b2BodyDef bodyDef;
    bodyDef.position.Set(2.0f, 2.0f);
    bodyDef.type = b2_dynamicBody;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&fixtureDef);

    aim = false;
    aimAngle = 0.0f;
    health = 5;
    fire = false;
    fire_delay = false;

    bodyRect.w = 50;
    bodyRect.h = 50;
    headRect.w = 50;
    headRect.h = 50;

```



```

        }
        break;

        case SDL_MOUSEBUTTONDOWN:

            if ((*gameData).event.button.button ==
SDL_BUTTON_RIGHT) {
                aim = true;
            }
            if ((*gameData).event.button.button ==
SDL_BUTTON_LEFT) {
                fire = true;
            }

            break;

        case SDL_MOUSEBUTTONUP:

            if ((*gameData).event.button.button ==
SDL_BUTTON_RIGHT) {
                aim = false;
            }
            if ((*gameData).event.button.button ==
SDL_BUTTON_LEFT) {
                fire = false;
            }

            break;
    }
}
}
}

```

```

void Player::fixed_update() {

    b2Vec2 vel;
    vel.x = transform.normalize().x * moveSpeed *
(*gameData).fixedDeltaTime;
    vel.y = transform.normalize().y * moveSpeed *
(*gameData).fixedDeltaTime;

    transform.rotation = body->GetAngle();

    // Current rotation
    glm::quat currentRotation = glm::angleAxis(transform.rotation,
glm::vec3(0, 0, 1));
    // Target rotation
    glm::quat targetRotation = glm::angleAxis(transform.rotation,
glm::vec3(0, 0, 1));

    if (vel.x != 0 || vel.y != 0) {
        targetRotation = glm::angleAxis(glm::atan(vel.y, vel.x),
glm::vec3(0, 0, 1));
    }
}

```

```

        float    angleDiff    =    glm::eulerAngles(targetRotation).z    -
glm::eulerAngles(currentRotation).z;
        while(angleDiff < -glm::pi<float>()){
            angleDiff += 2 * glm::pi<float>();
        }
        while(angleDiff > glm::pi<float>()){
            angleDiff -= 2 * glm::pi<float>();
        }

        float    angVel    =    angleDiff    *    rotationSpeed    *
(*gameData).fixedDeltaTime;

        body->SetLinearVelocity(vel);
        body->SetAngularVelocity(angVel);
    }

void Player::update() {

    set_position();
    set_sprites();

    if(aim == true){
        SDL_Point mousePoint;
        SDL_GetMouseState(&mousePoint.x, &mousePoint.y);
        aimAngle = glm::atan(mousePoint.y - transform.position.y,
mousePoint.x - transform.position.x);
    }
    else{
        aimAngle = body->GetAngle();
    }
    if(fire == true && fire_delay == false){
        fire_delay = true;
        parent->parent->children[2]->add_child(new
UnitBullet("Bullet", TOWER_DEFENSE, gameData, parent->parent->children[2],
transform, aimAngle, 1));
        timerID1 = SDL_AddTimer(500, delay_callback, &fire_delay);
    }
}

void Player::render() {

    SDL_Point headPoint;
    headPoint.x = headRect.w / 2;
    headPoint.y = headRect.h / 2 + 10;

    SDL_RenderCopyEx((*gameData).renderer, gameData-
>textureManager.bodySprite1, NULL, &bodyRect, glm::degrees(body->GetAngle())
+ 90, NULL, SDL_FLIP_NONE);
    SDL_RenderCopyEx((*gameData).renderer, gameData-
>textureManager.headSprite1, NULL, &headRect, glm::degrees(aimAngle) + 90,
&headPoint, SDL_FLIP_NONE);
}

void Player::set_sprites() {

```

```

        bodyRect.x = transform.position.x - bodyRect.w / 2.0;
        bodyRect.y = transform.position.y - bodyRect.h / 2.0;

        headRect.x = transform.position.x - headRect.w / 2.0;
        headRect.y = transform.position.y - headRect.h / 2.0 - 10;
    }

    void Player::set_position(){
        transform.position.x = body->GetPosition().x * gameData-
>METER_TO_PIXEL;
        transform.position.y = body->GetPosition().y * gameData-
>METER_TO_PIXEL;
    }

    Uint32 Player::delay_callback(Uint32 interval, void* param){
        (*(bool *)param) = false;
        return 0;
    }

```

A.82 Файл Unit.h

```

#pragma once

#include "GameObject.h"

class Unit : public GameObject{
public:
    // Common
    float health = 1;
    bool dead = false;

    bool isBuilding = false;
    bool click = false;

    Unit(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~Unit() override;
};

```

A.83 Файл Unit.cpp

```

#include "Unit.h"

Unit::Unit(std::string nameValue, EventState objectEventValue, GameData
*gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

```

```

}

Unit::~Unit() {

}

```

A.84 Файл UnitManager.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>

class UnitManager : public GameObject{
public:

    UnitManager(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~UnitManager() override;

    void events() override;
    void update() override;

};

```

A.85 Файл UnitManager.cpp

```

#include "UnitManager.h"
#include "TowerDefense.h"
#include "GameData.h"
#include "Base.h"
#include "Player.h"
#include "SmallWall.h"

UnitManager::UnitManager(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){

    // Base
    add_child(new Base("Unit Base", TOWER_DEFENSE, gameData, this));

    // Player
    add_child(new Player("Unit Player", TOWER_DEFENSE, gameData,
this));

    // Test wall
    //add_child(new SmallWall("Unit Small wall", TOWER_DEFENSE,
gameData, this, false));
    //((SmallWall *)children[2])>create_body(5, 4, false);
}

```

```

}
UnitManager::~UnitManager() {

}

void UnitManager::update() {

    if(((Base *)children[0])->dead == true){
        ((TowerDefense *)parent)->finish_game();
    }

    int j = children.size() - 1;
    while(j >= 0){
        if(((Unit *)children[j])->dead == true){
            delete_child(j);
        }
        --j;
    }
}

void UnitManager::events() {

    // Check current event and decide what we can do with it
    switch(gameData->event.type){

        // Test
        case SDL_KEYDOWN:
            switch (gameData->event.key.keysym.sym)
            {
                case SDLK_LEFT:
                    //gameData-
>world.DestroyBody(((SmallWall *)children[2])->body);
                    //((SmallWall *)children[2]) -
>fixtureDef.isSensor = true;
                    //((SmallWall *)children[2]) -
>create_body(5, 4);
                    break;
                case SDLK_RIGHT:
                    //gameData-
>world.DestroyBody(((SmallWall *)children[2])->body);
                    //((SmallWall *)children[2]) -
>fixtureDef.isSensor = false;
                    //((SmallWall *)children[2]) -
>create_body(5, 4);
                    break;
            }
            break;
    }
}

```

A.86 Файл Enemy.h

```

#pragma once

#include "GameObject.h"

```

```

class Enemy : public GameObject{
public:
    // Common
    float health = 1;
    int damage = 1;
    int money = 1;
    bool dead = false;

    Enemy(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue);
    ~Enemy() override;
};

```

A.87 Файл Enemy.cpp

```

#include "Enemy.h"

Enemy::Enemy(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue) : GameObject(nameValue,
objectEventValue, gameDataValue, parentValue){

}

Enemy::~~Enemy(){

}

```

A.88 Файл Enemy1.h

```

#pragma once

#include "Enemy.h"
#include <SDL2/SDL.h>
#include <box2d/box2d.h>

class Enemy1 : public Enemy{
public:
    // Graphics
    SDL_Rect bodyRect = {0, 0, 40, 40};

    // Common
    float moveSpeed = 50;
    float rotationSpeed = 200;

    Transform *target = NULL;

```

```

        // Physics
        b2FixtureDef fixtureDef;
        b2Body* body;

        Enemy1(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue, float xValue, float yValue,
Transform *targetValue);
        ~Enemy1() override;

        void fixed_update() override;
        void update() override;
        void render() override;

        void set_sprites();
        void set_position();

};

```

A.89 Файл Enemy1.cpp

```

#include "Enemy1.h"
#include "GameData.h"

Enemy1::Enemy1(std::string nameValue, EventState objectEventValue,
GameData *gameDataValue, GameObject *parentValue, float xValue, float yValue,
Transform *targetValue) : Enemy(nameValue, objectEventValue, gameDataValue,
parentValue){

    // Common
    target = targetValue;

    // Physics
    b2PolygonShape shape;
    shape.SetAsBox(0.2f, 0.2f);

    fixtureDef.shape = &shape;
    fixtureDef.density = 0.1;
    fixtureDef.filter.categoryBits = ENEMIES;
    fixtureDef.filter.maskBits = UNITS | ENEMIES | UNIT_BULLET |

OTHER;

    b2BodyDef bodyDef;
    bodyDef.position.Set(xValue, yValue);
    bodyDef.type = b2_dynamicBody;
    bodyDef.userData.pointer = reinterpret_cast<uintptr_t>(this);

    body = (*gameData).world.CreateBody(&bodyDef);
    body->CreateFixture(&fixtureDef);

    set_position();
    set_sprites();
}

```

```

        health = 3;
        damage = 1;
        money = 50;
    }
    Enemy1::~Enemy1() {
        gameData->world.DestroyBody(body);
    }

    void Enemy1::set_sprites() {
        bodyRect.x = transform.position.x - bodyRect.w / 2.0;
        bodyRect.y = transform.position.y - bodyRect.h / 2.0;
    }

    void Enemy1::set_position() {
        transform.position.x = body->GetPosition().x * gameData->METER_TO_PIXEL;
        transform.position.y = body->GetPosition().y * gameData->METER_TO_PIXEL;
    }

    void Enemy1::fixed_update() {
        transform.velocity.x = target->position.x - transform.position.x;
        transform.velocity.y = target->position.y - transform.position.y;

        b2Vec2 vel;
        vel.x = transform.normalize().x * moveSpeed * (*gameData).fixedDeltaTime;
        vel.y = transform.normalize().y * moveSpeed * (*gameData).fixedDeltaTime;

        transform.rotation = body->GetAngle();

        // Current rotation
        glm::quat currentRotation = glm::angleAxis(transform.rotation, glm::vec3(0, 0, 1));
        // Target rotation
        glm::quat targetRotation = glm::angleAxis(transform.rotation, glm::vec3(0, 0, 1));

        if(vel.x != 0 || vel.y != 0){
            targetRotation = glm::angleAxis(glm::atan(vel.y, vel.x), glm::vec3(0, 0, 1));
        }

        float angleDiff = glm::eulerAngles(targetRotation).z - glm::eulerAngles(currentRotation).z;
        while(angleDiff < -glm::pi<float>()){
            angleDiff += 2 * glm::pi<float>();
        }
        while(angleDiff > glm::pi<float>()){
            angleDiff -= 2 * glm::pi<float>();
        }
    }

```

```

    }

    float    angVel    =    angleDiff    *    rotationSpeed    *
(*gameData).fixedDeltaTime;

    body->SetLinearVelocity(vel);
    body->SetAngularVelocity(angVel);
}
void Enemy1::update() {

    set_position();
    set_sprites();

}
void Enemy1::render() {

    SDL_RenderCopyEx((*gameData).renderer,                gameData-
>textureManager.enemy1, NULL, &bodyRect, glm::degrees(body->GetAngle()) + 90,
NULL, SDL_FLIP_NONE);

}

```

A.90 Файл EnemyManager.h

```

#pragma once

#include "GameObject.h"
#include <SDL2/SDL.h>
// #include "EnemyData.h"

class EnemyManager : public GameObject{

public:

    // Graphics
    SDL_Rect wallRect = {0, 0, 0, 0};

    EnemyManager(std::string    nameValue,                EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue);
    ~EnemyManager() override;

    void events() override;
    void update() override;
    void render() override;

    void set_size();
    void spawn_enemies(int dayValue);

};

```

A.91 Файл EnemyManager.cpp

```
#include "EnemyManager.h"
#include "GameData.h"
#include "Enemy1.h"
#include "Enemy2.h"
#include "Enemy3.h"
#include <glm/glm.hpp>
#include <glm/gtc/random.hpp>
#include <time.h>
```

```
EnemyManager::EnemyManager(std::string nameValue, EventState
objectEventValue, GameData *gameDataValue, GameObject *parentValue) :
GameObject(nameValue, objectEventValue, gameDataValue, parentValue){
```

```
    set_size();
```

```
}
EnemyManager::~EnemyManager() {
```

```
}
```

```
void EnemyManager::events() {
```

```
    // Resizing event
    switch((*gameData).event.type){
        case SDL_WINDOWEVENT:
            switch((*gameData).event.window.event){
                case SDL_WINDOWEVENT_RESIZED:
                    set_size();
                    break;
            }
            break;
    }
```

```
}
```

```
void EnemyManager::update() {
```

```
    int j = children.size() - 1;
    while(j >= 0){
        if(((Enemy *)children[j])->dead == true){

            delete_child(j);
            if(children.size() == 0){
                (*gameData).dayActive = false;
                ++gameData->userData.dayCounter;
            }

            }
        --j;
    }
```

```

    }

    void EnemyManager::render() {

        SDL_SetRenderDrawColor(gameData->renderer, 255, 0, 0, 200);
        SDL_RenderFillRect(gameData->renderer, &wallRect);

    }

    void EnemyManager::set_size() {

        wallRect.h = gameData->screenHeight;
        wallRect.w = 0.5f * gameData->screenWidth;
        wallRect.x = 0.8f * gameData->screenWidth;
        wallRect.y = 0;

    }

    void EnemyManager::spawn_enemies(int dayValue) {

        float beginX = 12;
        float endX = 14;
        float beginY = 1;
        float endY = 7;

        switch(dayValue) {
            case 1:

                for(int i = 0; i < 3; ++i)
                    add_child(new Enemy1("Enemy 1", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));

                break;

            case 2:

                for(int i = 0; i < 6; ++i)
                    add_child(new Enemy1("Enemy 1", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));
                    add_child(new Enemy2("Enemy 2", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));

                break;

            case 3:

                for(int i = 0; i < 9; ++i)
                    add_child(new Enemy1("Enemy 1", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));
                    for(int i = 0; i < 3; ++i)
                        add_child(new Enemy2("Enemy 2", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));

                break;

            case 4:

                for(int i = 0; i < 9; ++i)

```



```

        add_child(new Enemy3("Enemy 3", TOWER_DEFENSE,
gameData, this, glm::linearRand(beginX, endX), glm::linearRand(beginY, endY),
&(parent->children[0]->children[0]->transform)));
        break;
    }
}

```

A.92 Файл Enumerations.h

```
#pragma once
```

```
enum EventState{
```

```
    MAIN_MENU,
```

```
    GAME_ROOM,
```

```
    SHOP_ROOM,
```

```
    TUTORIAL_ROOM,
```

```
    SETTINGS_ROOM,
```

```
    INFO_ROOM,
```

```
    EXIT_ROOM,
```

```
    TOWER_DEFENSE,
```

```
    TOWER_ROOM,
```

```
    NULL_EVENT
```

```
};
```

```
enum PhysicalLayer{
```

```
    UNITS = 0b0001,
```

```
    ENEMIES = 0b0010,
```

```
    UNIT_BULLET = 0b0100,
```

```
    OTHER = 0b1000
```

```
};
```

ДОДАТОК Б
Слайди презентації



Рисунок Б.1 – Слайд № 1

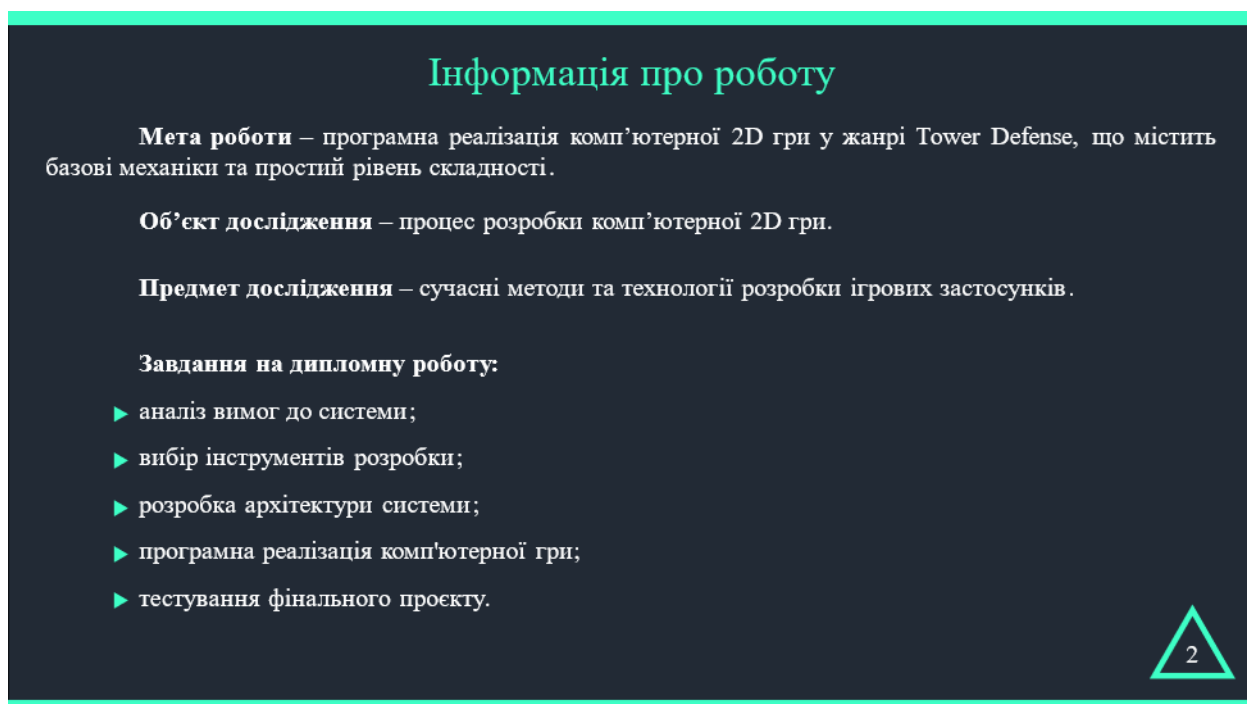


Рисунок Б.2 – Слайд № 2

Огляд існуючих аналогів

Характеристика / Назва гри	Mindustry	Bloons Tower Defense	Plants vs. Zombies	Thronefall
Геймплей	Стратегічне будівництво та захист фабрик	Різноманітність веж та ворогів, динамічні рівні	Унікальний геймплей з різними типами рослин та ворогів	Стратегічний геймплей з елементами фентезі
Графіка	Висока	Середня	Середня	Висока
Режим гри	Одиночна гра, мультиплеєр, режими викликів	Одиночна гра, мультиплеєр, спеціальні режими	Одиночна гра, пазли, міні-ігри	Одиночна гра, мультиплеєр, режими викликів
Системні вимоги	Середні	Середні	Низькі	Високі

3

Рисунок Б.3 – Слайд № 3

Огляд існуючих патернів

Характеристика / Назва патерну	Game Object	Game Object with Components	Entity Component System	Custom System
Складність реалізації	Середня	Низька	Висока	Висока
Складність інтеграції	Низька	Середня	Висока	Низька
Швидкість розробки	Середня	Висока	Середня	Низька
Ступінь дублювання коду	Середній	Низький	Низький	Високий
Навантаження на CPU	Середнє	Високе	Низьке	Низьке
Ефективність управління RAM	Середня	Низька	Висока	Висока
Тип проєкту	Indie	AA	AAA	AAA

4

Рисунок Б.4 – Слайд № 4

Вибір мови програмування

Характеристика / Назва	C++	C#	Python
Продуктивність	Дуже висока	Середня	Низька
Використання в геймдеві	Широко використовується для розробки AAA ігор	Популярний в геймдеві, особливо завдяки Unity	Використовується для створення простих ігор
Графічний інтерфейс	Прямий доступ до графічного движка	Має не погані можливості для створення графіки	Обмежені можливості
Контроль ресурсів	Має повний контроль над ресурсами	Має середній контроль над ресурсами	Має невеликий контроль над ресурсами
Інтеграція з іншими інструментами	Має велику кількість бібліотек та фреймворків	Підтримує широкий спектр інтеграції з різними інструментами	Існує мала кількість бібліотек та фреймворків
Комунітет та підтримка	Має велику та активну спільноту, зокрема в середовищі Unreal Engine	Широка спільнота розробників, зокрема в середовищі Unity	Має невелику підтримку у розробці ігор

5

Рисунок Б.5 – Слайд № 5

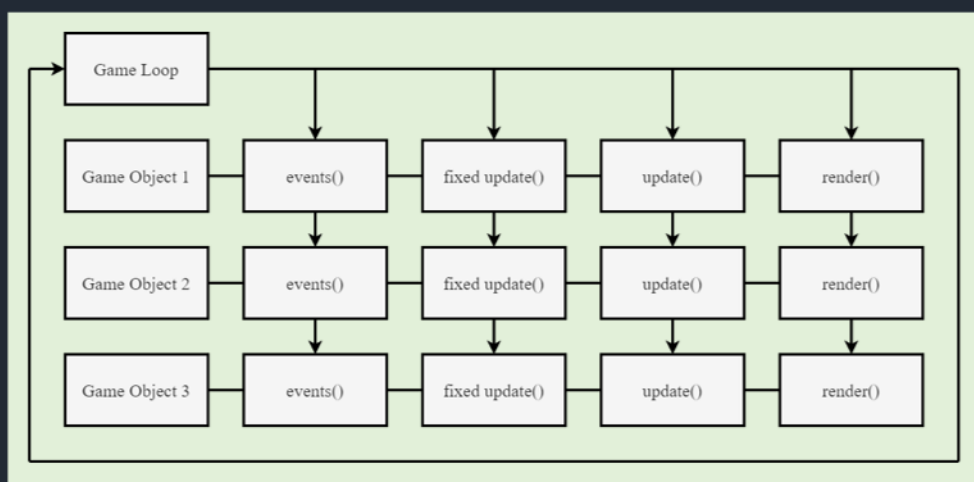
Вибір мультимедійної бібліотеки

Характеристика / Назва	SDL	SFML	Allegro
Кросплатформеність	Підтримує Windows, macOS, Linux, Android, iOS	Підтримує Windows, macOS, Linux, iOS, Android	Підтримує Windows, macOS, Linux, Android, iOS
Мови програмування	C, C++, та інші	C++, з підтримкою для C та інших мов	C, C++, та інші
Простота використання	Простий у використанні, але може знадобитися додатковий код для деяких функцій	Легкий у використанні та має простий API	Складний у використанні
Мультимедійні можливості	Надає доступ до аудіо, відео, графіки та інших операцій	Підтримує роботу з 2D графікою, аудіо та відео	Надає функції для роботи з графікою, аудіо та відео
Спільнота користувачів	Активна спільнота користувачів та широкі можливості підтримки	Має активну спільноту користувачів	Невелика спільнота користувачів
Розширення	Має підтримку великої кількості плагінів та розширень	Підтримує додаткові модулі та розширення, але може бути обмежена	Має певну кількість розширень, але може бути менш гнучкою
3D графіка	Має обмежену підтримку 3D графіки	Може бути інтегрована з OpenGL для 3D графіки	Має базову підтримку 3D графіки

6

Рисунок Б.6 – Слайд № 6

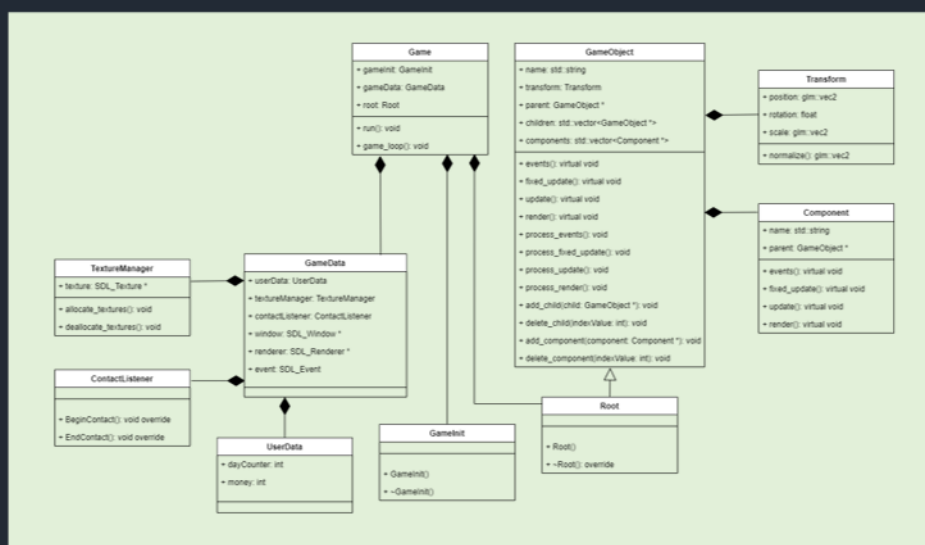
Принцип роботи ігрових об'єктів



7

Рисунок Б.7 – Слайд № 7

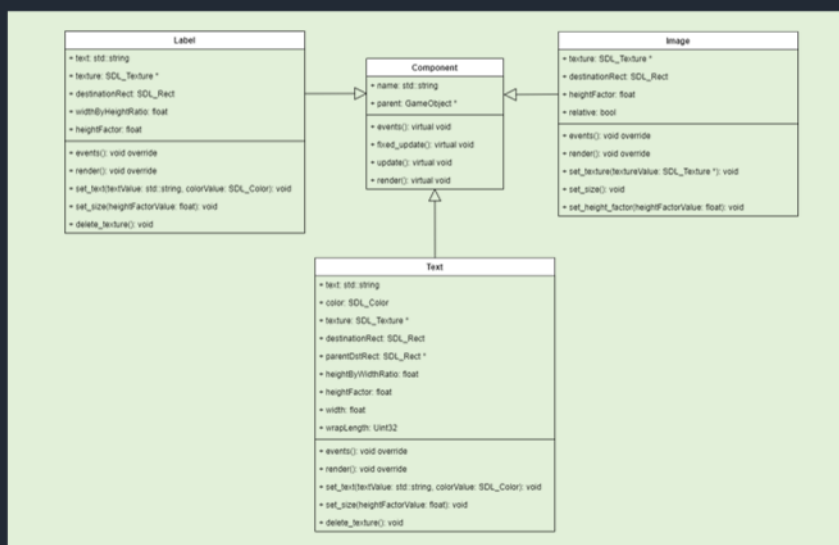
Основні класи ігрового процесу



8

Рисунок Б.8 – Слайд № 8

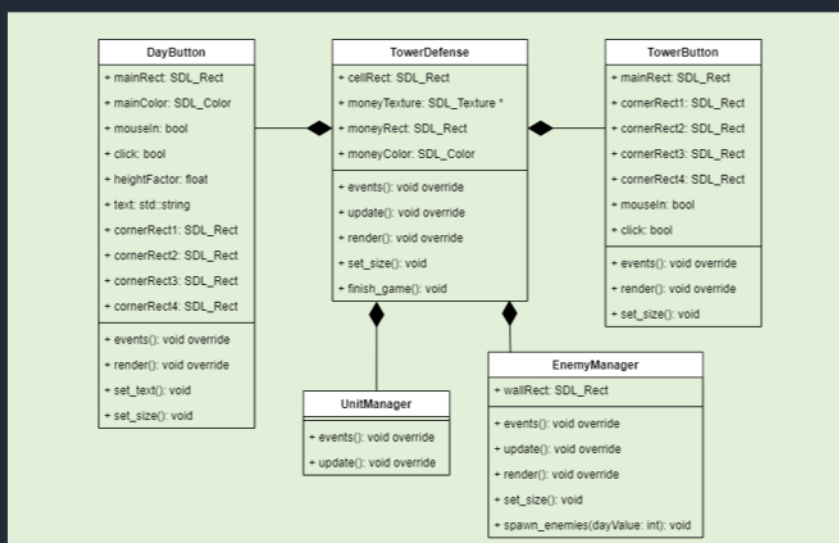
Основні компоненти



9

Рисунок Б.9 – Слайд № 9

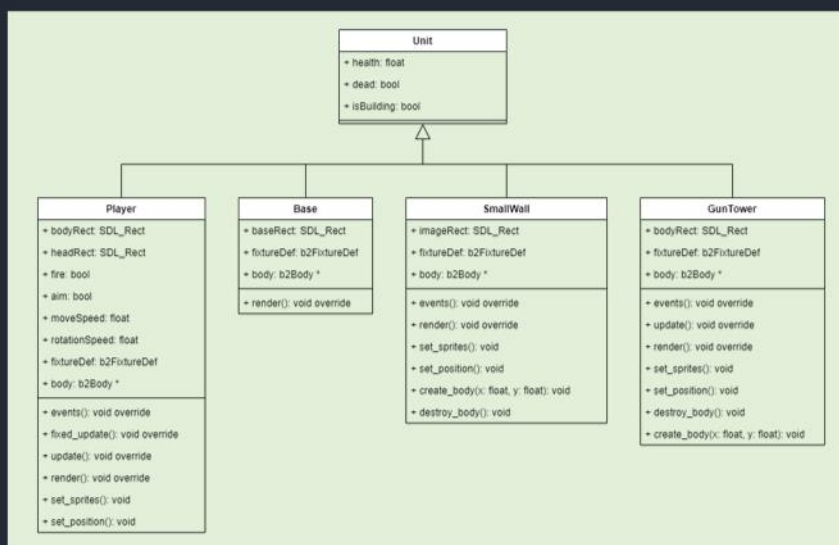
Головний режим гри



10

Рисунок Б.10 – Слайд № 10

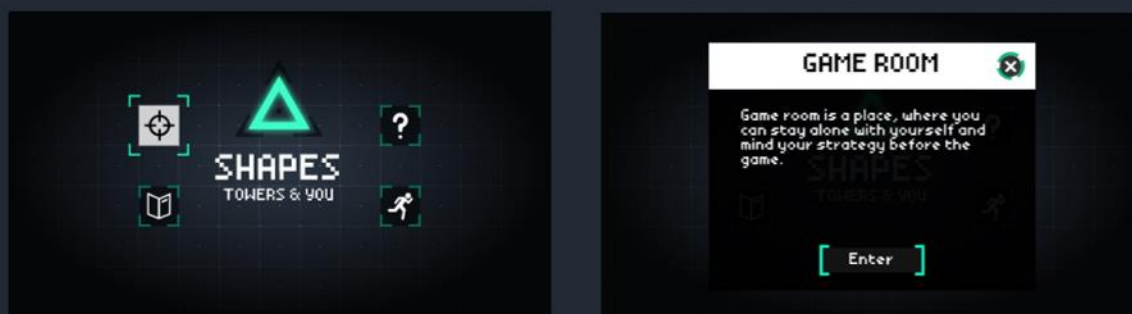
Союзні одиниці



11

Рисунок Б.11 – Слайд № 11

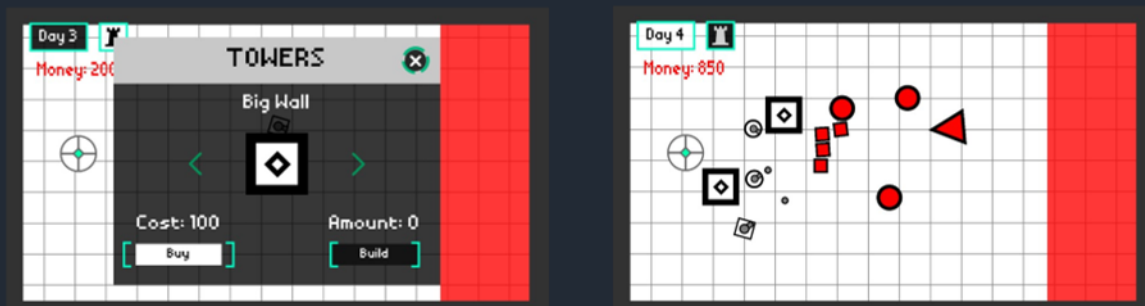
Інтерфейс комп'ютерної гри



12

Рисунок Б.12 – Слайд № 12

Інтерфейс комп'ютерної гри



13

Рисунок Б.13 – Слайд № 13

Висновок

В ході виконання кваліфікаційної роботи бакалавра було розроблено комп'ютерну гру, де гравець може керувати головним персонажем і будувати різні вежі, які захищатимуть базу гравця. Дана гра може бути опублікована та використана у розважальних сферах.

Розроблена гра має наступний функціонал:

- ▶ швидкий запуск гри;
- ▶ зручне головне меню;
- ▶ зручний та адаптований під різні монітори ігровий інтерфейс;
- ▶ управління головним персонажем, який здатен атакувати ворогів;
- ▶ захист основної бази гравця;
- ▶ будівництва захисних та атакуючих веж;
- ▶ хвилі ворогів;
- ▶ кінець гри, який відображає результати гравця.

14

Рисунок Б.14 – Слайд № 14