КВАЛІФІКАЦІЙНА РОБОТА

Група ІПЗс-2019
Мантуляк Д.В.

2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

на правах рукопису

Мантуляк Дмитро Володимирович

УДК 004.378

Розробка 2D аркадної гри з імплементацією рекламних вставок засобами Java та LibGDX

Спеціальність 121 – «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття кваліфікації бакалавра

Нормоконтроль	Студент
_____Стисло О.В. (підпис, дата, розшифрування підпису)	_____Мантуляк Д.В. (підпис, дата, розшифрування підпису)
Допускається до захисту Завідувач кафедри	Керівник роботи
_____к.т.н., доц. Пашкевич О.П. (підпис, дата, розшифрування підпису)	_____ к.т.н., доц. Бойчук А.М. (підпис, дата, розшифрування підпису)

Івано-Франківськ – 2023

ЗВО УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних та прикладних наук

Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»

Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

« ___ » _____ 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мантуляк Дмитро Володимирович

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи

Розробка 2D аркадної гри з імплементацією рекламних вставок засобами Java та LibGDX

керівник роботи:

Бойчук Андрій Михайлович, кандидат фізико-математичних наук

затверджена наказом вищого навчального закладу від « 11» листопада 2022 року

№ 155/1НВ

2. Термін подання студентом роботи 14.06.2023

3. Вихідні дані роботи: Мова програмування Java,LibGDX

4. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити)

1. Аналіз наявних аналогів мобільних ігор

2. Проектування структури ігри

3. Розробка структури ігри

4.Формування звіту

5. Дата видачі завдання 10.10.2022

КОНСУЛЬТАНТИ РОЗДІЛІВ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Розділ	Консультант (прізвище, ініціали та посада)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	Аналіз ринку мобільних ігор	25.03.2023	Виконано
2.	Проектування ігри	08.04.2023	Виконано
3.	Розробка ігри	15.04.2023	Виконано
4.	Формування висновків	29.04.2023	Виконано
5.	Навчального сайту	10.05.2023	Виконано
6.	Оформлення пояснювальної записки	18.05.2023	Виконано

Студент _____ Мантуляк Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Бойчук А.М.
(підпис) (прізвище та ініціали)

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Сторінка	Опис графічного матеріалу	Сторінка	Опис графічного матеріалу
15	Скріншот гри « Space shooter - Galaxy attack »	34	Головне меню
17	Скріншот гри « Smash hit»	34	Таблиця лідерів
18	Скріншот гри «Fruit ninja»	35	Вікно About
21	Приклад етапу розробки гри у Unreal Engine	36	Ігровий процес
22	Приклад етапу розробки гри у Unity	36	Ігровий процес
24	Приклад етапу розробки гри у Godot	37	Ігровий процес
32	Варіант фону гри	37	Зміна лічильника Score
32	Варіанти головних елементів гри.	38	Вікно Game over
33	Варіанти вибуху головних елементів гри.	38	Вписання ім'я
33	Аркуш спрайтів	39	Класи проекту

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці гри "Astrostrike" з використанням фреймворка LibGDX та мови програмування Java. Гра є 2D космічним шутером, де гравець контролює космічний корабель і бореться зі своїми ворогами, уникаючи їхні атаки та знищуючи їх.

У роботі детально досліджуються основні аспекти розробки гри, включаючи створення графічного інтерфейсу користувача, обробку вводу, логіку гри, управління рухом об'єктів, розміщення елементів на екрані, реалізацію ворожих сил, анімацію, звуковий супровід та систему очків.

У процесі розробки використовувалися відомі підходи та методики програмування графічних додатків, такі як використання спрайтів, атласів текстур, обробки колізій, асинхронного завантаження ресурсів та міжнародна локалізація.

Для реалізації гри використовувалися також додаткові інструменти та бібліотеки, зокрема SpriteBatch для відображення графіки, Camera та Viewport для керування відображенням, а також Android-фреймворк для розгортання гри для відображення на мобільних пристроях.

У результаті роботи була створена повноцінна гра "Astrostrike", яка демонструє навички розробки графічних додатків з використанням фреймворка LibGDX. Робота може бути використана як основа для подальшого розширення та вдосконалення гри, а також як приклад розробки ігрового додатку з використанням сучасних технологій та практик програмування.

КЛЮЧОВІ СЛОВА: ГРА, LIBGDX, JAVA, РОЗРОБКА ГРАФІЧНИХ ДОДАТКІВ, ГРАФІЧНІ ЕФЕКТИ, ГЕЙМДИЗАЙН.

SUMMARY

The qualification work is dedicated to the development of the game "Astrostrike" using the LibGDX framework and the Java programming language. The game is a 2D space shooter where the player controls a spaceship and fights against enemies, avoiding their attacks and destroying them.

The thesis thoroughly explores the main aspects of game development, including the creation of the graphical user interface, input processing, game logic, object movement control, screen element placement, enemy forces implementation, animation, sound effects, and scoring system.

Well-known approaches and techniques of graphical application programming were used during the development, such as sprite usage, texture atlases, collision detection, asynchronous resource loading, and international localization.

Additional tools and libraries were utilized to implement the game, including SpriteBatch for graphics rendering, Camera and Viewport for display management, and the Android framework for deploying the game on mobile devices.

As a result of this work, a fully functional game "Astrostrike" was created, showcasing skills in graphical application development using the LibGDX framework. The thesis can serve as a foundation for further expansion and enhancement of the game and as an example of game application development using modern technologies and actual programming practices.

Keywords: GAME, LIBGDX, JAVA, GRAPHICAL APPLICATION DEVELOPMENT, GRAPHICAL EFFECTS, GAMEDESIGN.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ……7
ВСТУП……..……..8
РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД ……10
1.1 Опис предметної області…….……10
1.2 Характеристика об’єкту розробки……12
1.3 Огляд аналогів……13
1.4 Вимоги до проекту……..18
1.5 Вибір засобів розробки……...19
Висновки до розділу 1……..27
РОЗДІЛ 2.ПРОЕКТНА ЧАСТИНА……28
2.1 Прототипування інтерфейсу……..28
Висновки до розділу 2…..……...29
РОЗДІЛ 3.ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ…….30
3.1 Спрайти проекту…….30
3.2 Опис програмного коду……..38
Висновки до розділу 3…….……53
ВИСНОВКИ……..54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…….55

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

UI – Інтерфейс користувача

UE – двигун Unreal Engine

libGDX – двигун для розробки ігрових додатків

JVM – Java Virtual Machine

AAA – клас високобюджетних комп’ютерних ігор

ВСТУП

В сучасному світі розробка комп'ютерних ігор стала надзвичайно популярною галуззю, яка поєднує технічні навички, творчий потенціал та інноваційні підходи. Аркадні ігри, зокрема 2D-ігри, завжди залучають широке коло гравців своєю простотою та веселощами.

Метою даного дипломного проекту є розробка 2D аркадної гри з використанням мови програмування Java та фреймворка LibGDX. Основним аспектом реалізації є імплементація рекламних вставок, що дозволить забезпечити фінансову стабільність гри та залучити увагу гравців.

Рекламні вставки є важливою частиною сучасних ігрових додатків, дозволяючи розробникам отримувати прибуток від своїх проектів. Їх правильна імплементація забезпечує монетизацію гри, зберігаючи при цьому задоволення та досвід користувачів.

У даному дипломному проекті будуть досліджені та розроблені методи та інструменти для створення 2D аркадної гри, зосереджені на реалізації рекламних вставок. В результаті цієї роботи очікується отримати готовий продукт - захоплюючу гру, яка буде ефективно використовувати рекламні вставки, забезпечуючи зручний та економічно вигідний досвід для користувачів.

Мета роботи. Створенні функціональної та привабливої гри, яка забезпечуватиме генерацію прибутку за рахунок рекламних вставок.

Об’єкт роботи. Процес розробки 2D аркадної гри з використанням мови програмування Java та фреймворка LibGDX.

Предмет роботи. Реалізація рекламних вставок у створеній 2D аркадній грі з метою монетизації та залучення уваги користувачів.

Завдання роботи. Розробка гри зі зручним інтерфейсом, захоплюючим геймплеєм та реалізацією рекламних вставок. В рамках завдання потрібно створити різноманітні рівні, графіку, звуковий супровід та оптимізувати процес вставок реклами.

Методи роботи. У роботі будуть використовуватись методи аналізу, проектування та програмування, зокрема використання мови програмування Java, фреймворка LibGDX

Результати роботи. Отримання готової 2D аркадної гри, яка має захоплюючий геймплей, зручний інтерфейс та успішно імплементує рекламні вставки, що забезпечать генерацію прибутку та залучення гравців.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Опис предметної області

Існує кілька категорій гравців- хардкорні, мідкорні та казуальні

Основний упор розробників ігор завжди був спрямований на них. Проте ці категорії становлять дуже невеликий відсоток людей в обсязі всієї планети.

Останнім часом аркадні проекти стали орієнтуватися на нон-геймерів, тобто на основну частину людей, які в принципі ніколи не грали в ігри.

Стандартні категорії гравців мають закладені патерни поведінки, які проектуються в базові, звичні їм механіки. Однак у нон-геймерів ці патерни взагалі відсутні. Тому вони намагаються перекласти на подібні проекти патерни, які були отримані в добре знайомих додатках, таких як WhatsApp, Tinder або Instagram.

Одним із завдань розробників аркадних ігор є створення геймплею, який підлаштовується під патерни, зрозумілі нон-геймерам. Важливо розуміти, що нон-геймери - це максимально широка аудиторія, і саме вони роблять продукт масовим та приносять найбільше прибутку [7].

Основою аркадних ігрових продуктів є геймплей. Це саме поняття core gameplay, яке не має конкретної мети гри. Гравець, який грає в ці ігри, повторює тисячі разів одні й ті самі дії. Геймплей у багатьох проектах має медитативний характер, оскільки гравець концентрується на одній дії [8].

Історія аркадних ігор налічує багато років і пройшла через численні етапи розвитку, від початкових приставок до сучасних ігрових систем і мобільних платформ. Ось коротка історія аркадних ігор:

Початок (1970-і роки): Аркадні ігри з'явилися в першій половині 1970-х років. Першим комерційно успішним аркадним автоматом була гра "Pong", створена компанією Atari. "Pong" стала популярною грою, що симулює пінг-понг, і послужила основою для подальшого розвитку аркадних ігор.

Золота ера (1980-ті роки): В 1980-х роках аркадні ігри пережили свою золоту еру. Було створено безліч іконічних ігор, таких як "Pac-Man", "Space Invaders", "Donkey Kong", "Galaga" та "Street Fighter". Ці ігри здобули величезну популярність та стали символами епохи.

Перехід до 3D (1990-ті роки): З появою комп'ютерної графіки 3D аркадні ігри почали переходити до нового рівня. Гра "Virtua Fighter" від Sega відкрила двері до тривимірного бойового жанру. Також з'явилися інші популярні ігри, такі як "Street Fighter II", "Mortal Kombat" та "Time Crisis", які використовували передові графічні можливості.

Аркадні автомати та консолі (2000-ті роки): З появою домашніх ігрових консолей та комп'ютерів аркадні автомати стали менш популярними. Багато з класичних аркадних ігор були портовані на консолі, що дозволило гравцям насолоджуватися їх улюбленими іграми вдома.

Сучасність (з 2010-х років): Сьогодні аркадні ігри присутні не тільки на ігрових консолях, але і на мобільних пристроях. Мобільні аркадні ігри стали дуже популярними завдяки своїй доступності та простоті. Існує безліч різноманітних аркадних ігор, які пропонують різні жанри та механіки гри.

Аркадні ігри змінилися з часом, але їх основна мета залишається незмінною - надати гравцеві швидку, розважальну ігрову досвіду, який викликає емоції та конкуренцію. Вони стали невід'ємною частиною культури відеоігор і продовжують розвиватися й еволюціонувати, пропонуючи нові ігрові концепції та інновації.

Розглядаючи «Space shooter - Galaxy attack» як приклад, Розробники тримають базу гравців залученою до повернення у гру кожного разу при її оновленні. Нові рівні додаються кожного тижня, кілька щоденних нагород за виконані завдання. Крім того у гравця є можливість виконувати ігрові квести різної складності, щоб отримати ще кращу видобуток.

Пропонування постійних бонусів за поповнення рахунку спонукає гравців витрачати значну суму грошей на видаватися більш вигідні пропозиції, наприклад, отримання ексклюзивних скарбів, таких як епічні костюми, за які необхідно інвестувати понад 100 доларів.

Особливістю аркадних ігор є те, що всі представники жанру використовують майже однакову ігрову механіку. Основне завдання гравця – якнайдовше протриматися «живим» в межах ігрового поля та заробити якомога більше балів або внутрішньо ігрових цілей.

1.2 Характеристика об’єкту розробки

При проектуванні аркадної гри важливо враховувати кілька основних вимог. Перш за все, геймплей гри має бути захопливим і цікавим для гравців. Гра повинна бути швидкоплинною, динамічною та веселою, з простими, але викликальними механіками. Управління грою повинно бути простим і інтуїтивним, щоб гравці могли легко керувати своїм персонажем або об'єктом.

Графічний дизайн має бути привабливим та привертати увагу гравців. Використання яскравих кольорів, деталізованих об'єктів та ефектних анімацій може створити привабливу графіку. Гра повинна також мати варіативність у рівнях або місіях, щоб пропонувати різні виклики та складності для гравців. Система прогресу або розблокування може надати стимул для продовження гри та досягнення нових можливостей [9].

Звуковий дизайн є ще одним важливим елементом. Атмосферний звуковий супровід, який включає музику та звукові ефекти, може покращити настрій та іммерсію гравців. Крім того, розробка гри з урахуванням різних платформ, таких як комп'ютери, консолі або мобільні пристрої, може розширити аудиторію гри [18].

1.3 Огляд аналогів

При проектуванні інформаційної системи слід враховувати цілі, для яких створюється програмний продукт, умови його експлуатації, процес розробки та вимоги майбутніх користувачів.

У випадку аркадних ігор дизайн рівнів є основою. Під час проектування, важливо вивчити конкуренцію, звернути увагу на оптимальне співвідношення нових елементів на рівнях, зовнішній вигляд елементів для найпопулярніших ігор та цільових аудиторіях, а також популярні та модні механіки.

На основі отриманих висновків, необхідно об'єднати всі ідеї та відібрати найбільш перспективні механізми та елементи. Далі, він буде відображати початкові концепції, на основі яких будуть створені графічні елементи. Рівні мають варіюватися за складністю, деякі повинні бути важкими, інші - легкими, а також естетично привабливими [19].

Спочатку розглянемо найбільш популярні аналоги, щоб зрозуміти головні від'ємності між ними. Першим аналогом обрана гра «Space shooter – Galaxy attack» – легендарна гра, яку люблять мільйони гравців у всьому світі.Гравцю потрібно керувати космічним кораблем та знищувати хвилі ворогів, щоб перейти на наступний рівень і отримати винагороду.

Потрібно вправно керувати кораблем за допомогою швидкої реакції та розумного планування, та пройти якнайбільше рівнів в межах ігрової сессії.

Гра містить тисячу рівнів, в яких змінюється швидкість проходження та кількість елементів поля (рис. 1.1).

Рисунок 1.1 – Скріншот гри « Space shooter - Galaxy attack »

Особливості гри Space shooter - Galaxy attack включають:

Екшн-орієнтований геймплей: Гра пропонує швидкий темп інтенсивних бойових сцен, де гравець керує космічним кораблем і бореться з безліччю ворожих супутників та босів.

Епічні космічні битви: Гра пропонує захоплюючі битви в космосі, де гравець зіштовхується з масштабними ворожими флотами і повинен проявити навички стрільби та маневрування, щоб вижити.

Різноманітність зброї та апгрейдів: Гравець має можливість покращувати свій космічний корабель, розблоковувати нові види зброї та екіпувати його

потужними апгрейдами, що дозволяє налаштовувати свою стратегію і підходити до гри за власними уподобаннями.

Різноманітність ворогів: Гра пропонує багато різних типів ворожих супутників і босів з унікальними атаками і поведінкою. Кожен з них вимагає від гравця особливої тактики та стратегії, щоб бути переможеним

Місії та досягнення: Гра має режим з різноманітними місіями та досягненнями, що додає виклику та розваги до геймплею. Гравець може виконувати різноманітні завдання та покращувати свої навички.

Візуальний стиль і атмосфера: Гра має яскраву та деталізовану графіку, ефектні візуальні ефекти та захоплюючу атмосферу космічних пригод

Можливість грати в одиночному та мультиплеєрному режимах: Гравець може насолоджуватися грою самостійно або з друзями, виконуючи завдання разом або змагаючись в режимі PvP.

«Space shooter - Galaxy attack» безкоштовна, але додаткові предмети в грі вимагають реальних коштів [14].

«Smash Hit» це відома безкоштовна гра, як « Space shooter - Galaxy attack » від Playrix.У грі Smash Hit гравець рухається вперед у віртуальному просторі, розбиваючи скляні перешкоди з точними кидками металевих куль, де кожен крок потребує уважності та реакції на швидко змінюючийся ландшафт.На даний момент у гри більше ніж 100 мільйонів завантажень, та близько 5 мільйонів відгуків у Google Play Market

У грі Smash Hit геймплей полягає в тому, що гравець керує кулею, яку він кидає в напрямку перед собою. Головна мета полягає в розбиванні скляних об'єктів, які зустрічаються на шляху (рис. 1.2).

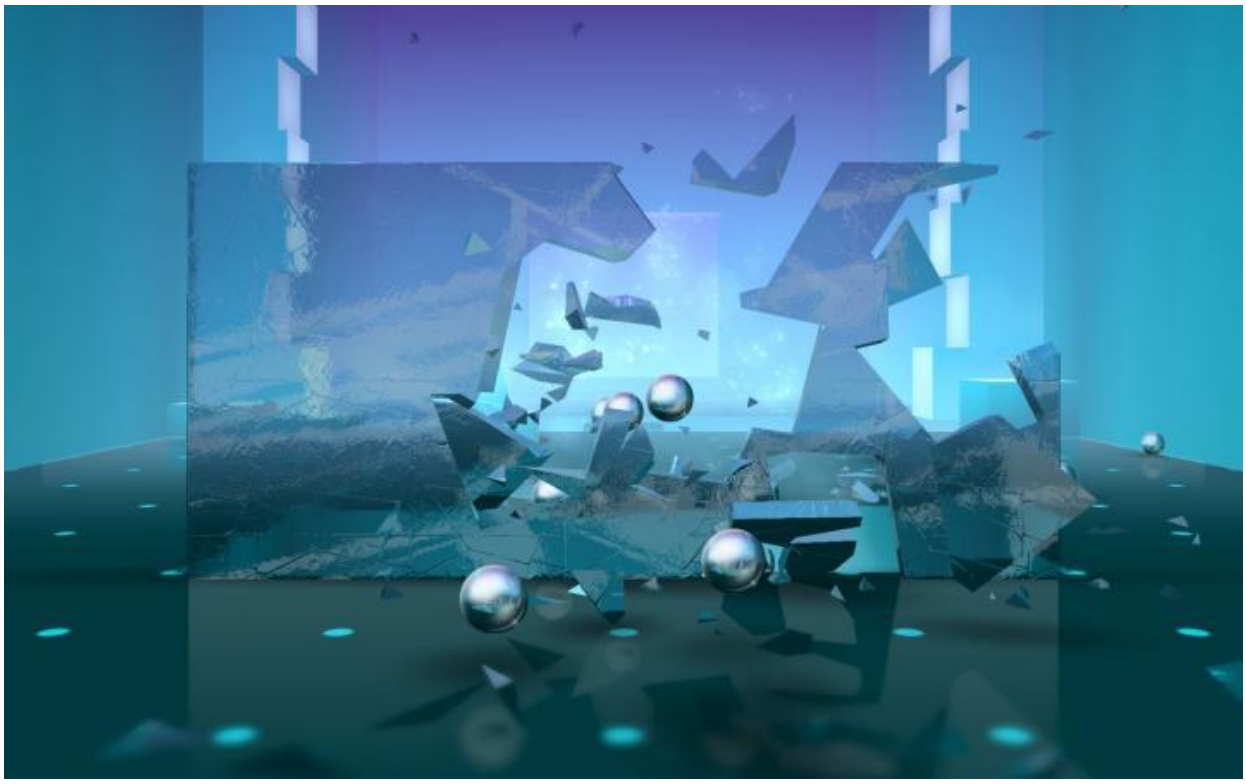

Рисунок 1.2 – Скріншот гри « Smash hit»

Гравець повинен майстерно наводити мішень і запускати кулі, щоб точно розбити об'єкти і продовжити свій шлях.У грі присутні різноманітні локації з унікальними ландшафтами та перешкодами. Гравець має швидко реагувати на змінюючийся оточуючий світ і зберігати достатню кількість куль, щоб продовжувати рухатися вперед.Гра пропонує режим безперервного проходження, де гравець може випробувати свою витривалість та спробувати пройти якомога більше рівнів без зупинки.

Геймплей гри Smash Hit вимагає від гравця точності, швидкості реакції і вміння стратегічно планувати свої дії. Це динамічна та захоплююча аркадна гра, де кожен кидок має значення і кожна помилка може призвести до поразки [15].

«Fruit Ninja» це популярна аркадна гра, в якій гравець має роль ніндзя, що рубає фрукти. Основна мета гри полягає в тому, щоб використовуючи пальці, розрубати різноманітні фрукти, які літають по екрану.

У грі присутні різні режими, такі як "Класичний", де гравець має рубати якомога більше фруктів, уникати бомб та збирати бонуси; "Аркада", де гравець має досягти певної кількості очок за обмежений час; "Вибухові фрукти", де фрукти можуть бути бомбами і потрібно уникати їх вибуху; та "Безперервний режим", де гравець може рубати фрукти без обмежень (рис. 1.3).

Рисунок 1.3 – Скріншот гри «Fruit ninja»

Гра використовує інтуїтивні жести, дозволяючи гравцю рубати фрукти одним рухом. Крім звичайних фруктів, у грі можуть з'являтися спеціальні предмети, такі як бомби, подвійні очки або мультиплікатори, які додають різноманітність інтерактивності та стратегії гри.

Fruit Ninja пропонує захоплюючий і швидкоплинний геймплей, де гравець може випробувати свої навички та реакційну швидкість. Гра також має чарівну графіку та звуковий супровід, що додають насолоди під час гри.

Ця гра стала популярною серед гравців різного віку і надихла багато подібних аркадних ігор. Вона пропонує просту, але веселу геймплейну концепцію, яка залучає гравців до повторного рубання фруктів та покращення своїх рекордів у таблиці лідерів [20].

Після аналізу топових аналогів при подальшому проектуванні об’єкту розробки слід враховувати декілька основних критеріїв:

– різноманіття графічних елементів ігрового поля та основних спрайтів;

– різноманітні рівні;

– можливість грати оффлайн;

– звукові ефекти до подій у грі

1.4 Вимоги до проекту

Для того, щоб гра була захоплюючою, гравці мають відчувати широкий спектр емоцій. Гармонійний баланс між складністю і легкістю, цікавістю і нудотою є основою успіху. Точне налаштування цього балансу дозволяє гравцям відчувати насичені емоції, наприклад, радість від успішного заключного ходу. Графік витрат і доходів може змінюватися за синусоїдальним законом або бути лінійним. Це дозволяє створювати періоди дефіциту і періоди перевищення, що впливає на почуття гравця.

Розробник гри, створюючи такий потік, може контролювати емоції гравця, створюючи напруження, а потім нагороджуючи його. Оптимальною практикою є пропонувати 2-3 різних емоції протягом однієї сесії гри. Однак, є складні рівні, коли гравець постійно відчуває лише обурення, наприклад, через необхідність платити для подолання перешкод, і такі моменти можуть бути використані для монетизації гри. Під час розробки сценарію рівнів необхідно враховувати ці аспекти.

1.5 Вибір засобів розробки

Розробка ігор ніколи не була простою, оскільки є багато різних двигунів і фреймворків, які пропонуються для вибору з різних мов програмування.

Вибір правильного двигуна є важливим кроком для розробника, і серед популярних варіантів можна виділити Unity, Unreal Engine, GameMaker та інші. Більшість нових мобільних ігор створюються в Unity, який має потужні можливості для графіки, керування монетизацією та інших функцій. Unreal Engine підтримує C++, Godot використовує Python, libGDX - Java, а Unity пропонує мову програмування C#.

Зробити правильний вибір двигуна може бути викликом для початківців, оскільки є багато альтернатив. Unreal Engine 4 має спеціальний інтерфейс для полегшення роботи початківцям, а також бібліотеку активів, які можна використовувати у грі. Цей двигун доступний безкоштовно, але після досягнення певного рівня доходу потрібно виплачувати 5% роялті. Unreal Engine є одним з найпопулярніших двигунів, розроблених компанією Epic Games, і пропонує широкий функціонал для створення ігор AAA-якості. Він безкоштовний з умовами та роялті, але має складний інтерфейс (рис. 1.4).

В сучасних ігрових розробках великий акцент ставиться на геймдизайні, креативному дизайні та створенні контенту, і Unreal Engine надає всі необхідні інструменти для дизайнерів, художників і програмістів, включаючи відкритий вихідний код, що дозволяє кастомізувати будь-яку частину проекту та забезпечує більшу гнучкість системи. Завдяки відкритості, в Unreal Engine існують великі спільноти, і офіційне управління знайденими помилками швидко вноситься в оновлення.

Рисунок 1.4 – Приклад етапу розробки гри у Unreal Engine

Ще однією перевагою Unreal Engine є наявність мультиплеєрної гри "з коробки", що дозволяє швидко створювати проект зі вбудованою мережевою грою. Такі можливості особливо важливі в багатокористувацьких іграх і є серйозною перевагою Unreal Engine порівняно з Unity, який їх не надає.

Unity, у свою чергу, є мультиплатформеною середовищем розробки ігор, яке дозволяє створювати додатки для більш як 25 різних платформ, включаючи ПК, ігрові консолі, мобільні пристрої та інтернет-прикладання. Unity є найпопулярнішим ігровим двигуном у світі завдяки своїм багатофункціональним можливостям та гнучкості, що дозволяє створювати практично будь-яку гру (рис. 1.5).

Unity є популярним серед розробників як у групі хобістів, так і серед студій AAA завдяки своїм неперевершеним міжплатформним можливостям. Використання Unity було успішним у створенні таких відомих ігор, як Pokemon Go, Heathstone, Rimworld, Cuphead та багато інших.

Рисунок 1.5 – Приклад етапу розробки гри у Unity

Незважаючи на те, що "3D" присутнє у назві Unity, він також надає інструменти для розробки 2D ігор. Програмістам сподобаються API сценаріїв C# та вбудована інтеграція з Visual Studio. Unity також пропонує JavaScript як мову сценаріїв і MonoDevelop як альтернативу Visual Studio для тих, хто бажає вибрати інше інтегроване середовище розробки [16].

Завдяки потужним інструментам анімації, Unity також стає вподобаною платформою для художників, дозволяючи легко створювати тривимірну анімацію або 2D-анімацію з нуля. Безкоштовна версія Unity Personal дозволяє розробникам випускати ігри без витрат на програмне забезпечення, якщо їхні доходи від ігор, створених за допомогою Unity, не перевищують 100 000 доларів

Corona - це двигун, призначений для створення 2D ігор, таких як платформери, шутери з видом зверху та ігри в ізометрії.[11] Цей двигун є

безкоштовним, дозволяючи розробникам зберігати всі прибутки від своїх ігор. Автори Corona SDK заробляють на продажу плагінів у внутрішньому магазині.

Для розробки ігор на платформі Corona потрібно мати знання скриптової мови Lua. Проект можна легко компілювати з мінімальними змінами.

Godot є безкоштовним та відкритим двигуном для розробки ігор, який набуває все більшої популярності серед розробників. Він пропонує рішення для створення як 2D, так і 3D ігор на різних платформах, включаючи Windows, macOS, Linux, Android, iOS та інші.

Godot має інтуїтивний і легкий у використанні інтерфейс, що дозволяє швидко створювати графічні та фізичні ефекти, анімацію персонажів, системи частинок та багато іншого. Він підтримує мови сценаріїв, такі як GDScript (мова, розроблена спеціально для Godot), C#, C++ і VisualScript, що дає розробникам велику свободу вибору.

Одним з головних переваг Godot є його вбудована система редактора, яка забезпечує повний цикл розробки гри. Він має інструменти для сценаріїв, управління ресурсами, візуального редагування інтерфейсу користувача, налаштування фізики та багато іншого. Крім того, Godot підтримує реалізацію мультиплеєрних ігор з можливістю з'єднання через локальну мережу або за допомогою Інтернета.

Ще одна важлива риса Godot - це активна спільнота розробників, яка допомагає один одному, надає поради та ресурси. Користувачі можуть обмінюватися знаннями, додатками, шаблонами і розширеннями, що сприяє швидкому розвитку і покращенню двигуна (рис. 1.6).

Рисунок 1.6 – Приклад етапу розробки гри у Godot

Плюси Godot:

Безкоштовність та відкритий вихідний код: Godot є повністю безкоштовним двигуном для розробки ігор. Розробники можуть використовувати його без обмежень та сплачувати роялті. Бути відкритим вихідним кодом також означає, що розробники можуть модифікувати та пристосовувати двигун до своїх потреб, а також вносити внески до спільноти розробників та користувачів.

Мультиплатформеність: Godot дозволяє розробникам створювати ігри для різних платформ, включаючи Windows, macOS, Linux, Android, iOS та багато інших. Це дозволяє розробникам розгортати свої ігри на багатьох пристроях без необхідності повторної розробки для кожної платформи.

Легкість використання та широкий функціонал: Godot має інтуїтивний та легкий у використанні інтерфейс, що дозволяє розробникам швидко створювати ігровий контент. Він також пропонує багато вбудованих інструментів та

функціоналу, таких як візуальне редагування, система анімації, система фізики та багато іншого, що сприяє розробці високоякісних ігор.

Мінуси Godot:

Обмежена документація та ресурси: У порівнянні з іншими популярними двигунами, такими як Unity або Unreal Engine, Godot має меншу кількість документації та навчальних ресурсів. Це може ускладнити для новачків оволодіння двигуном та вирішення проблем, з якими вони можуть зіткнутися під час розробки.

Обмежені можливості 3D-рендерингу: Хоча Godot підтримує створення 3D-ігор, його можливості в цій області обмежені порівняно з більш потужними двигунами, такими як Unreal Engine. Це може становити проблему для розробників, які планують створювати складні та реалістичні 3D-сцени.

Менша спільнота розробників: У порівнянні з дуже великими спільнотами, які мають Unity або Unreal Engine, спільнота розробників Godot є меншою. Це може вплинути на доступність підтримки та наявність готових рішень для різних проблем, з якими можуть зіткнутися розробники. Однак спільнота постійно росте, і ця проблема може зменшитися в майбутньому.

LibGDX - це відкритий програмний фреймворк для розробки ігор, написаний на мові програмування Java з деякими компонентами C і C++ для продуктивності. Цей фреймворк дозволяє розробляти ігри як для настільних, так і для мобільних платформ, використовуючи спільну кодову базу. Він є кросплатформеним і підтримує різні операційні системи, такі як Windows, Linux, Mac OS X, Android, iOS, BlackBerry, а також може працювати у веб-браузерах з підтримкою WebGL.

Перевага LibGDX полягає в його простоті та гнучкості, що дозволяє розробникам втручатися в будь-який аспект гри та налаштовувати його за своїми потребами. Для розробки під Android не потрібно встановлювати

додаткові плагіни, оскільки є прямий доступ до всіх можливостей Android SDK.[1] Однак, єдиним недоліком є те, що мова програмування Java не підтримується на iOS, тому якщо ви розробляєте мобільну гру на Java, вона буде доступна тільки для платформи Android.

Фреймворк LibGDX написаний на мові програмування Java і має добре документовану API.[2] Він легкий у налагодженні та використанні, і дозволяє використовувати всі функції Java 8, що підтримується на платформі Android. Управління залежностями виконується за допомогою Gradle, який є зручним інструментом. Хоча фреймворк має багато вбудованих компонентів, але їх кількість все ж менша, ніж у Unity [17].

Мова програмування Java була розроблена компанією Sun Microsystems та є об'єктно-орієнтованою. Вихідний код програм на Java компілюється в спеціальний байт-код за допомогою компілятора javac, який потім виконується на віртуальній машині Java.

Віртуальна машина Java (Java Virtual Machine або JVM) - це програма, яка виконує байт-код і перетворює його на інструкції, зрозумілі для обладнання[3].

Однією з головних переваг такого підходу є повна незалежність від операційної системи та апаратного забезпечення, що дозволяє запускати Java-додатки на будь-якому пристрої, який підтримує відповідну віртуальну машину.

Окрім цього, важливою особливістю технології Java є гнучка система безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які дії, які порушують встановлені повноваження програми (наприклад, незаконний доступ до даних або спроба підключитися до іншого комп'ютера), спричиняють негайне припинення роботи програми.

Серед недоліків використання віртуальної машини варто відзначити зниження продуктивності, яке зазвичай компенсується за допомогою таких методів:

JIT-технологія (Just-In-Time Compilation) - трансляція байт-коду в машинний код безпосередньо під час роботи програми;

використання переносного коду (наприклад, native коду) в стандартних бібліотеках, наприклад, SWT;

апаратні засоби, які прискорюють обробку байт-коду, наприклад, технологія Jazelle, яка підтримується деякими процесорами ARM.

Переваги мови програмування Java включають:

Об'єктно-орієнтованість: в Java все є об'єктом, що дозволяє легко розширювати функціональність через наслідування і композицію об'єктів.

Платформонезалежність: Java не залежить від конкретної платформи, оскільки байт-код виконується на віртуальній машині. Це дозволяє розповсюджувати додатки через Інтернет і запускати їх на різних пристроях, які підтримують JVM.

Java забезпечує простоту вивчення та розробки програм. Методи перевірки автентичності використовують шифрування з відкритим ключем. Компілятор генерує архітектурно-нейтральні об'єкти файлу, що дозволяє виконувати скомпільований код на різних процесорах з встановленою Java Runtime.

Java є портативною мовою програмування, оскільки вона є архітектурно-нейтральною та не залежить від реалізації конкретних специфікацій. Компілятор Java написаний на ANSI C з використанням переносимих принципів, що є підмножиною POSIX.

Java докладає зусиль для виявлення помилок у різних ситуаціях, засновуючись на часі компіляції, перевірці помилок та перевірці під час виконання. Вона також дозволяє розробникам писати програми, які можуть виконувати багато завдань одночасно, що сприяє створенню налагоджуваних інтерактивних графічних додатків.

Java байт-код перетворюється на льоту в машинні інструкції і не зберігається окремо. Це полегшує процес і робить його більш швидким і ефективним. Введення Just-In-Time компілятора дозволяє досягнути високої продуктивності виконання коду.

Програмування на Java вважається більш динамічним порівняно з C або C++, оскільки воно підлаштоване під змінні умови. Програми можуть ефективно обробляти інформацію та виконувати дії на основі цієї інформації, такі як перевірка доступу до об'єктів під час виконання.

Після аналізу існуючих програмних засобів для реалізації ігор, вирішено використовувати двигун libGDX та мову програмування Java.

Висновки до Розділу 1

В розділі аналітичного огляду було проведено детальний аналіз існуючих ігрових рішень та технологій, що використовуються у галузі розробки графічних додатків. Огляд охоплював різні аспекти, такі як графічний двигун, мови програмування, інструментарій та популярні фреймворки для створення ігрових додатків.

РОЗДІЛ 2. ПРОЕКТНА ЧАСТИНА

В акркадних іграх головний екран складається ігрового персонажа та перешкод, які гравці повинні долати. Кожен рівень має місії, які гравець повинен виконати, перш ніж закінчиться його ігрове життя. Головна мета гравця полягає у зароблені найбільшої кількості балів

Один з основних викликів для розробників аркадних ігор - створення захоплюючих та цікавих рівнів, щоб зберегти інтерес гравців. Вимірювання складності кожного рівня, виявлення помилок через ігрове тестування та досягнення загальної рівноваги - це дуже трудомісткий процес. Балансування рівня за допомогою послідовного тестування є складним завданням для розробника, і це викликає найбільше хвилювання у них [12].

2.1 Прототипування інтерфейсу

Прототипування є корисним інструментом для перевірки інтерфейсних рішень та загальної роботи продукту. Щоб зекономити час розробників і дизайнерів та уникнути зайвої переробки інтерфейсу в самому продукті, вони використовують прототипи. Крім того, безкоштовні програми для прототипування веб-сайтів та мобільних додатків допомагають економити не лише час, а й бюджет.

Проектування інтерфейсу ігор ґрунтується на основних принципах, які дозволяють розробникам досягти найкращих результатів:

Правильне зонування простору: Важливо виділити окремий екран для кожного користувальницького завдання. Розробники уникають "затемнення" функцій, оскільки це може призвести до плутанини. Чітке визначення "одного

завдання" також може бути складним, оскільки багато процесів мають кілька етапів. Важливо організувати простір таким чином, щоб гравець інтуїтивно міг використовувати інтерфейс гри, розуміючи призначення кожного елемента.

Відповідність орієнтації пристрою: Жанр та тип гри вимагають певної орієнтації екрана, яка може бути вертикальною або горизонтальною.

Використання додаткових зон: При проектуванні інтерфейсу ігор враховуються додаткові області - слайд-панелі, що розгортаються жестом або натисканням на приховану кнопку. Гравець не переміщується в інше місце, а швидко повертається до робочої області, повертаючись до основного екрану. Важливо також враховувати приховані області операційної системи та жести виклику, щоб уникнути незручностей для гравця, які можуть випадково відкрити "шторку" під час виклику інвентарю.

Мінімалізм: Перед створенням інтерфейсу гри, важливо видалити зайві деталі. Елементи переглядаються, а непотрібні елементи виключаються залежно від їх функціоналу або стилістичного оформлення.

Наприклад, гра "Manor Matters" має головний ігровий екран, який містить основні елементи для спрощення сприйняття. Так само інтерфейс завдань організований з урахуванням підказок, блоків підсилювачів, таймера, кнопки паузи, а зайві елементи виключаються [13].

Висновки до розділу 2

У розділі проектної частини було виконано детальне проектування гри "Astrostrike" з використанням фреймворка LibGDX та мови програмування Java. Цей розділ включав розробку структури гри, графічного інтерфейсу користувача, логіки гри, обробки вводу, управління рухом об'єктів, розміщення елементів на екрані, реалізацію ворожих сил, анімацію та систему очків.

РОЗДІЛ 3. ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

Ідея подібних ігор досить проста – є поле з постійно наступаючими перешкодами в бік гравця, Дані елементи відрізняються виглядом, анімацією, впливом на гравця, тощо. Завдання – минувати або знищувати перешкоди для подальшого просуванню по ігровому полю.

Існує багато різновидностей даного жанру, для дипломної роботи було обрано класичний варіант – космічний Shoot ’em up [4]. У цих іграх графічний персонаж рухається вперед автоматично і часто представляє собою літаючий транспортний засіб, такий як літак або космічний корабель, який вистрілює по багатьох ворогах, уникаючи перешкод [5].

3.1 Спрайти проекту

Спрайти - це графічні зображення, що представляють різні активи гри. Вони включають персонажів гравців, ворогів, снаряди та інші об'єкти. Усі ці елементи в грі є спрайтами, і вони зустрічаються всюди: на титульному екрані, на рівнях гри і навіть на ігровому екрані.

Спрайти використовуються в іграх для створення сцен. Кожен спрайт представляє окремий об'єкт. "Аркуш спрайтів" - це набір статичних зображень, які послідовно відтворюються для створення анімації.

Давайте розглянемо спрайти, що використовувалися під час створення ігрового додатку. По-перше, це задній план (бекграунд). Для фону було використано спрайт,на який накладено зображення зірок з анімацією їх падіння. Це допомагає збудити цікавість гравця до гри і мотивує його продовжувати пройти нові випробування (рис. 3.7).

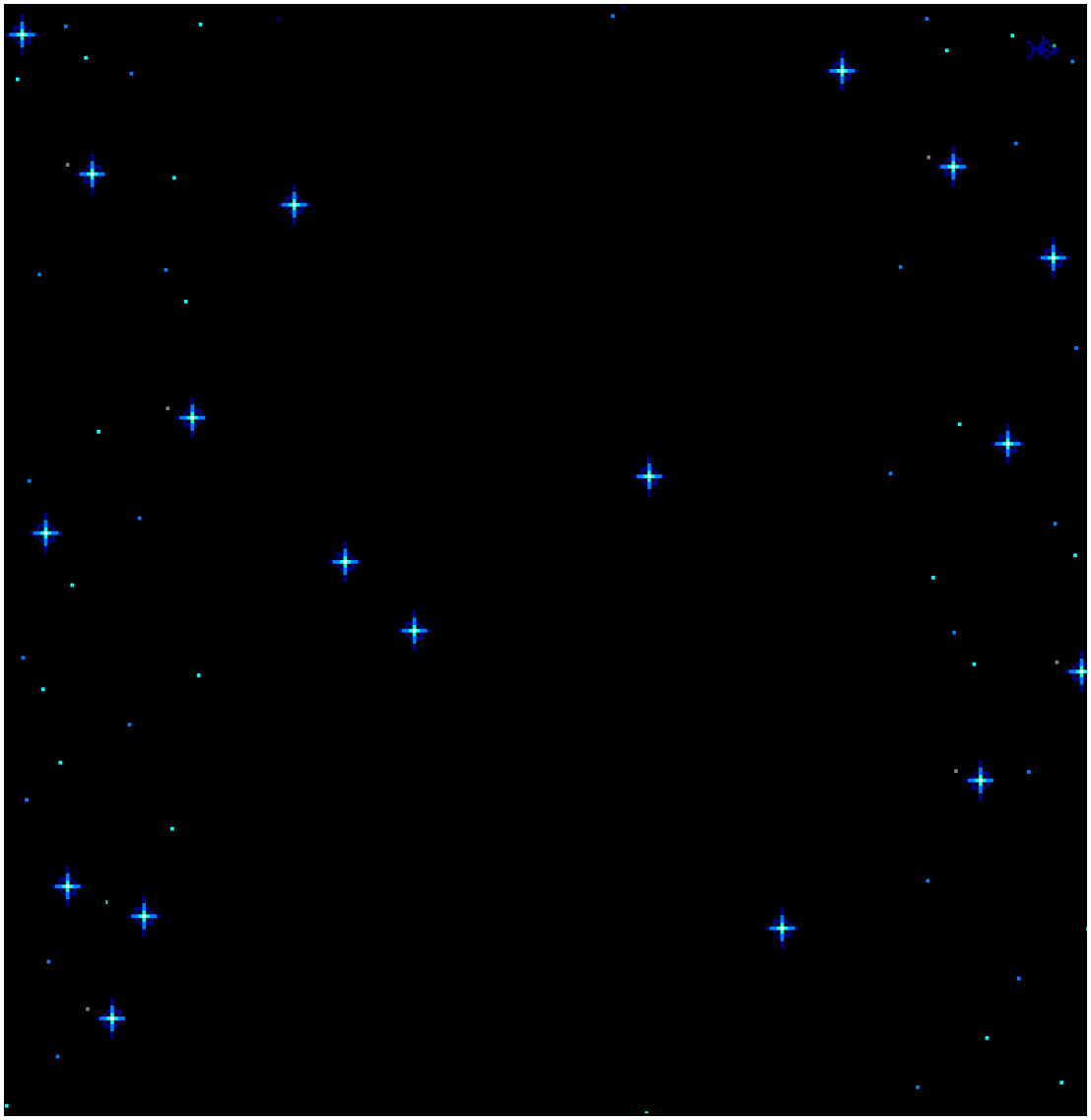

Рисунок 3.7 – Варіант фону гри

Наступний крок – це створення обновних елементів гри (рис. 3.8).

Рисунок 3.8 – Варіанти головних елементів гри.

Для кожного з цих елемені було промальовано анімація при влучанні (рис. 3.9).

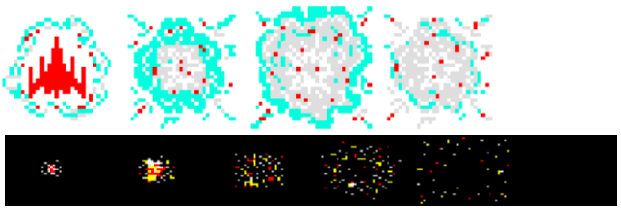

Рисунок 3.9 – Варіанти вибуху головних елементів гри.

Далі потрібно промалювати елементи управління, додаткові ефекти та з’єднати всі спрайти у єдиний аркуш спрайтів (рис. 3.10).

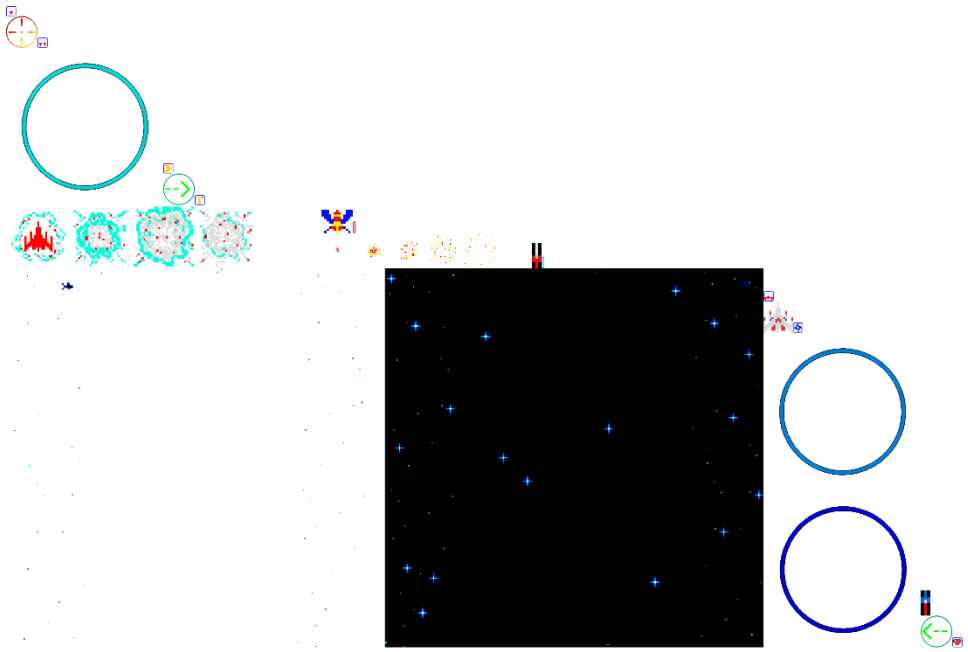

Рисунок 3.10 – Аркуш спрайтів

Розглянемо поетапно роботу ігрового додатку. Після запуску з’являється головне меню зверху якого зазначена назва гри, а нижче розташовані кнопки Play, Ranking About та Exit (рис. 3.11).

Рисунок 3.11 – Головне меню

Натиснувши на кнопку Ranking гравця переносить на таблицю лідерів у якій зображені як завчасно прописані,так і особисті результати гравця.

Натиснувши за межами таблиці гравця верне у головне меню (рис. 3.12).

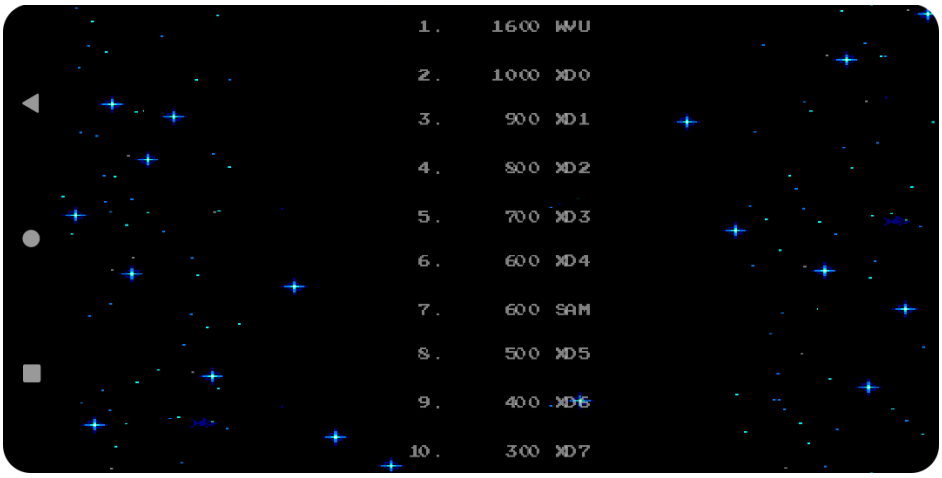

Рисунок 3.12 – Таблиця лідерів

Натиснувши на кнопку About гравцю виведется вікно з інформацією про цей проект та посилання при натисненні на яке, відкриється сайт фреймворка.

Щоб повернутися у головне меню потрібно натиснути на кнопку <<Leave to menu (рис. 3.13).

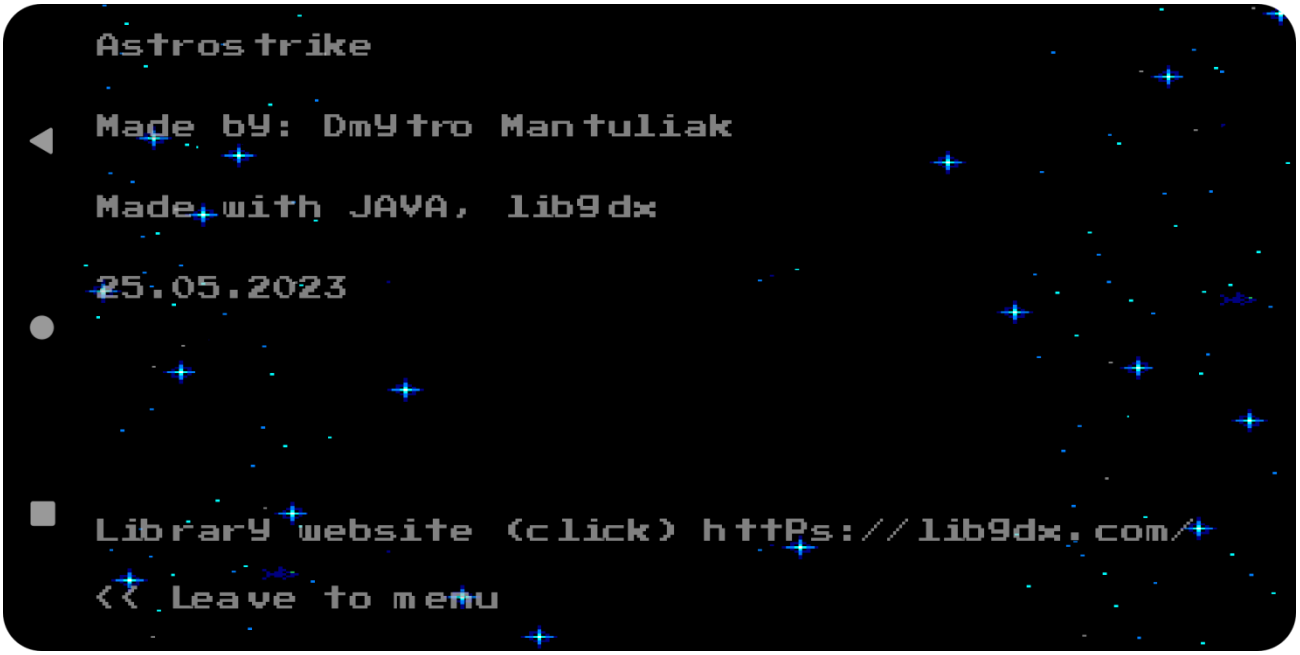

Рисунок 3.13 – Вікно About

Після натиснення на кнопку Play починається ігровий процес.

У нижньому центрі ігрового поля з’явиться ігровий персонаж, з лівого та правого боку від якого розташовані кнопки управління та кнопка пострілу.

З верхнього края екрану на персонажа будунь насуватися вороги, які мають хаотичний рух та періодично будуть атакувати його.

У верхньому центрі ігрового поля знаходиться лічильник очків (Scores), який збільшується після знищення ворога (рис. 3.14) У верхніх кутах екрану зображені показники щитів(Shields) та життів(Lives).При попаданні ворожого пострілу у персонажа,буде знятий один щит, а при ненаявності щитів, постріл забере життя.Після цього персонаж вибухне та поновиться у гру.

У кожного ворога також є щит, який знищиться після попадання, а наступнипостріл знищить самого ворога (рис. 3.15, 3.16, 3.17).

Рисунок 3.14 – Зміна лічильника Score

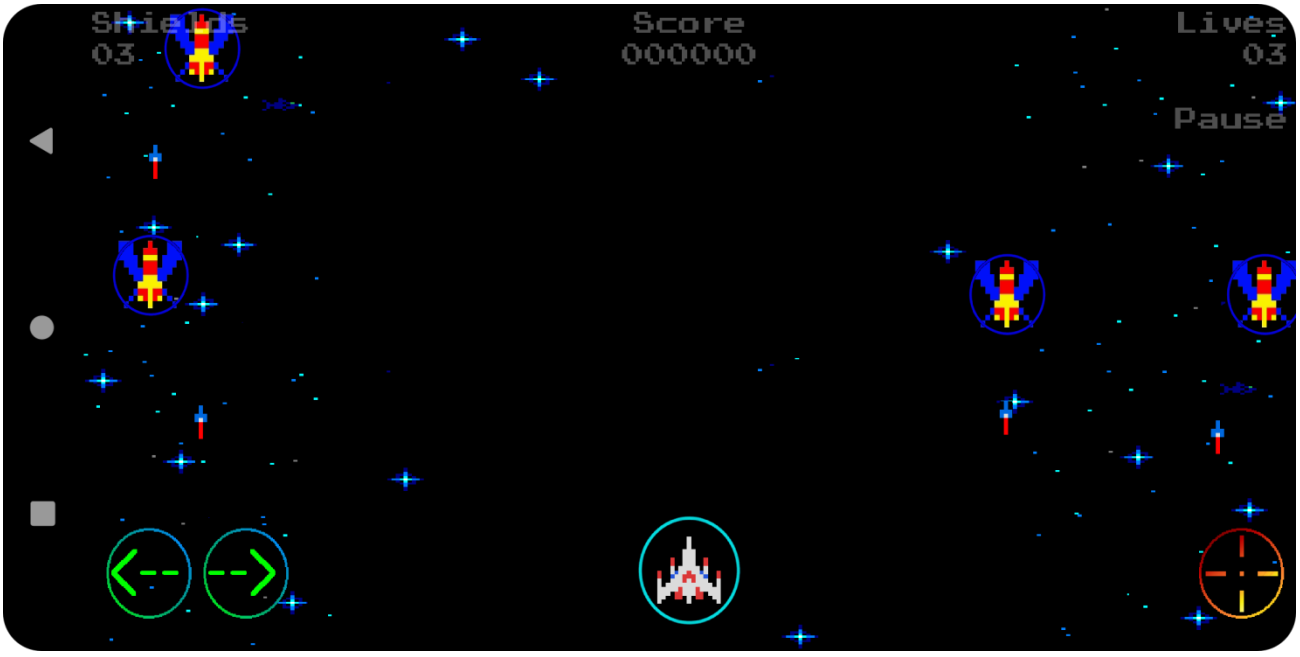

Рисунок 3.15 – Ігровий процес

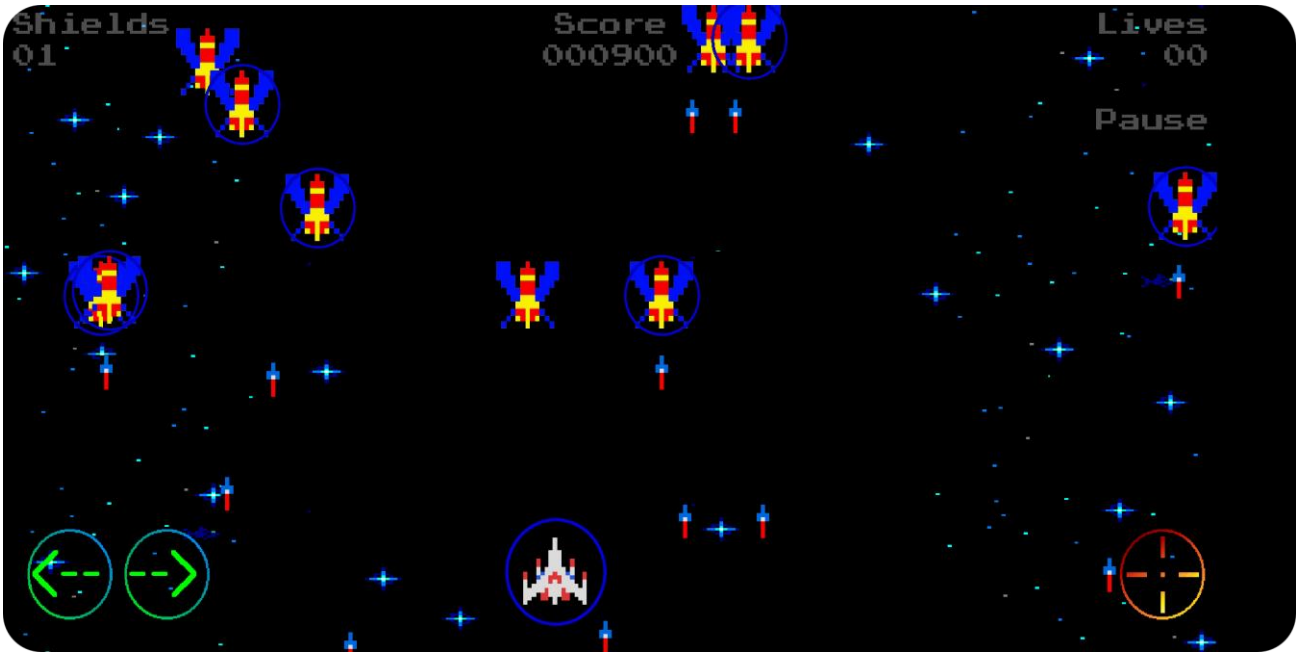

Рисунок 3.16 – Ігровий процес

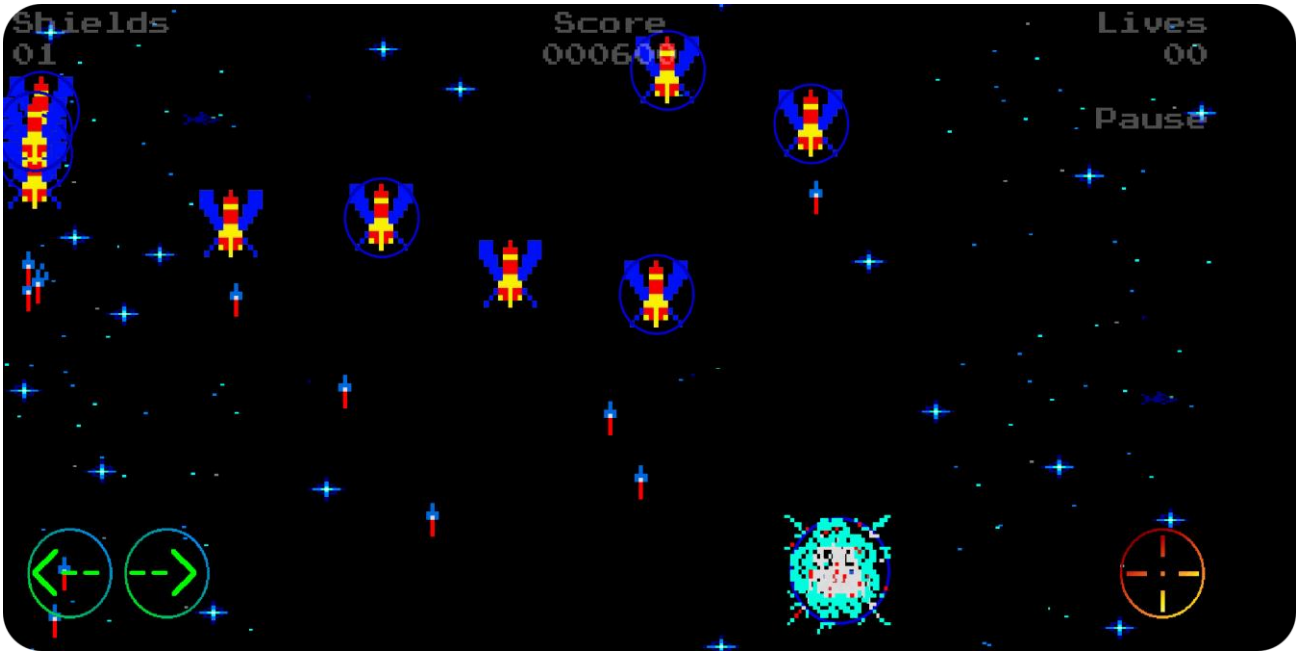

Рисунок 3.17 – Ігровий процес

Якщо показник життів дійшов до нуля гравця перекине у вікно Game over, де він побачит кількість очків та зможе вписати своє ім’я, яке потім висвітиться у таблиці лідерів.

Щоб вписати ім’я оберіть кожну літеру шляхом свайпу вверх або вниз (літери розміщенні у алфавітному порядку), та натисніть кнопку Submit score

Якщо не бажаєте вносити результат, то натисніть кнопку << Leave to menu, яка перенесе вас у головне меню (рис. 3.18, 3.19).

Рисунок 3.18 – Вікно Game over

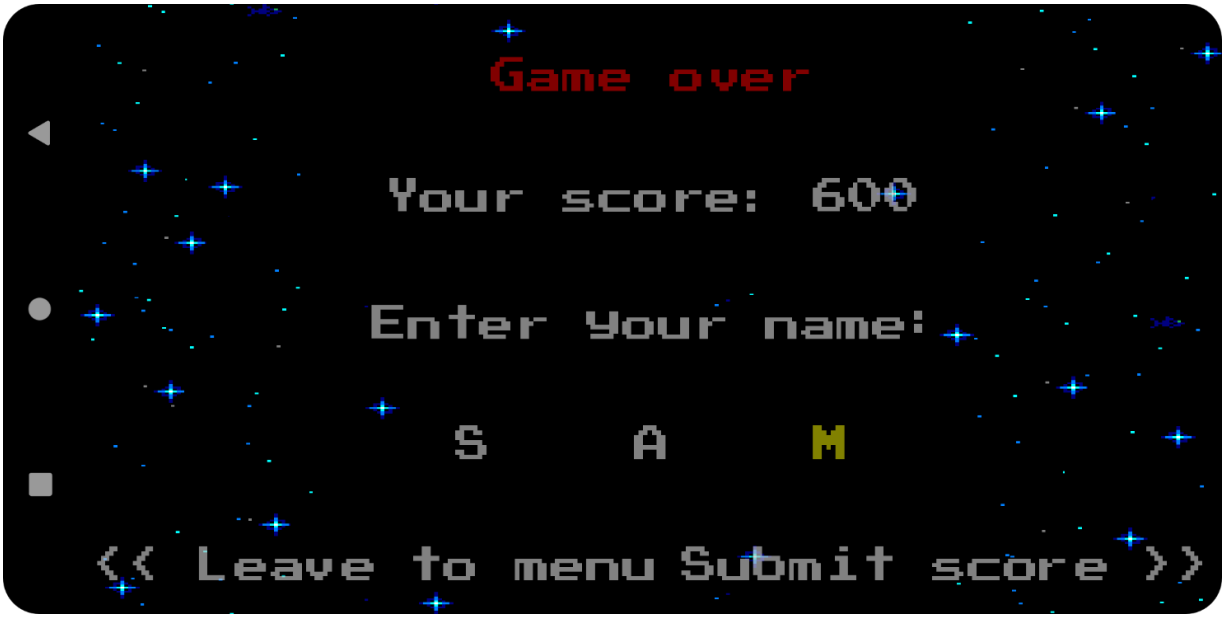

Рисунок 3.19 – Вписання ім’я

3.2 Опис програмного коду

Нижче наведені основні класи проекту (рис. 3.20).

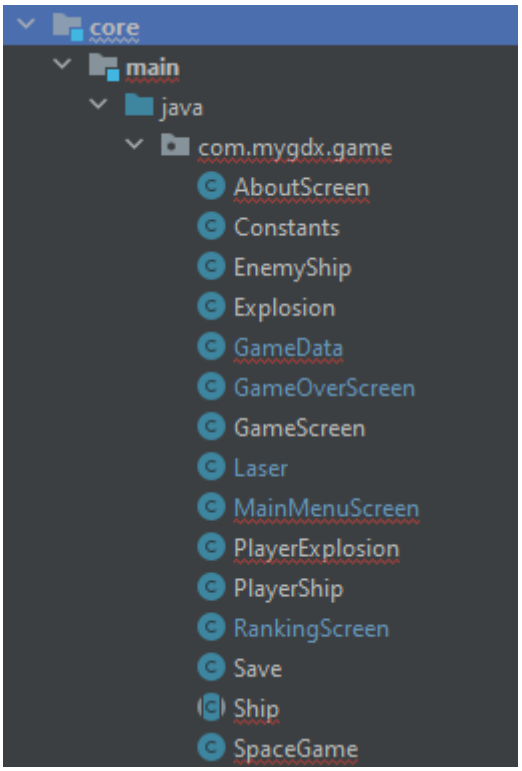

Рисунок 3.20 – Класи проекту

Абстрактний клас Game, який реалізує інтерфейс ApplicationListener в рамках фреймворку LibGDX.

```
public abstract class Game implements ApplicationListener {
	protected Screen screen;
	@Override
	public void dispose () {
		if (screen != null) screen.hide();
	}
	@Override
	public void pause () {
		if (screen != null) screen.pause();
	}
	@Override
```

```
	public void resume () {
		if (screen != null) screen.resume();
	}
	@Override
	public void render () {
		if (screen != null) screen.render(Gdx.graphics.getDeltaTime());
	}
	@Override
	public void resize (int width, int height) {
		if (screen != null) screen.resize(width, height);
	}
	public void setScreen (Screen screen) {
		if (this.screen != null) this.screen.hide();
		this.screen = screen;
		if (this.screen != null) {
			this.screen.show();
			this.screen.resize(Gdx.graphics.getWidth(),
Gdx.graphics.getHeight());
		}
	}
	public Screen getScreen () {
		return screen;}
```

В його структуру входять поля:

Screen screen: Це поле зберігає поточний екран гри.

dispose(): Метод, який викликається при закритті гри. Він приховує поточний екран, якщо він існує.

pause(): Метод, який викликається при призупинці гри. Він призупиняє поточний екран, якщо він існує.

resume(): Метод, який викликається при відновленні гри після призупинки. Він відновлює поточний екран, якщо він існує

render(): Метод, який викликається на кожній ітерації головного циклу гри. Він викликає метод render() поточного екрана, передаючи йому час, що пройшов з останнього виклику.

resize(int width, int height): Метод, який викликається при зміні розмірів вікна гри. Він передає нові розміри поточному екрану, якщо він існує.

setScreen(Screen screen): Метод, який встановлює новий екран гри. Він приховує попередній поточний екран, якщо він існує, та встановлює новий екран. Він також викликає методи show() та resize() нового екрана.

getScreen(): Метод, який повертає поточний екран гри.

Цей клас надає загальну структуру для керування екранами та їх життєвим циклом у грі. Конкретні імплементації класу Screen можуть використовуватись для відображення різних станів гри, таких як головне меню, екран гри тощо. Цей клас забезпечує зручну маніпуляцію екранами та передачу керування їх функціями під час виконання гри.

Код нижче представляє клас SpaceGame, який розширює клас Game і слугує основним класом для гри.

```
public class SpaceGame extends Game {
	public static Random random = new Random();
	public SpriteBatch batch;
	@Override
	public void create(){
		this.setScreen(new MainMenuScreen(this));
	}
	@Override
	public void dispose() {
	}
	@Override
	public void render() {
		super.render();
	}
	@Override
	public void resize(int width, int height) {
	}
}
```

Основні компоненти цього класу включають:

Random random: об'єкт класу Random, який використовується для генерації випадкових чисел у грі.

SpriteBatch batch: об'єкт класу SpriteBatch, який використовується для відображення спрайтів у грі.

Метод create встановлює початковий екран гри як екземпляр класу MainMenuScreen.[10]2 Цей метод викликається при створенні SpaceGame.

Методи dispose, render та resize визначені, але залишені порожніми. Ці методи можуть бути розширені для виконання різних дій під час звільнення ресурсів, відображення кадрів гри та зміни розміру вікна гри відповідно.

Отримуючи виклик render, він передає його до батьківського класу Game, що активує відображення поточного екрана гри.

Клас SpaceGame є головним класом гри, ініціалізує головні компоненти гри та керує переходами між екранами.

Клас MainMenuScreen, який імплементує інтерфейс Screen у фреймворку.

```
public class MainMenuScreen implements Screen {
    FileHandle baseFileHandle = Gdx.files.internal("I18NStrings");
    I18NBundle localizationBundle;
    String local = java.util.Locale.getDefault().getLanguage();
    Save save;
    SpaceGame game;
    public SpriteBatch batch;
    private Camera camera;
    private Viewport viewport;
    private TextureAtlas textureAtlas;
    private TextureRegion[] backgrounds;
    public float[] backgroundOffsets = {0,0};
    public float backgroundMaxScrollingSpeed;
    public float time_from_last_choice_change = 0;
    public float max_time_from_choice_change = 1f;

    BitmapFont font;
    float hudVerticalMargin, hudLeftX, hudRightX, hudCenterX, hudSectionWidth;
    int choice_number = -1;
```

FileHandle baseFileHandle: Об'єкт FileHandle, який використовується для доступу до файлу зі строки перекладу гри.

I18NBundle localizationBundle: Об'єкт I18NBundle, який містить локалізовані рядки для гри.

String local: Рядок, який містить мовний код поточної локалізації, отриманий з поточної мови системи.

Save save: Об'єкт Save, який використовується для збереження стану гри.

SpaceGame game: Об'єкт SpaceGame, який представляє головний клас гри.

SpriteBatch batch: Об'єкт SpriteBatch, який використовується для відображення спрайтів на екрані.

Camera camera: Об'єкт Camera, який використовується для керування відображенням на екрані.

Viewport viewport: Об'єкт Viewport, який визначає область відображення.

TextureAtlas textureAtlas: Об'єкт TextureAtlas, який містить текстури для відображення об'єктів на екрані.

TextureRegion[] backgrounds: Масив TextureRegion, який містить фонові зображення для головного меню.

float[] backgroundOffsets: Масив чисел, який визначає зсув зображень.

float backgroundMaxScrollingSpeed: Максимальна швидкість прокрутки фонових зображень.

float time_from_last_choice_change: Час, що пройшов з останньої зміни вибраного пункту меню.

float max_time_from_choice_change: Максимальний час, через який відбудеться автоматична зміна вибраного пункту меню.

BitmapFont font: Об'єкт BitmapFont, який використовується для відображення тексту на екрані.

float hudVerticalMargin, hudLeftX, hudRightX, hudCenterX, hudSectionWidth: Змінні, що визначають розміщення та розміри елементів інтерфейсу користувача (HUD).

int choice_number: Змінна, що визначає номер вибраного пункту меню.

Цей клас відповідає за відображення головного меню гри, управління вибором пунктів меню, локалізацію тексту та відображення фонових зображень.

Він також залежить від інших класів та об'єктів для забезпечення потрібної функціональності головного меню.

Код абстрактного класу Ship, який визначає основні характеристики та функціональність корабля в грі.

```
public abstract class Ship {

    float movementSpeed;
    float laserWidth, laserHeight;
    float timeBetweenShots;
    float timeSinceLastShot =0;
    float laserMovementSpeed;
    int shield;
    Rectangle boundingBox;

    public Ship(float movementSpeed,
            float xCenter, float yCenter,
            float width, float height,
            float laserWidth, float laserHeight, float laserMovementSpeed, int shield,
            float timeBetweenShots,
            TextureRegion shipTextureRegion, TextureRegion laserTextureRegion,
TextureRegion shieldTextureRegion) {
        this.movementSpeed = movementSpeed;
        this.boundingBox = new Rectangle(xCenter - width/2, yCenter - height/2, width,
height);

        this.laserWidth=laserWidth;
        this.laserHeight = laserHeight;
        this.timeBetweenShots = timeBetweenShots;
        this.laserMovementSpeed = laserMovementSpeed;
        this.shield = shield;
        this.shipTextureRegion = shipTextureRegion;
        this.laserTextureRegion = laserTextureRegion;
        this.shieldTextureRegion = shieldTextureRegion;
    }
```

float movementSpeed: Швидкість руху корабля.

float laserWidth, laserHeight: Ширина та висота снарядів, які випускає космічний корабель.

float timeBetweenShots: Інтервал часу між пострілами.

float timeSinceLastShot: Час, що пройшов з останнього пострілу.

float laserMovementSpeed: Швидкість руху снарядів.

int shield: Захист корабля.

Rectangle boundingBox: Прямокутник, що обмежує корабель.

public Ship(...): Конструктор класу Ship, який ініціалізує поля об'єкту корабля. Він отримує різні параметри, такі як швидкість руху, координати центру, розміри корабля та снарядів, швидкість руху снарядів, захист, а також текстури для відображення корабля, снарядів та захисту.

Цей клас є абстрактним, що означає, що він не може бути безпосередньо створений, але може бути розширений іншими класами, які конкретизують певний тип корабля в грі.

Код класу PlayerShip, який розширює абстрактний клас Ship і визначає специфічні характеристики та функціональність корабля гравця в грі.

```
public class PlayerShip extends Ship{

    int lives;

    public PlayerShip(float movementSpeed,
                      float xCenter, float yCenter,
                      float width, float height,
                      float laserWidth, float laserHeight,
                      float laserMovementSpeed, float timeBetweenShots, int shield,
                      TextureRegion shipTextureRegion, TextureRegion
laserTextureRegion,TextureRegion shieldTextureRegion) {
        super(movementSpeed, xCenter, yCenter, width, height, laserWidth, laserHeight,
laserMovementSpeed, shield, timeBetweenShots, shipTextureRegion, laserTextureRegion,
shieldTextureRegion);
        lives = 3;
        this.timeBetweenShots = 250000000;
    }

    @Override
    public Laser[] fireLasers() {
        Laser[] laser = new Laser[1];
        laser[0] = new
Laser(boundingBox.x+boundingBox.width*0.6f,boundingBox.y+boundingBox.height*1f,
                laserWidth, laserHeight,laserMovementSpeed,laserTextureRegion);

        timeSinceLastShot =0;

        return laser;
    }
```

```
}
```

int lives: Кількість життів гравця.

public PlayerShip(...): Конструктор класу PlayerShip, який викликає конструктор батьківського класу Ship з вказаними параметрами і ініціалізує поле lives значенням 3. Крім того, він також змінює значення timeBetweenShots на 250000000 (цей час використовується для обмеження пострілів гравця).

@Override public Laser[] fireLasers(): Перевизначений метод fireLasers() з класу Ship. В цьому методі створюється новий снаряд (об'єкт Laser) з вказаними координатами та параметрами швидкості руху, а також встановлюється значення timeSinceLastShot на 0. Метод повертає масив снарядів, який містить лише один снаряд.

Клас PlayerShip додає специфічну функціональність для корабля гравця, таку як керування пострілами та кількість життів гравця.

Код класу EnemyShip, який є підкласом класу Ship і описує ворожий корабель у грі.

```
public class EnemyShip extends Ship{
   Vector2 directionVector;
   float timeSinceLastDirectionChange=0;
   float directionChangeFrequency=0.65f;
   public EnemyShip(float movementSpeed,
                    float xCenter, float yCenter,
                    float width, float height,
                    float laserWidth, float laserHeight,
                    float laserMovementSpeed, float timeBetweenShots, int shield,
                    TextureRegion shipTextureRegion, TextureRegion laserTextureRegion,
TextureRegion shieldTextureRegion) {
          super(movementSpeed, xCenter, yCenter, width, height, laserWidth, laserHeight,
laserMovementSpeed, shield, timeBetweenShots, shipTextureRegion,
laserTextureRegion,shieldTextureRegion);
          directionVector = new Vector2(0, -1);
   }
```

Vector2 directionVector: Вектор напрямку руху ворожого корабля.

float timeSinceLastDirectionChange: Час, що пройшов з моменту останньої зміни напрямку руху.

float directionChangeFrequency: Частота зміни руху ворожого корабля.

public EnemyShip(...): Конструктор класу EnemyShip, який приймає параметри, необхідні для ініціалізації ворожого корабля. Він викликає конструктор батьківського класу Ship для ініціалізації загальних полів корабля, а також ініціалізує вектор напрямку руху.

Клас EnemyShip розширює функціональність базового класу Ship, додавши можливість зміни напрямку руху ворожого корабля. Він також містить всі необхідні дані та функціональність для ворожого корабля, такі як рух, вогонь зброєю та ушкодження.

Код класу Laser, який описує снаряд у грі.

```
public class Laser {
    Rectangle boundingBox;
    float movementSpeed;
    TextureRegion textureRegion;
    public Laser(float xPosition, float yPosition, float width, float height, float
movementSpeed, TextureRegion textureRegion) {
        this.boundingBox = new Rectangle(xPosition - width/2, yPosition, width, height);
        this.movementSpeed = movementSpeed;
        this.textureRegion = textureRegion;
    }
    public void draw(Batch batch)
    {batch.draw(textureRegion,boundingBox.x -
boundingBox.width/2,boundingBox.y,boundingBox.width,boundingBox.height);}
}
```

Rectangle boundingBox: Прямокутник, який визначає межі снаряду.

float movementSpeed: Швидкість руху снаряду.

TextureRegion textureRegion: Область текстури, яка представляє зображення снаряду.

public Laser(...): Конструктор класу Laser, який приймає параметри, необхідні для ініціалізації снаряду. В ньому створюється прямокутник

boundingBox з вказаними координатами та розмірами, а також ініціалізується швидкість руху та текстура снаряду.

public void draw(Batch batch): Метод draw(), який відповідає за відображення снаряду на екрані. В цьому методі зображення снаряду малюється на екрані з використанням батча (Batch), використовуючи вказану текстуру та координати з прямокутника boundingBox.[6]

Клас Laser представляє снаряд у грі і містить необхідні дані та функціональність для його руху та відображення.

Код класу Explosion, який описує вибух у грі.

```
public class Explosion {
   private Animation<TextureRegion> explosionAnimation;
   private float explosionTimer;
   private Rectangle boundingBox;
   Explosion(Texture texture, Rectangle boundingBox, float totalAnimationTime){
      this.boundingBox = boundingBox;
            TextureRegion[][] textureRegion2D = TextureRegion.split(texture, 111,
111);

      TextureRegion[] textureRegion1D = new TextureRegion[6];

      for(int i=0;i<6;i++){
         textureRegion1D[i] = textureRegion2D[0][i];
      }
      explosionAnimation = new Animation<TextureRegion>(totalAnimationTime/5,
textureRegion1D);
      explosionTimer=0;
   }
```

private Animation<TextureRegion> explosionAnimation: Анімація вибуху, представлена як об'єкт типу Animation з використанням TextureRegion.

private float explosionTimer: Час, що пройшов після початку вибуху.

private Rectangle boundingBox: Прямокутник, що обмежує межі вибуху.

Explosion(Texture texture, Rectangle boundingBox, float totalAnimationTime): Конструктор класу Explosion, який приймає текстуру,

прямокутник boundingBox та загальний час анімації totalAnimationTime вибуху. У цьому конструкторі:

textureRegion2D - двовимірний масив TextureRegion, який отримується шляхом розбиття текстури на частини з розміром 111x111 пікселів.

textureRegion1D - одновимірний масив TextureRegion, який отримується шляхом конвертації першого рядка textureRegion2D в одновимірний масив.

explosionAnimation - ініціалізується як об'єкт типу Animation, де кожен кадр анімації має тривалість totalAnimationTime/5, а кадри беруться з масиву.

explosionTimer ініціалізується значенням 0.

Клас Explosion предоставляє функціональність для відтворення анімації вибуху на екрані гри. Він використовує текстуру, прямокутник boundingBox та час анімації, щоб створити анімаційний ефект вибуху.

Код класу Save, який відповідає за збереження даних гри.

```
public class Save {
   public GameData gd;
   public Save(){
      gd= new GameData();
   }
   public void save(){
      try {String temp="";
         FileHandle file = Gdx.files.local("scores.txt");
         for(int i = 0; i < 10; i++) {
            if(i!=9)
             temp += gd.getNames()[i] + "," + gd.getHighScores()[i] + "\n";
            else temp += gd.getNames()[i] + "," + gd.getHighScores()[i];
         }file.writeString(temp, false);
      }catch(Exception e){
         e.printStackTrace();
         Gdx.app.log("Error", "Save Exception");
         Gdx.app.exit();
      }
   }
```

public GameData gd: Об'єкт типу GameData, який представляє дані гри.

public Save(): Конструктор класу Save, який ініціалізує об'єкт GameData.

public void save(): Метод save(), який зберігає дані гри. У цьому методі:

Створюється об'єкт FileHandle за допомогою Gdx.files.local().

За допомогою циклу for формується рядок temp, який містить імена гравців та їх високі рахунки, розділені комами та з новим рядком Записується рядок temp у файл file за допомогою методу writeString().

У разі виникнення виключення виводиться відповідне повідомлення про помилку, ігрова програма закривається.

Клас Save дозволяє зберігати дані гри у файл "scores.txt". Це використовується, наприклад, для збереження рекордів гравців.

Код класу GameData, який використовується для зберігання та управління даними гри.

```
public class GameData implements Serializable {
   private static final long serialVersionUID = 1;

   private final int MAX_SCORES = 10;
   public long[] highScores;
   public String[] names;

   private long tentativeScore;

   public GameData(){
      highScores = new long[MAX_SCORES];
      names = new String[MAX_SCORES];
   }
   public long[] getHighScores(){
      return highScores;
   }
   public String[] getNames(){
      return names;
   }
   public long getTentativeScore(){
      return tentativeScore;
   }
   public void setTentativeScore(int i){
      tentativeScore = i;
   }
   public boolean isHighScore(long score){
      return score > highScores[MAX_SCORES-1];
   }
```

private static final long serialVersionUID = 1;: Поле, яке вказує версію серіалізації класу GameData.

private final int MAX_SCORES = 10;: Кількість максимальних рекордів, що можуть бути збережені.

public long[] highScores;: Масив, який містить високі рахунки гравців.

public String[] names;: Масив, який містить імена гравців.

private long tentativeScore;: Тимчасовий рахунок.

public GameData(): Конструктор класу GameData, який ініціалізує масиви highScores та names.

public void init(): Метод init(), який налаштовує порожню таблицю високих рахунків. У цьому методі:

Встановлюються високі рахунки та імена для певних позицій у масивах highScores та names.

Викликається метод sortHighScores() для сортування високих рахунків у спадному порядку.

public long[] getHighScores(): Метод getHighScores(), який повертає масив високих рахунків.

public String[] getNames(): Метод getNames(), який повертає масив імен.

public long getTentativeScore(): Метод getTentativeScore(), який повертає тимчасовий рахунок.

public void setTentativeScore(int i): Метод setTentativeScore(), який встановлює значення тимчасового рахунку.

public boolean isHighScore(long score): Метод isHighScore(), який перевіряє, чи є заданий рахунок високим рахунком.

```
public void addHighScore(long newScore, String name){
   if(isHighScore(newScore)){
      highScores[MAX_SCORES-1] = newScore;
      names[MAX_SCORES-1]=name;
      sortHighScores();
```

```
        }
    }
    public void sortHighScores() {
        for (int i = 0; i < MAX_SCORES-1; i++)
            for (int j = 0; j < MAX_SCORES-i-1; j++)
                if (highScores[j] < highScores[j+1])
                {
                    // swap arr[j+1] and arr[j]
                    long temp = highScores[j];
                    highScores[j] = highScores[j+1];
                    highScores[j+1] = temp;

                    String tempString = names[j];
                    names[j] = names[j+1];
                    names[j+1] = tempString;
                }
        }
}
```

private static final long serialVersionUID = 1;: Поле, яке вказує версію серіалізації класу GameData.

private final int MAX_SCORES = 10;: Кількість максимальних рекордів, що можуть бути збережені.

public long[] highScores;: Масив, який містить високі рахунки гравців.

public String[] names;: Масив, який містить імена гравців.

private long tentativeScore;: Тимчасовий рахунок.

public GameData(): Конструктор класу GameData, який ініціалізує масиви highScores та names.

public void init(): Метод init(), який налаштовує порожню таблицю високих рахунків. У цьому методі:

Встановлюються високі рахунки та імена для певних позицій у масивах highScores та names.

Викликається метод sortHighScores() для сортування високих рахунків у спадному порядку.

public long[] getHighScores(): Метод getHighScores(), який повертає масив високих рахунків.

public String[] getNames(): Метод getNames(), який повертає масив імен.

public long getTentativeScore(): Метод getTentativeScore(), який повертає тимчасовий рахунок.

public void setTentativeScore(int i): Метод setTentativeScore(), який встановлює значення тимчасового рахунку.

public boolean isHighScore(long score): Метод isHighScore(), який перевіряє, чи є заданий рахунок високим рахунком.

public void addHighScore(long newScore, String name): Метод addHighScore(), який додає новий високий рахунок та ім'я до таблиці високих рахунків. Якщо новий рахунок є високим, то він заміщує найнижчий рахунок у таблиці, і таблиця сортується за спаданням рахунків.

public void sortHighScores(): Метод sortHighScores(), який сортує таблицю високих рахунків та відповідні імена за спаданням рахунків. Використовується алгоритм сортування бульбашкою.

Клас GameData використовується для збереження та управління даними про високі рахунки гравців. Ці дані можуть бути ініціалізовані, перевірені, змінені та сортовані за рахунками.

Також у проекті використовуються:

Клас GameScreen використовується для відображення гри, управління гравцем та ворогами, обробки взаємодії з кнопками та обробки логіки гри;

Клас AboutScreen використовується для відображення екрану "Про гру", обробки взаємодії з користувачем та відображення відповідних текстів та графічних елементів;

Клас Constants використовуються для визначення розмірів, позицій та інших параметрів елементів графічного інтерфейсу користувача, таких як кнопки, кораблі, лазери тощо.

Клас AndroidLauncher який є вхідною точкою для запуску гри на платформі Android. Він розширює клас AndroidApplication з бібліотеки LibGDX, яка надає функціональність для розробки і запуску ігор на різних платформах.

Клас RankingScreen який реалізує екран рейтингу гравців, де показуються найкращі результати гри. Він реалізує інтерфейс Screen і відображає відповідний інтерфейс користувача та логіку взаємодії з користувачем.

Конструктор GameOverScreen який отримує посилання на об'єкт SpaceGame і останній набраний рахунок гравця. Він ініціалізує змінні та об'єкти для подальшого використання на екрані гри.

Висновки до розділу 3

У розділі опису реалізації проекту було представлено детальний огляд реалізованих функцій і особливостей гри "Astrostrike". В цьому розділі були описані конкретні рішення та підходи, використані під час розробки, а також наведені приклади коду для кращого розуміння реалізації.

Під час реалізації проекту були успішно виконані такі завдання, як створення графічного інтерфейсу користувача, обробка вводу, логіка гри, управління рухом об'єктів, розміщення елементів на екрані, реалізація ворожих сил, анімація та система очків.

ВИСНОВКИ

У даній дипломній роботі була розроблена гра "Astrostrike" з використанням фреймворку LibGDX. Гра є космічним шутером, де гравець керує космічним кораблем і зіштовхується зі ворожими супротивниками.

У грі реалізовано основні елементи, необхідні для гри: головний екран гри, екран паузи, екран гри після поразки, екран про гру, екран рейтингу гравців. Кожен екран має власне відображення та взаємодії з користувачем.

Гра має графічний інтерфейс, що включає текстури для головних об'єктів гри, таких як космічний корабель, ворожі кораблі, вибухи, лазери та інші.

Окрім того, у грі реалізовано систему рейтингу гравців, де кращі результати зберігаються та можуть бути переглянуті гравцем.

Загальний код гри розбитий на окремі класи, що забезпечує чистоту коду та легкість розширення функціональності гри.

В цілому, гра "Astrostrike" є успішною реалізацією космічного шутера з елементами геймплею, графічним оформленням, а також системою рейтингу гравців. Вона може бути використана як основа для подальшого розвитку і розширення функціональності гри.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Документація Android studio. вебсайт. URL: https://developer.android.com/studio/intro (дата звернення: 15.05.2023)

2. Документація LibGDX. вебсайт URL: https://libgdx.com/dev/ (дата звернення: 16.05.2023)

3. Gerber A., Craig C. Introducing Android Studio. *Learn Android Studio*. Berkeley, CA, 2015. С. 1–26. URL: https://doi.org/10.1007/978-1-4302-6602-0_1 (дата звернення: 19.05.2023)

4. Oehlke A. Learning Libgdx Game Development. Packt Publishing, Limited, 2013. (дата звернення: 19.05.2023)

5. Stemkoski L. Shoot-em-up games. Java Game Development with LibGDX. Berkeley, CA, 2018. С. 83–98. URL: https://doi.org/10.1007/978-1-4842-3324-5_4 (дата звернення: 21.05.2023)

6. Szumny K. Gra na system Android wykorzystująca sensory oraz biblioteki LibGDX i Box2D : master's thesis. URL: https://ruj.uj.edu.pl/xmlui/handle/item/ (дата звернення: 22.05.2023)

7. Wallich P. Building your own arcade game [Hands On]. IEEE Spectrum. 2011. Т. 48, № 7. С. 26–27. URL: https://doi.org/10.1109/mspec.2011.5910440 (дата звернення: 23.05.2023)

8. Черпатюк А. С. Комп’ютерна гра в жанрі Аркада : bachelor's thesis. 2020. 84 с. URL: https://ela.kpi.ua/handle/123456789/34537 (дата звернення: 23.05.2023)

9. Тарапата Н., Семьонова М., Смотр О. О. Комп’ютерна гра. Інструменти і методологія створення комп’ютерних ігор : thesis. 2017. URL: http://hdl.handle.net/123456789/4367 (дата звернення: 25.05.2023)

10. DiMarzio J. F. Creating a Menu. Android Arcade Game App. Berkeley, CA, 2012. С. 13–21. URL: https://doi.org/10.1007/978-1-4302-4546-9_3 (дата звернення: 26.05.2023)

11. Free 2D Game Engine comparison careerkarma.com вебсайт. URL : https://careerkarma.com/blog/2d-game-engines/ (дата звернення: 15.05.2023)

12. Черпатюк А. С. Комп'ютерна гра в жанрі Аркада : bachelor's thesis. 2020. 84 с. URL: https://ela.kpi.ua/handle/123456789/34537 (дата звернення: 17.05.2023)

13. Вступ до історії геймдизайну. Частина 1: Аркадні ігри vokigames.com: вебсайт. URL: https://vokigames.com/ua/vstup-do-istoriyi-gejmdyzajnu-chastyna-1-arkadni-igry/ (дата звернення: 18.05.2023)

14. 187 Mobile Gaming Statistics for 2023 That Will Blow Your Mind udonis.co вебсайт. URL: https://www.blog.udonis.co/mobile-marketing/mobile-games/mobile-gaming-statistics (дата звернення: 22.05.2023)

15. Mobile Gaming Industry 2023 is.com вебсайт. URL: https://www.is.com/community/blog/mobile-gaming-trends/ (дата звернення: 20.05.2023)

16. Kosidło P., Kowalczyk K., Badurowicz M. Comparison of capabilities of the Unity environment and LibGDX in terms of computer game development. Journal of Computer Sciences Institute. 2021. Т. 21. С. 324–329. URL: https://doi.org/10.35784/jcsi.2735 (дата звернення: 12.05.2023).

17. Sajina R., Orehovacki T. User experience evaluation of 2D side-scrolling game developed using Overlap2D game editor and LibGDX game engine. 2018 41st International Convention on Information and Communication Technology,

Electronics and Microelectronics (MIPRO), м. Opatija, 21–25 трав. 2018 р. 2018. URL: https://doi.org/10.23919/mipro.2018.8400284 (дата звернення: 12.05.2023).

18. Merikivi J., Tuunainen V., Nguyen D. What makes continued mobile gaming enjoyable?. Computers in Human Behavior. 2017. Т. 68. С. 411–421. URL: https://doi.org/10.1016/j.chb.2016.11.070 (дата звернення: 12.06.2023).

19. Ринок відеоігор 2022: смартфони продовжують домінувати, а консолі втрачають прибуток. Itc.ua вебсайт URL: https://itc.ua/ua/articles/1125628/ (дата звернення: 20.05.2023)

20. Ігрові підсумки тижня: обвал у Європі, портрет мобільного геймера та стан ринку казуальних ігор. dev.ua вебсайт URL: https://dev.ua/blogs/posts/orest-1682142681 (дата звернення: 29.05.2023)