МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний аркадний шутер «Survivor» для ОС iOS»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-12 Акименко Владислав Валерійович

Кваліфікаційну роботу
захищено на засіданні ЕК
з оцінкою _____ «___» грудня 2022 р.

Науковий керівник _____ к.т.н., доц., Марченко А.В.
(підпис)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2022

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____ С. М. Ващенко
«___» _____ 2022 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Акименко Владислав Валерійович
(прізвище, ім'я, по батькові)

1 Тема проекту *Мобільний аркадний шутер «Survivor» для ОС iOS*

затверджена наказом по університету від «___»_____2022 р. № _____

2 Термін здачі студентом закінченого проекту «___» ___грудня___ 2022 р.
3 Вхідні дані до проекту технічне завдання на розробку мобільного додатку

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі, проектування ігрового мобільного додатку, розробка ігрового мобільного додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, постановка задачі, аналіз шутерів-аналогів, порівняльна таблиця додатків-аналогів, основні механіки аркадних шутерів, функціональні вимоги до ігрового додатку, моделювання роботи ігрового додатку, діаграма варіантів використання, засоби реалізації, архітектура додатку, розробка ігрового додатку, демонстрація ігрового додатку, висновки

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1.	Аналіз стану мобільних шутерів	07.11.2022-09.11.2022	
2.	Постановка мети та задачі	10.11.2022-11.11.2022	
3.	Опис функціоналу	14.11.2022-14.11.2022	
4.	Визначення методів та інструментів реалізації	15.11.2022-15.11.2022	
5.	Розробка календарного плану	16.11.2022-16.11.2022	
6.	Визначення ризиків	17.11.2022-18.11.2022	
7.	Розробка прототипу	21.11.2022-22.11.2022	
8.	Розробка ігрових механік	22.11.2022-25.11.2022	
9.	Налаштування матеріалів та анімації	28.11.2022-30.11.2022	
10.	Налаштування ігрової сцени	01.12.2022-05.12.2022	
11.	Розробка пошкоджень гравця та ворогів	07.12.2022-08.12.2022	
12.	Розробка інтерфейсу додатку	24.11.2022-24.11.2022	
13.	Тестування ігрового додатку	07.12.2022-07.12.2022	
14.	Створення документації	07.12.2022-08.12.2022	
15.	Архівація проекту	09.12.2022-09.12.2022	

Магістрант _____ Акименко В.В.

Керівник роботи _____ к.т.н., доц. Марченко А.В.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Мобільний аркадний шутер «Survivor» для ОС iOS».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 24 найменувань, та 2 додатків. Загальний обсяг роботи – 99 сторінок, у тому числі 22 сторінок основного тексту, 2 сторінки списку використаних джерел, 59 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці мобільного аркадного шутеру «Survivor» для ОС iOS з видом камери збоку.

У першому розділі виконано дослідження області мобільних шутерів, проведено аналіз ігрових-додатків аналогів, сформовано основні механіки та особливості шутерів у мобільних іграх.

Другий розділ присвячено визначенню мети проекту, задач та засобам реалізації.

Третій розділ присвячено структурно функціональному моделюванню проекту.

В останньому розділі продемонструвати процес розробки ігрового додатку, створення екранів шутеру, встановлення параметрів розміру та розташування елементів зі встановленими обмеженнями, виконано налаштування візуальних ефектів гри, було продемонстровано роботу ігрового додатку.

Результатом проведеної роботи є мобільний аркадний ігровий шутер «Survivor» для ОС iOS.

Практичне значення роботи полягає у створенні мобільного додатку, який може покращити логічні, тактичні навички користувачів, а у майбутньому може використовуватися для отримання прибутку.

Ключові слова: аркадний шутер, SpriteKit, Swift, мобільний додаток, iOS, анімація.

ЗМІСТ

ВСТУП

Сьогодні технології займають досить вагому частку в сучасному світі. Зокрема, смартфони досить сильно інтегрувалися в наше повсякденне життя. На 2020 рік кількість людей в 50 найбільших країнах за населенням, яка користувалася смартфонами, становила 3,8 мільярда людей [1]. Смартфони зі своєю потужністю є по суті портативними версіями комп'ютерів, що можуть виконувати ряд складних обчислень в найкоротший термін. Варто також відзначити, що з появою інтернету мобільні пристрої розширили сфери застосування, що призвело до появи величезної кількості мобільних застосунків.

Разом із розвитком технічних характеристик смартфонів ігрова індустрія та сфера розваг з кожним роком почала все більше звертати увагу на можливості для розширення в сторону мобільного геймінгу. Збільшення діагоналі екрану, ємності акумулятора, обсягів оперативної пам'яті та потужності процесора дозволили залучити велику кількість розробників саме в ці досить прибуткові сфери. За окремими підрахунками, ігрова індустрія може досягти позначки в 200 мільярдів доларів прибутку на рік [2].

Тепер існує велике різноманіття жанрів ігор, проте найчастіше очолюють п'єдестал рейтингу Google Play Market та App Store саме шутери та платформери в 2D або ізометричному стилі [3].

Метою роботи є розробка мобільного аркадного шутеру «Survivor» для ОС iOS для покращення логічних, тактичних навичок у гравців.

Для досягнення мети потрібно виконати наступні задачі:

– провести аналіз предметної області;

– розглянути основні механіки аркадних шутерів;

– виконати проектування ігрового додатку «Survivor» на основі аналізу даних;

– на основі проекту розробити ігровий додаток «Survivor».

Об’єкт дослідження: аналіз способів розробки мобільних ігрових додатків, індустрії мобільного геймінгу, зокрема аркадних шутерів, їх механік, особливостей, обмежень.

Предмет дослідження: реалізація ігрового додатку та основних механік шутерів з використанням фреймворку SpriteKit.

Практичне значення: розробка мобільного ігрового додатку «Survivor» для покращення логічних, тактичних навичок у гравців, можливість отримання прибутку в майбутньому.

Ігровий додаток може бути покращений та доопрацьований у майбутньому з метою покращення візуального аспекту гри, додавання або зміна ігрових механік, додавання кастомізації для персонажа. Більш того, це один із напрямків для отримання потенційного прибутку з гри.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний аналіз мобільних шутерів в ігровій індустрії

Сьогодні ігри відіграють у житті багатьох людей дуже важливу роль. Кожна окрема людина знаходить в них свій власний сенс, спосіб розважитися, отримати позитивні емоції, покращити свої навички [4].

Основними платформами для розміщення мобільних ігрових додатків на сьогодні є Google Play Store та App Store, ігри в яких розміщені за категоріями, а також є рейтинги найбільш популярних ігор за різними параметрами. Варто відзначити, що аркадні шутери досить часто займають перші сходинки в топі ігор серед гравців [5].

Шутери є похідним жанром від екшн ігор, де зазвичай приділяється більша частина уваги на знищення суперників за допомогою арсеналу, яким може користуватись гравець [6]. Зазвичай це вогнепальна зброя, хоча можуть використовуватися і інші типи зброї, наприклад, гранати для завдання непрямої шкоди, щити та здоров’я виступають у якості захисних показників. Показник здоров’я може бути відображено різними способами, хоча найбільш часто його відображають за допомогою шкали або цифр, хоча й буває таке, що прямий індикатор відсутній. У випадку втрати всього здоров’я гравець помирає та має можливість почати обраний рівень з самого початку або з останньої контрольної точки. Зазвичай шутери є досить високо інтенсивним жанром за кількістю подій за невеликий проміжок часу, хоча в деяких проектах інколи існує етап для паузи та для створення нової стратегії.

1.2 Аналіз шутерів-аналогів

Зараз існує велика кількість шутерів, що мають різні механіки, тому перед початком проектування ігрового додатку варто провести аналіз схожих додатків, визначити ключові функціональні та програмні особливості, механіки для розробки майбутнього додатку. Для аналізу було обрано наступні проекти: Hotline Miami, 20 Minutes Till Dawn, Brawl Stars.

Hotline Miami

Hotline Miami – інді-гра, що представляє собою 2D гру з видом зверху в жанрі top down shooter [7]. Сюжет гри відбувається в альтернативній версії Маямі 1989 року, де гравцю необхідно примірити на себе роль антигероя, який протидіє підпільним кримінальним авторитетам цього міста, повного таємниць, які і варто розгадати. Серед особливостей гри можна відзначити особливу жорстокість, вид зверху, елементи стелсу, неймовірно стильний візуальний та аудіо ряд. Гра поділена на розділи, кожен розділ має декілька етапів

Кожен новий етап представляє собою закриту будівлю, камера направлена зверху донизу. Метою кожного етапу є знищення всіх ворогів на поточній території, збір певних предметів, що будуть важливими для сюжетної лінії, інколи сюди включається поєдинок з босом.

Бойова система досить динамічна, оскільки кожен противник має штучний інтелект, що може повести себе непередбачувано на кожному повторному проходженні етапу. Інколи з противників може випадати зброя, що може допомогти в проходженні окремого етапу, проте деякі види, як от вогнепальна зброя створює значний шум, що може привабити не бажану увагу інших ворогів. Окрім цього гравець завжди має лише одне життя та кожне влучання по ньому

означає провал на поточному етапі. Важливим елементом гри є планування власних дій для досягнення максимального результату при проходженні рівня.

На рисунку 1.1 представлено процес гри.

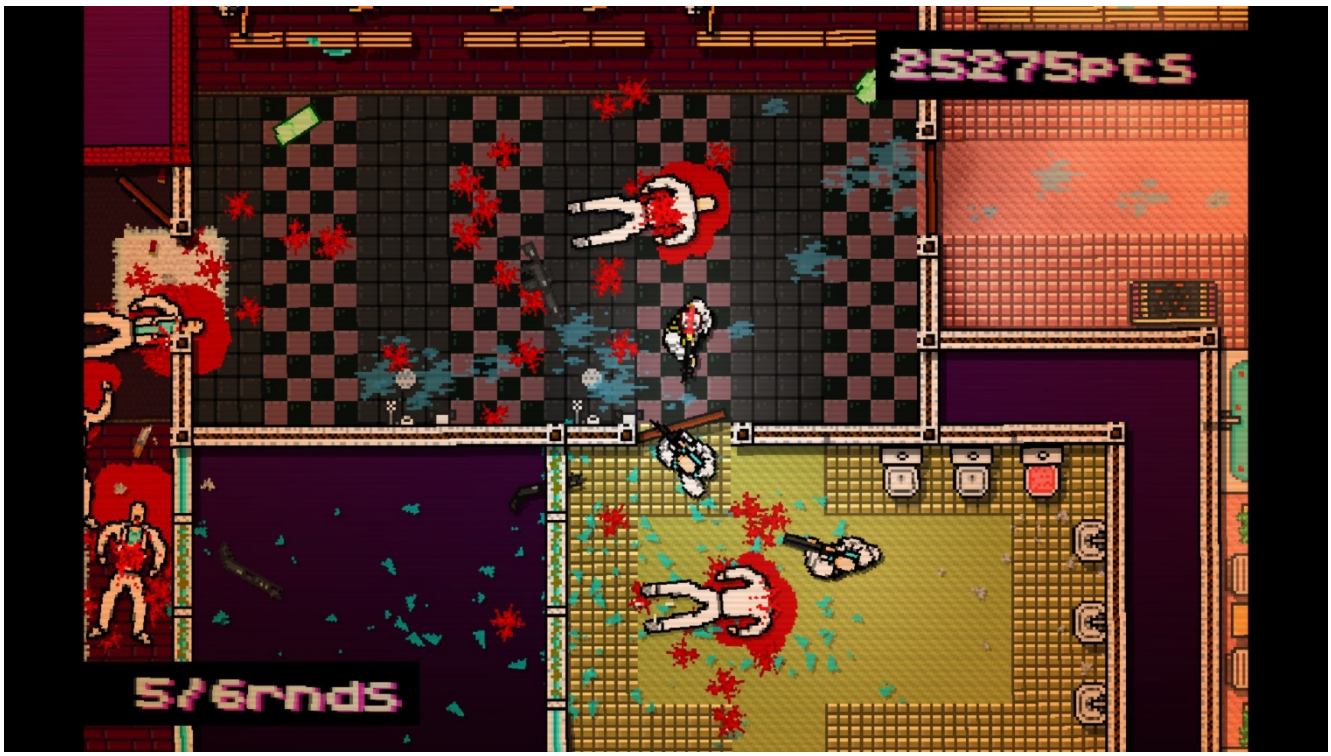


Рисунок 1.1 – Процес гри Hotline Mayami

20 Minutes Till Dawn

20 Minutes Till Dawn – інді-шутер від компанії Erabit. Гра являє собою 2D survival shooter з видом збоку [8]. Ціль гри полягає в тому, щоб протягом 20 хвилин реального часу відбивати напади монстрів. Гра має досить стильний візуальний ряд, виконаний в піксельній стилістиці.

Гра передбачає собою звичайні та короткі сесії по 20 і 10 хвилин відповідно, в яких гравцю необхідно протягом реального часу ухилятися від великої кількості різноманітних чудовиськ, використовуючи широкий арсенал зброї та користуючись спеціальними бонусами при підвищенні рівня. Бонуси являють собою випадковий набір із 4 особливостей для покращення захисту або збільшення

шкоди, яка буде завдана ворогам. Гра проходить на обмеженій території з ворогами, що стають більш небезпечними з кожною хвилиною, проведеною в сесії.

Гра є досить динамічною, керування здійснюється двома джойстиками, один з яких відповідає за пересування гравця, а інший за приціл під час стрільби. Джойстик, відповідальний за стрільбу, має 2 режими, відповідно до яких і здійснюється стрільба. Якщо гравець тримає джойстик по центру, то стрільба відбувається по найближчим ворогам, коли гравець рухає джойстик, то стрільба відбувається в визначеному напрямі.

Варто відзначити, що у перервах між сесіями гравець має можливість скористатися магазином, щоб придбати нових персонажів з унікальними здібностями, придбати нову зброю, а також покращити характеристики персонажів за допомогою за допомогою окремої вкладки з навичками.

Процес гри зображено на рисунку 1.2.

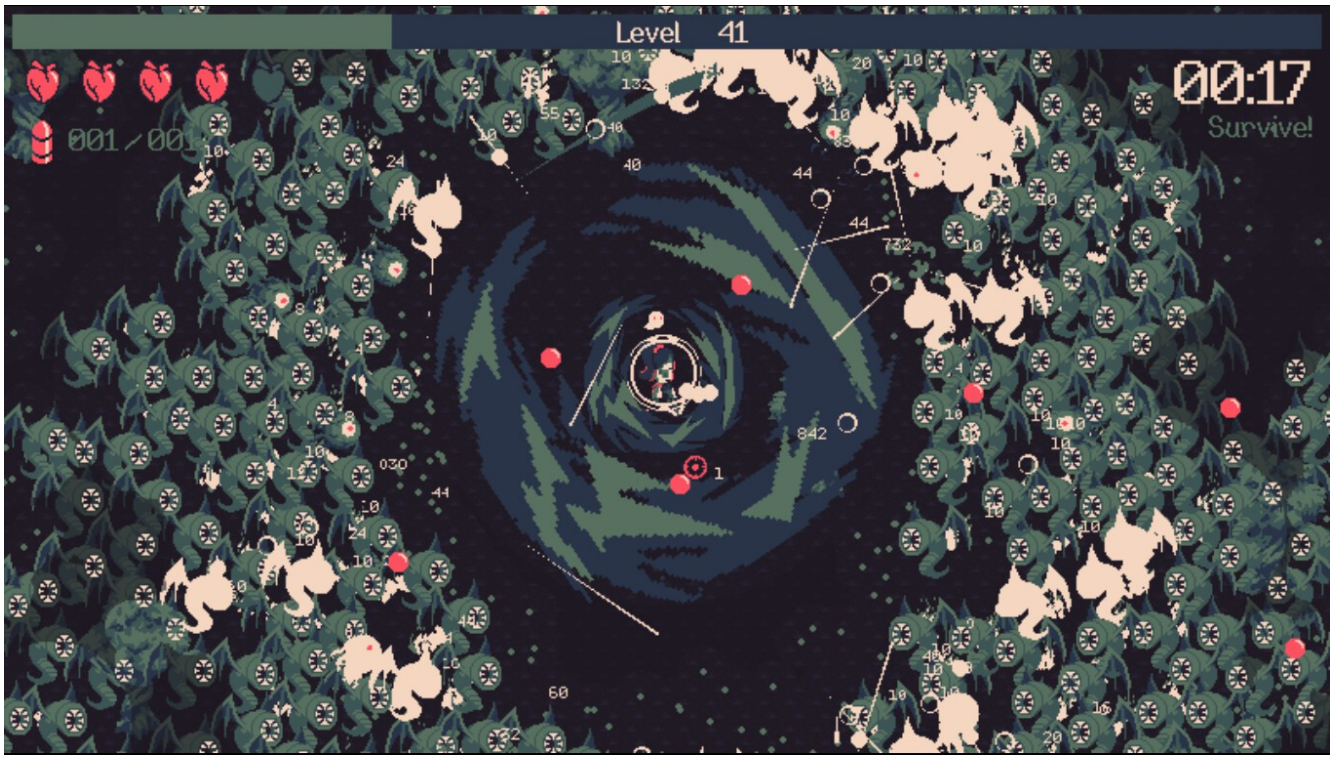


Рисунок 1.2 – Процес гри 20 Minutes Till Dawn

Brawl Stars

Brawl Stars – багатокористувальницька гра від компанії Supercell, що поєднує в собі жанри MOBA (багатокористувальницька онлайн бойова арена) та ізометричний 3D шутер [9].

Суть гри полягає в знищенні команди суперника в режимі реального часу з іншими гравцями, при цьому кожен гравець грає за унікального чемпіона з власними характеристиками та унікальними здібностями.

На рисунку 1.3 представлено процес гри.

Рисунок 1.3 – Процес гри в Brawl Stars

Керування в грі здійснюється за допомогою двох джойстиків для переміщення та пострілів, а також присутня кнопка активації спеціальної здібності.

Окрім цього в грі присутній магазин, що містить купу вигаданих та реальних персонажів з. можливістю поліпшення їх бойових характеристик або зміни їх зовнішнього вигляду.

Розглянувши декілька популярних проектів в жанрі shooter, були обрані особливості та механіки для майбутнього ігрового додатку. Зокрема, ігровий додаток буде виконаний в 2D з видом камери збоку, індикатором здоров’я у вигляді шкали. Шкалу здоров’я можна буде поповнити, підібравши відповідний бонус. Крім того, гравець, підібравши спеціальні кристали, зможе покращити свій арсенал при підвищенні рівня. Ціль гри полягатиме в якомога довшому відбиванні нападу ворогів.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

За результатом аналізу предметної області було сформульовану мету кваліфікаційної роботи: розробка мобільного аркадного шутера «Survivor» для ОС iOS.

Для досягнення поставленої мети дипломної роботи були визначені наступні задачі:

– проаналізувати обрану предметну область та додатки-аналоги;
– визначити інструментарій для створення додатку;
– провести планування робіт;
– виконати проектування ігрового додатку;
– розробити спрайти ігрового додатку;
– розробити основні механіки та поведінку ворогів;
– розробити інтерфейс та основні рівні додатку;
– налаштувати сцени та розробити магазин;
– виконати тестування додатку.

Ігровий додаток має бути направлений в першу чергу на цікаве проведення часу у грі, а також на покращення тактичних навичок та реакції.

Розробка ігрового додатку передбачає наступний функціонал:

– вибір рівня;
– проходження рівня;
– перегляд фінального результату;
– можливість покращення характеристик в магазині гри;
– налаштування параметрів гри.

2.2 Методи дослідження

Для розробки ігрового додатку були виділені основні методи дослідження, а саме теоретичні та емпіричні; системно-функціональні та моделювання.

Для вивчення основних механік шутерів використовується метод аналізу для більш глибшого розуміння основних принципів побудови шутерів, основні практики при розробці проектів даного напрямку.

Системно-функціональні методи та моделювання дозволяють створити функціональну модель процесу розробки ігрового додатку. Наступні моделі будуть створені за допомогою зазначеного методу:

– Діаграма IDEF0 [10] – для більш детального процесу побудови мобільного додатку;

– UML діаграма [11-12], зокрема діаграма варіантів використання (Use Case) для визначення основних можливих сценаріїв під час використання додатку.

2.3 Вибір засобів реалізації

Для створення мобільного аркадного шутера «Survivor» для ОС iOS було обрано наступні засоби реалізації:

– Мова програмування Swift;

– Середовище розробки XCode;

– Фреймворк SpriteKit для створення ігрової сцени;

– Adobe Photoshop для роботи з графікою додатку;

– UIKit and Storyboards для створення UI ігрового додатку [13].

Swift – мова програмування представлена компанією Apple на заміну Objective-C [14-16]. Swift є досить легко мовою для написання та читання коду,

синтаксис розробленим досить зручним чином, що дозволяє уникнути великої кількості помилок під час написання коду [17].

Серед основних переваг Swift можна відзначити наступні:

– мова програмування була розроблена під ОС iOS, що дозволяє простіше використовувати нативні фреймворки Apple;

– автоматичних лічильник посилань (Automatic Reference Counting) є одним із елементів для керування пам’яттю пристрою, що дозволяє менше піклуватися про її виділення на конкретні задачі;

– швидкість мови дозволяє значним чином заощадити час компіляції коду;

– захист від помилок. Runtime crash під час побудови проекту дозволяє завчасно виправити помилку під час написання коду.

Оскільки Swift є нативною мовою програмування для ОС iOS, то найліпшим рішенням для написання коду є середовище XCode, створене Apple для розробки додатків під усі пристрої компанії.

Беручи до уваги різні можливості для реалізації ігрової сцени було прийнято рішення використати фреймворк SpriteKit, що представляє собою досить елегантне рішення для створення 2D ігор без значних витрат ресурсів пристрою та з досить швидкою взаємодією [18].

Для створення та редагування спрайтів буде використана програма Adobe Photoshop, що містить великий набір інструментів для налаштування зображень під потреби розробника.

Інтерфейс додатку буде реалізовано за допомогою UI фреймворку UIKit, а також екрани додатку та різні елементи будуть створені за допомогою UIStoryboard [19].

3 ПРОЕКТУВАННЯ ІГРОВОГО МОБІЛЬНОГО ДОДАТКУ

3.1 Структурно-функціональне моделювання процесу

Для відображення функціонування системи використовується методологія IDEF0. Дана методологія дозволяє графічно відобразити за допомогою потоків даних логічні взаємозв’язки між компонентами ігрового додатку [20]. На рисунку 3.1 розглянуто діаграму A-0 верхнього рівня для ігрового додатку «Survivor».

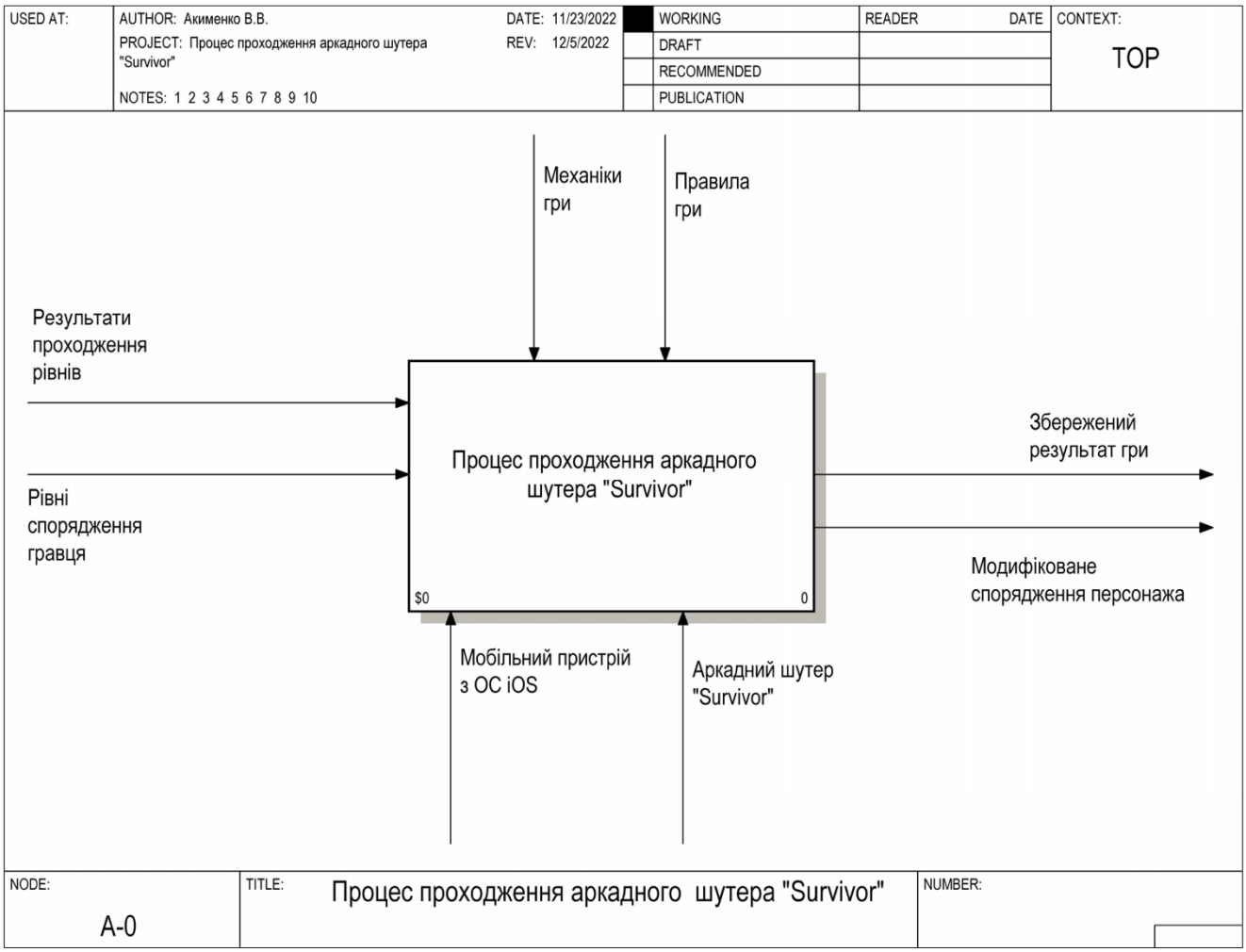


Рисунок 3.1 – Контекстна діаграма A-0

У ході аналізу предметної області були сформовані потоки даних.

– Входи моделі: результати проходження рівнів, рівні спорядження гравця.

– Виходи: збережений результат гри, модифіковане спорядження персонажа.

– Елементи управління: механіки гри, правила гри.

– Механізми: мобільний пристрій з ОС iOS, аркадний шутер «Survivor»

Наступним кроком є декомпозиція на функціональні блоки:

1. Ознайомлення з головним меню.
2. Вибір бажаного рівня.
3. Покупка рівнів спорядження в магазині гри.
4. Гра на обраному рівні.
5. Ознайомлення з результатами гри.

Декомпозиція діаграми А-0 представлена на рисунку 3.2.

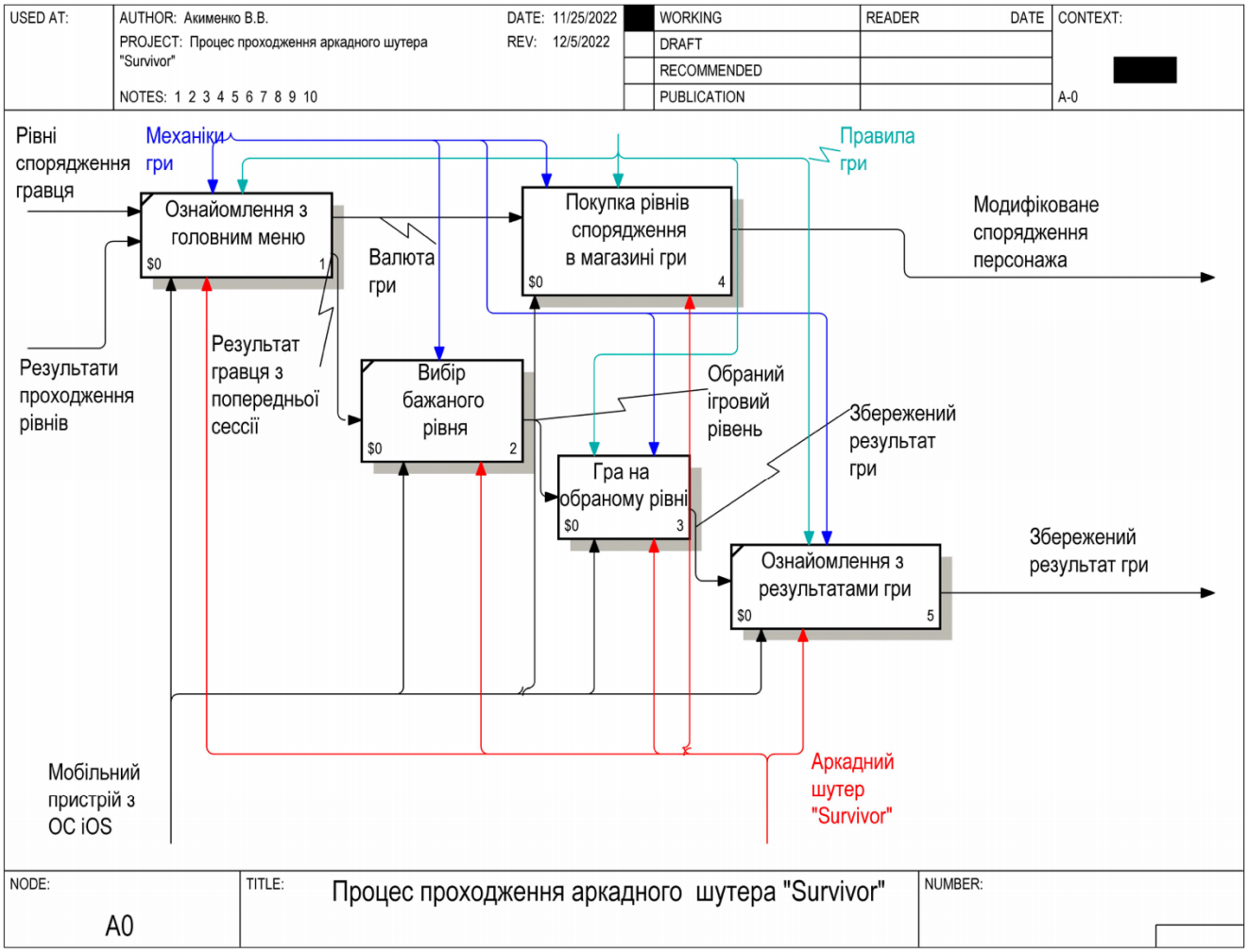


Рисунок 3.2 – Декомпозиція діаграми A-0

Для процесу «Покупка рівнів спорядження в магазині гри» було виконано декомпозицію. Процес був розбитий на наступні процеси: «Завантаження магазину гри», «Вибір конкретного спорядження», «Покупка нового рівня для спорядження». Декомпозицію процесу представлено на рисунку 3.3

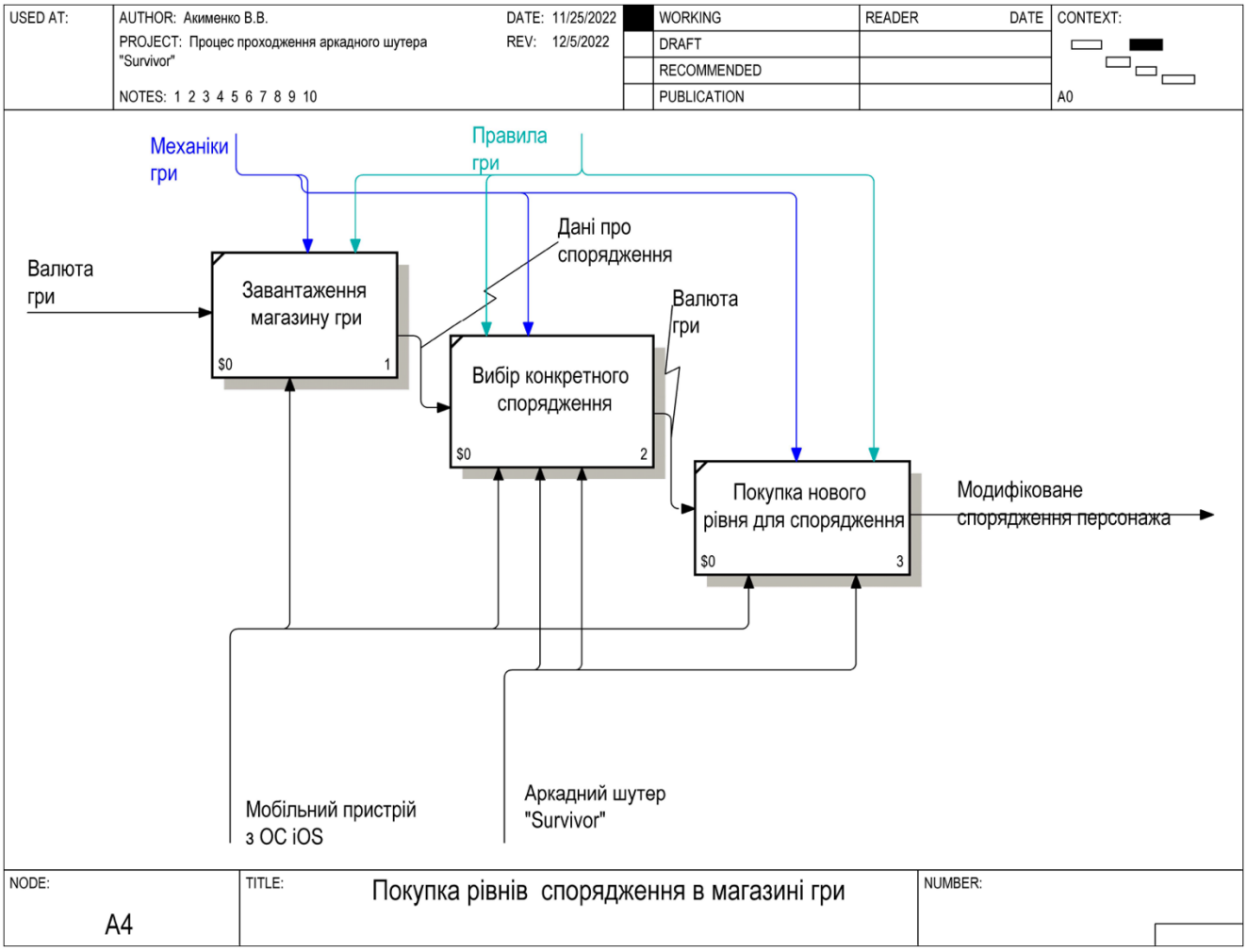


Рисунок 3.3 – Декомпозиція процесу «Покупка навичок в магазині гри»

Для процесу «Гра на обраному» було виконано декомпозицію. Процес був розбитий на 6 процесів: «Завантаження ігрової сцени», «Ініціалізація гравця», «Знищення ворогів», «Поліпшення характеристик», «Напад ворога», «Отримання летальної шкоди». Декомпозицію процесу представлено на рисунку 3.4

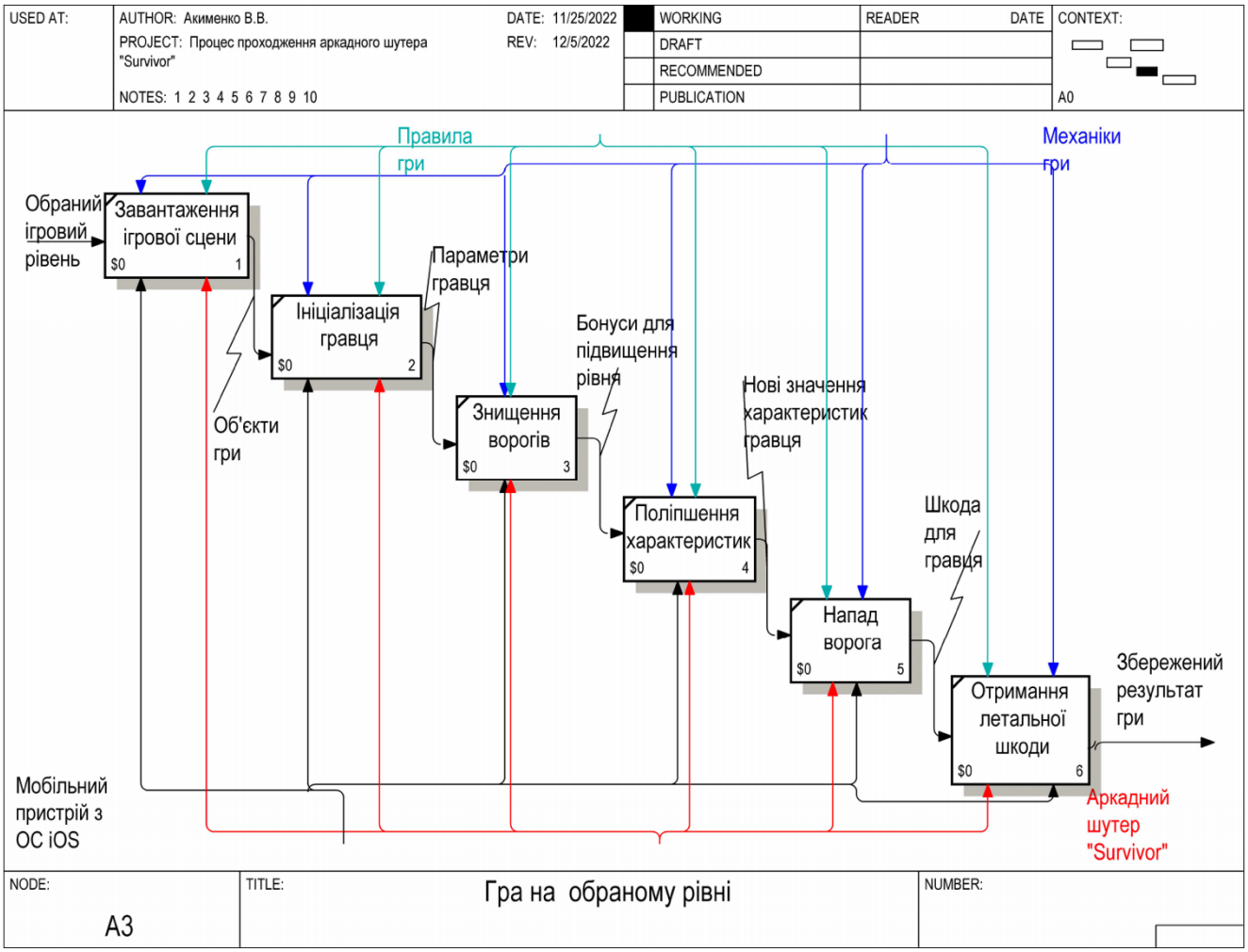


Рисунок 3.4 – Декомпозиція процесу «Гра на обраному рівні»

3.2 Моделювання діаграми варіантів використання

Для кращого розуміння функціонування ігрового додатку слід визначити можливі сценарії використання мобільного додатку.

UML – уніфікована мова моделювання для об’єктно-орієнтованого аналізу і проектування систем. Дана мова включає в себе різноманітні діаграми, проте для побудови варіантів використання буде використовуватися Use Case діаграма [21].

У якості акторів системи був виділений лише «Гравець». Основними операціями для актора «Гравець» є:

– вибір рівня гри;

– гра на обраному рівні;
– перегляд результатів гри;
– вибір налаштувань гри;
– перегляд магазину гри;
– покращення характеристик спорядження.

Діаграма варіантів використання в UML зображена на рисунку 3.5.

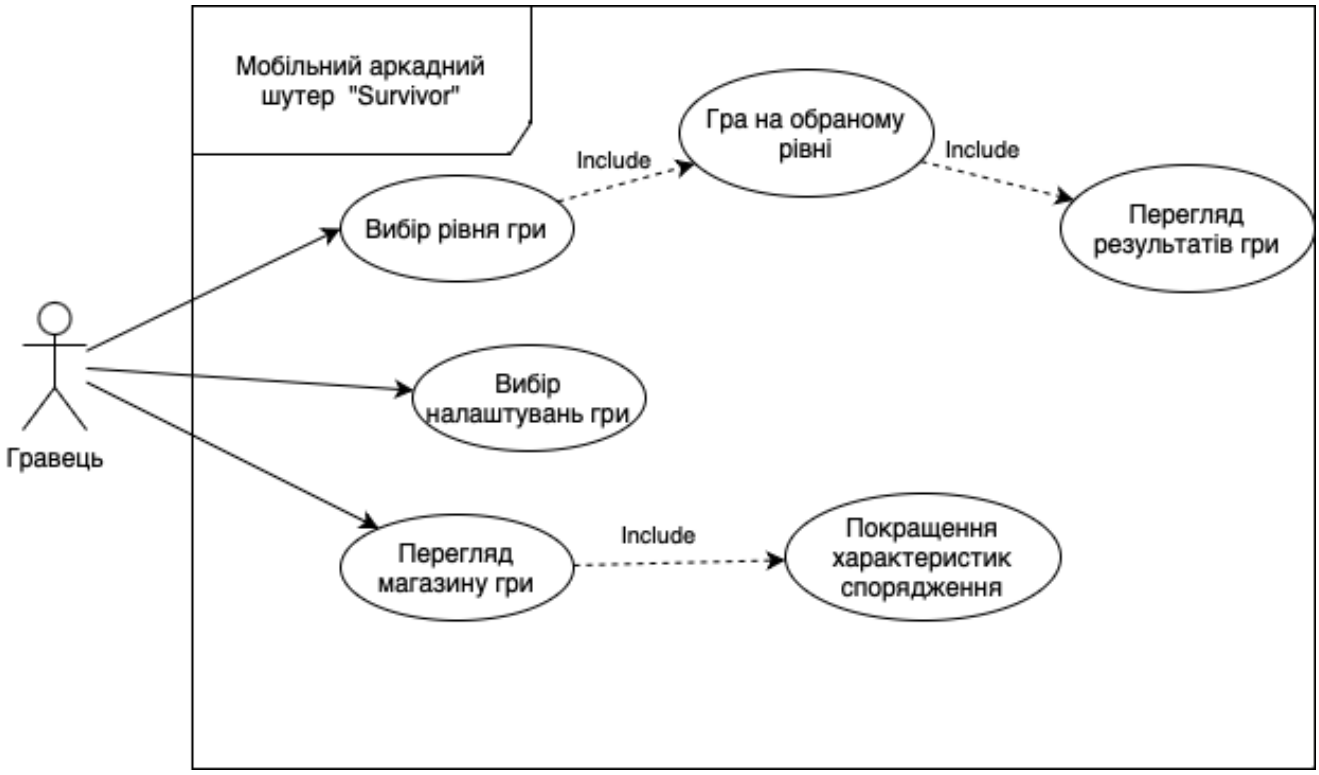


Рисунок 3.5 – Діаграма варіантів використання для гравця

4 РОЗРОБКА ІГРОВОГО МОБІЛЬНОГО ДОДАТКУ

4.1 Архітектура ігрового додатку

Архітектурний патерн MVC (Model-View-Controller) досить популярне рішення в багатьох мобільних додатках для ОС iOS [22]. Сам патерн складається з трьох компонентів: моделі(Model), виду(View), контролеру (Controller); і всі ці компоненти виконують свою роль [23]. Модель відповідає за зберігання даних, вид за відображення цих даних та в цілому за елементи інтерфейсу, що з'являються на екрані, в той час як контролер виступає проміжною ланкою між попередніми елементами, оновлює дані та передає їх між компонентами. В цілому такий підхід дозволяє побудувати досить просто та чітку архітектуру в додатку, що надає змогу легко та швидко будувати та додавати нову модулі без втручання в нові компоненти системи. Легке написання коду значною мірою спрощує майбутнє тестування, дозволяє більш швидше виконувати підтримку та оновлення коду відповідно да потреб розробника.

Рисунок 4.1 представляє собою архітектуру розроблюваного ігрового додатку.

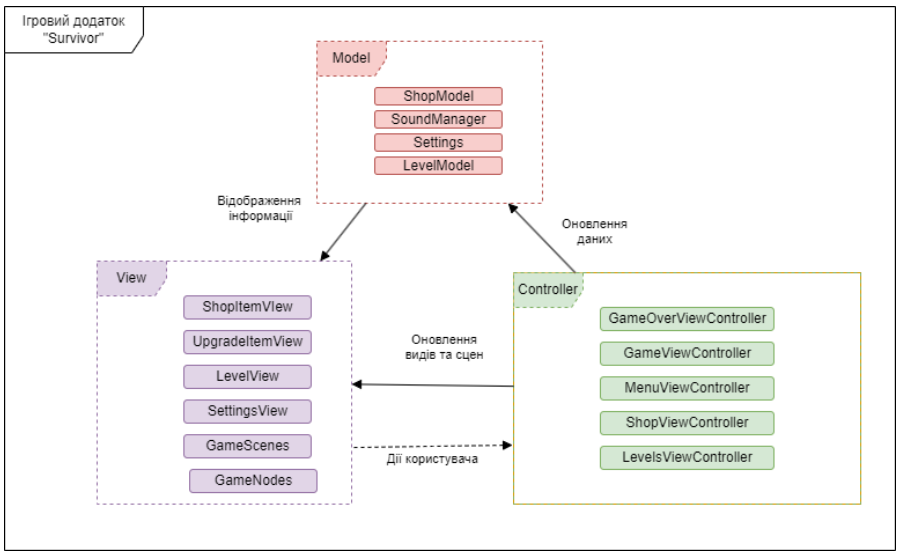


Рисунок 4.1 – Архітектура ігрового додатку

4.2 Програмна реалізація

В першу чергу необхідно створити за допомогою IDE XCode необхідно створити проект, попередньо обравши шаблон проекту у вигляді гри (рис 4.2).

В якості мови програмування необхідно вказати Swift, технологія для створення гри SpriteKit [24], а також обрати GameplayKit (рис. 4.3).

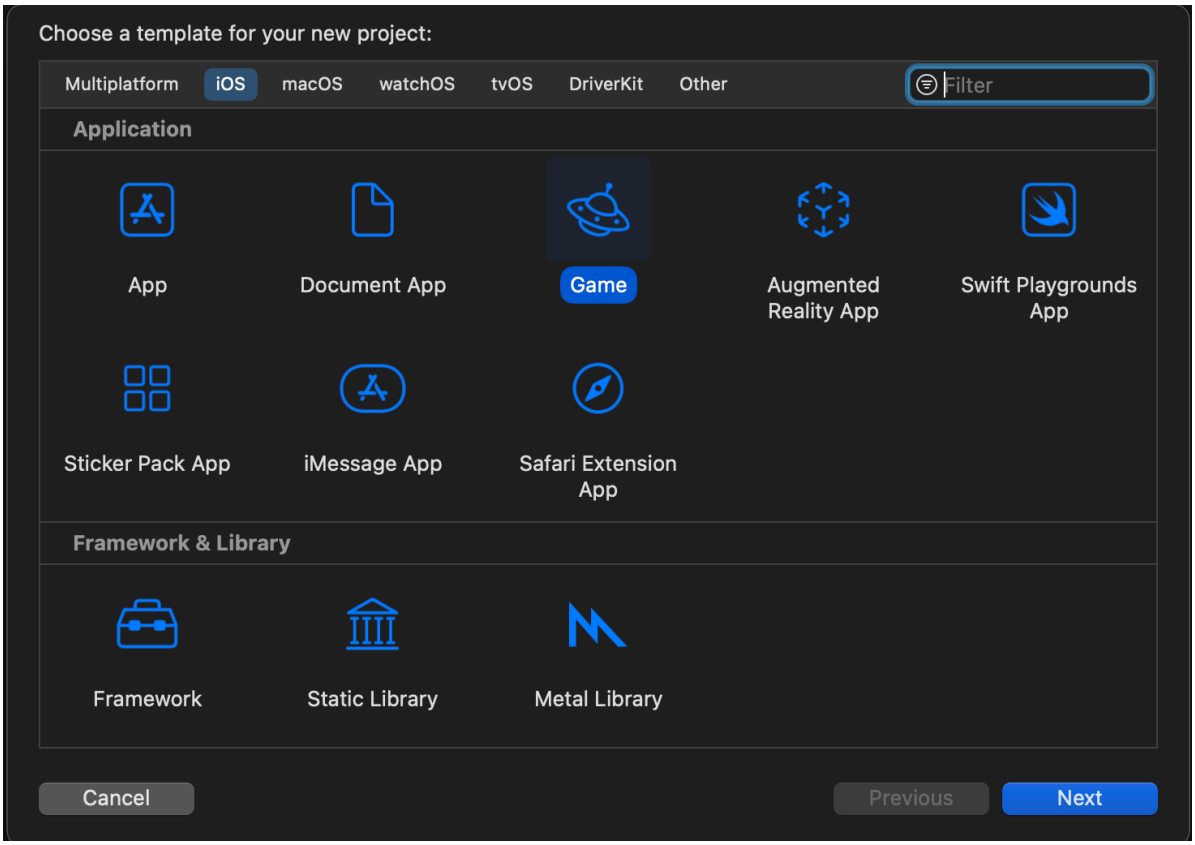


Рисунок 4.2 – Вибір шаблону гри

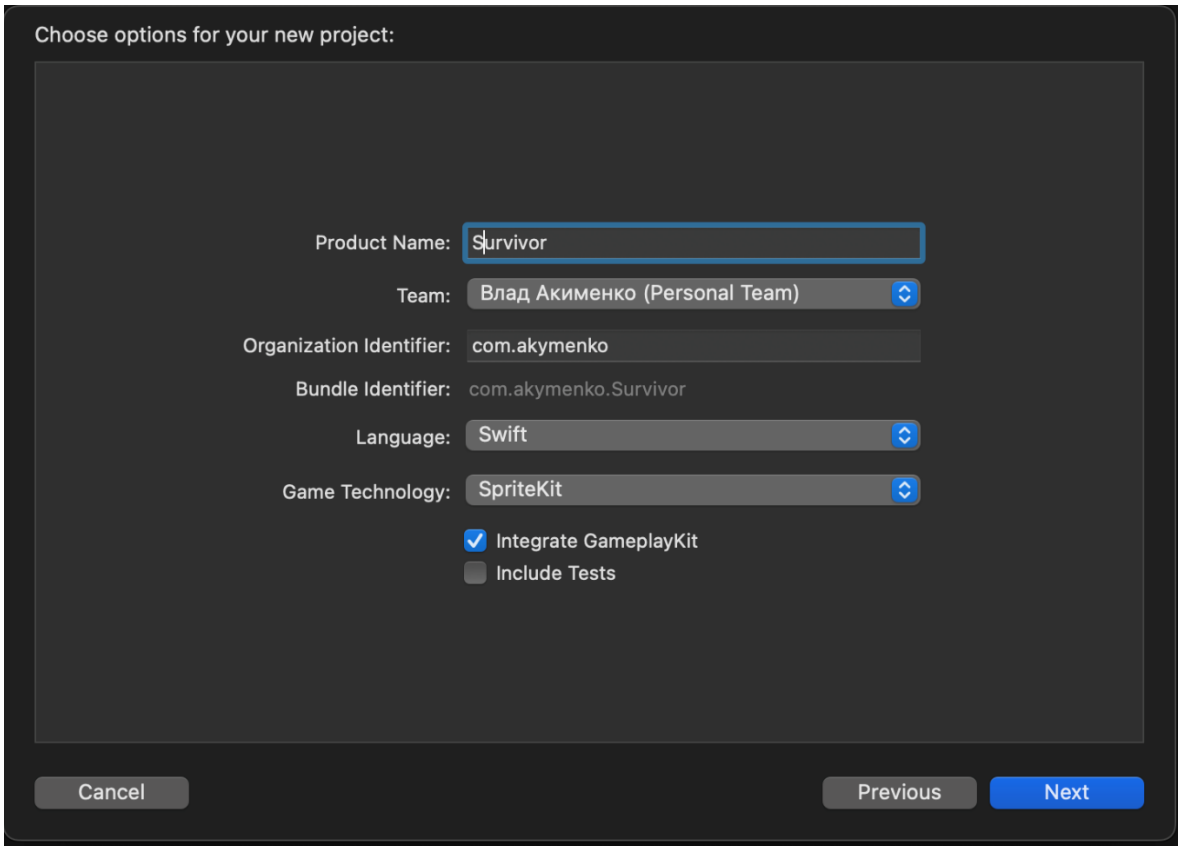


Рисунок 4.3 – Налаштування параметрів проекту

Окрім цього також необхідно налаштувати параметри проекту (рис. 4.4). Було обрано версію iOS 13.0 для розгортання проекту з метою розширення кількості можливих пристроїв, а також для тих користувачів, які з тих чи інших причин не оновлюють своє ПЗ. Було обрано портретну орієнтацію для ігрового додатку, пристроями для підтримки додатку є Iphone та Mac (пристрої були обрані за замовчуванням, з використанням збірки проекту на Iphone для обох випадків).

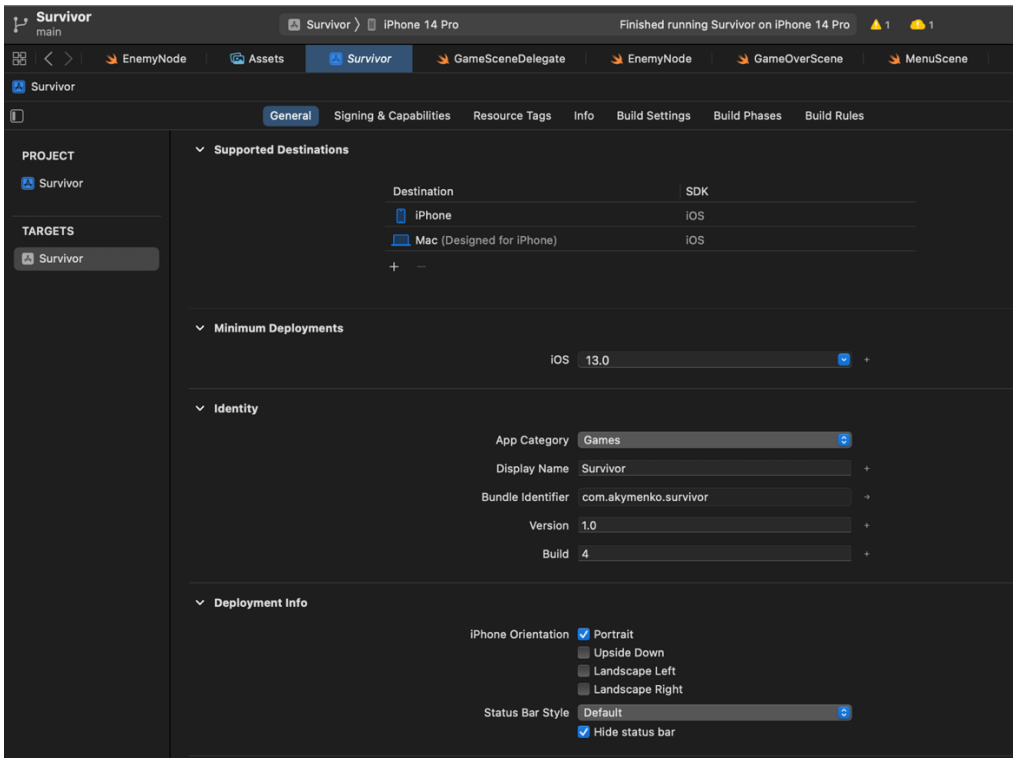


Рисунок 4.4 – Вибір параметрів для розгортання додатку

Оскільки для створення інтерфейсу додатку використовується фреймворк UIKit, Storyboards є однією із його складових. Налаштування екранів додатку та їх елементів було виконано в файлі Main.storyboard (рис. 4.5).

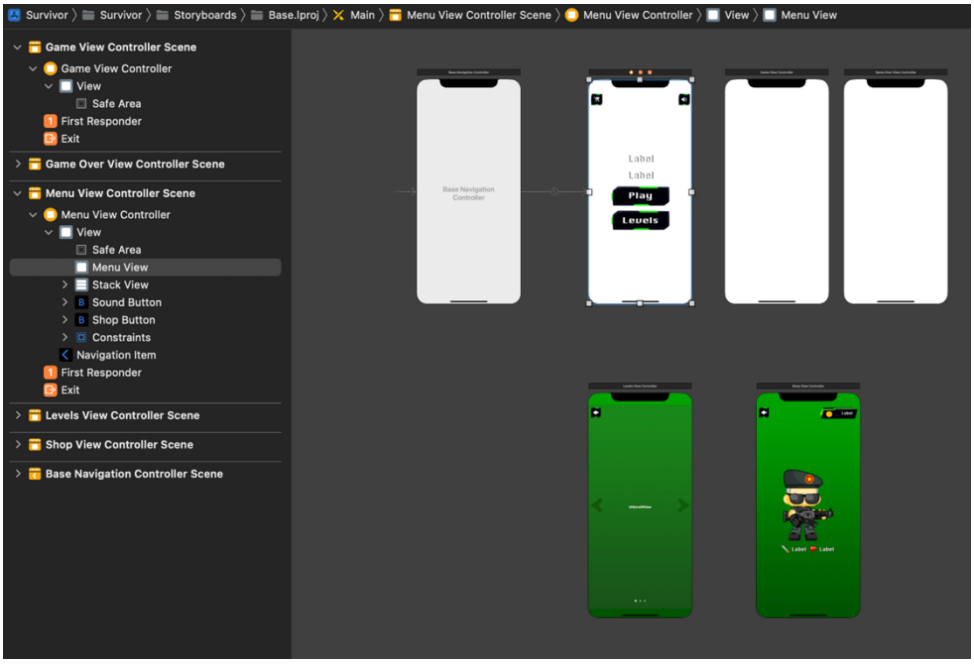


Рисунок 4.5 – Екрани ігрового додатку

Також окрім екранів гри було розроблено окремі види для спливаючих вікон та для окремих елементів інтерфейсу. Зокрема для кожного елементу були налаштовані constraints (обмеження), які встановлюють параметри для розміру та розташування елементів для коректного розташування та масштабування елементів на різних пристроях з різним розміром екрану(рис. 4.6).

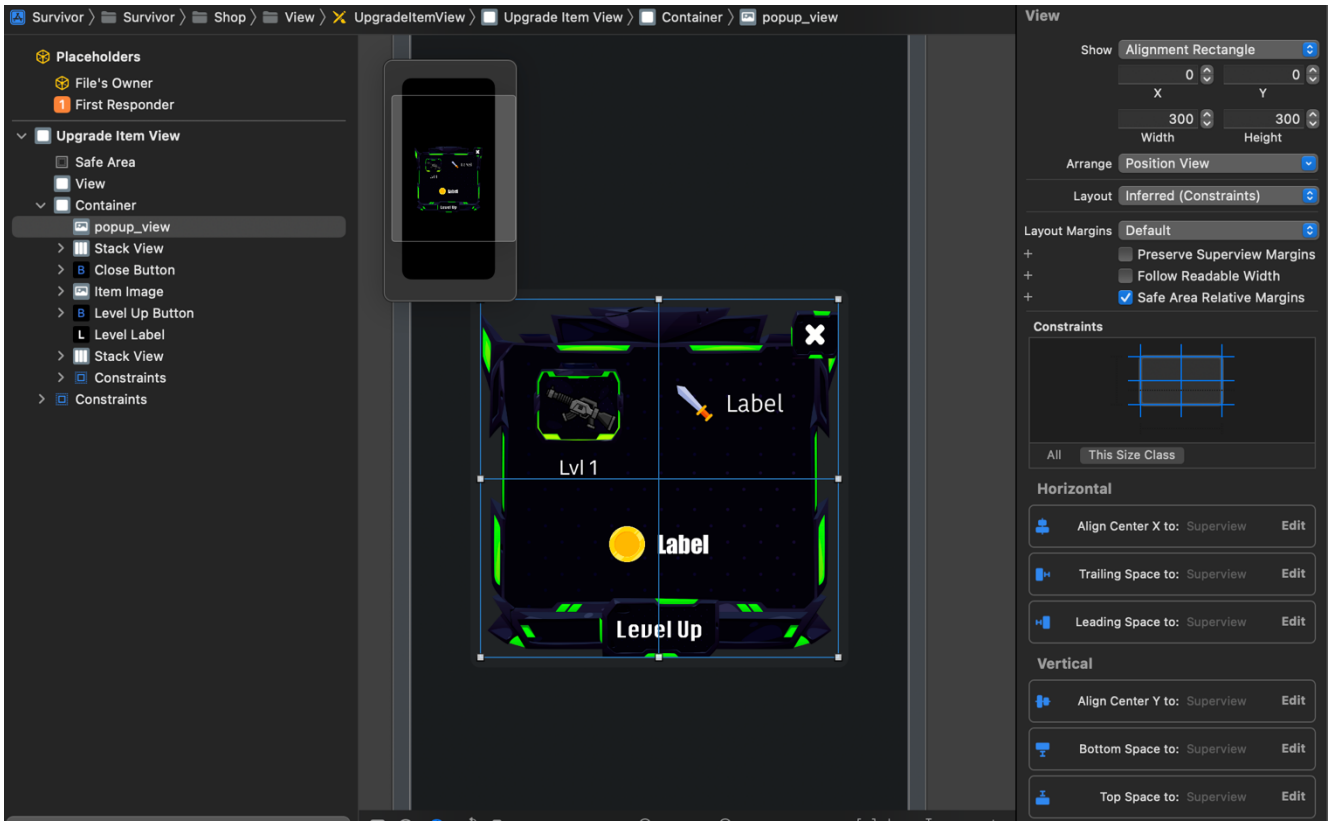
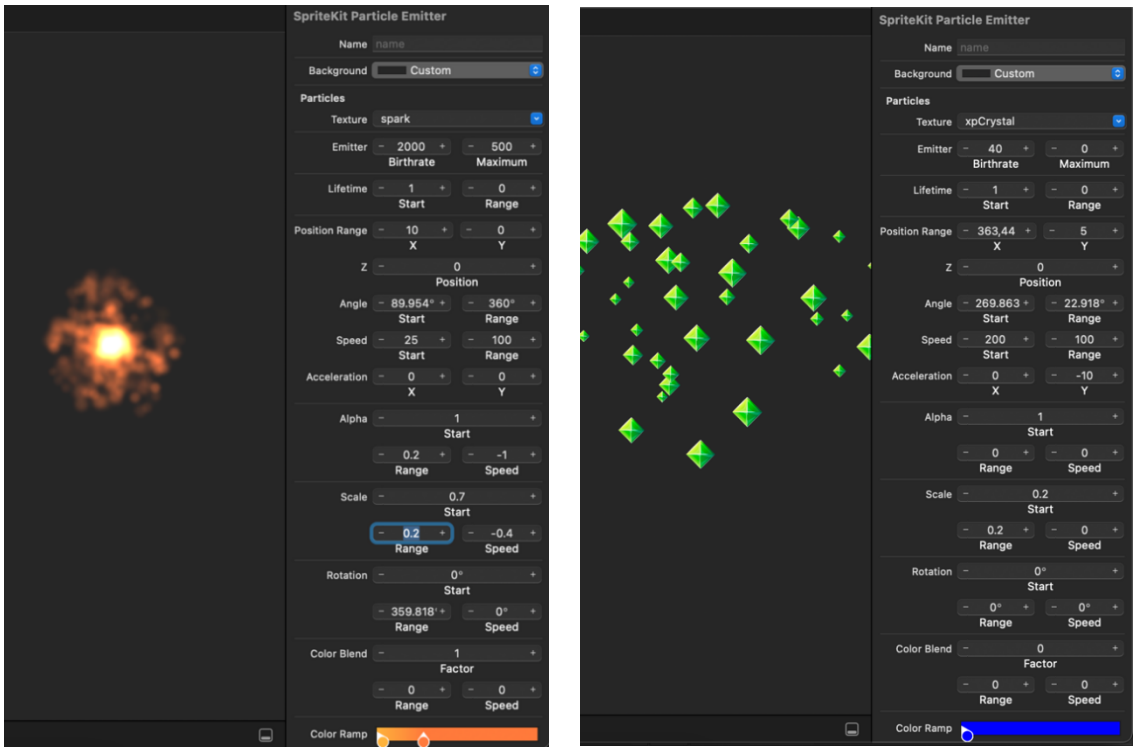


Рисунок 4.6 – Налаштування constraints для виду UpgradeItemView

Для створення візуальних ефектів, таких як вибух, поранення гравця, тощо було створено частинки (particles), які дозволяють виконати налаштування параметрів швидкості, розміру, кількості елементів, тощо.

Приклад налаштування ефектів зображено на рисунку 4.7.



а б

Рисунок 4.7 – Параметри particle для ефекту вибуху (а) та отримання рівня (б)

Загальна структура файлів проекту для ігрового додатку представлена на рисунку 4.8.

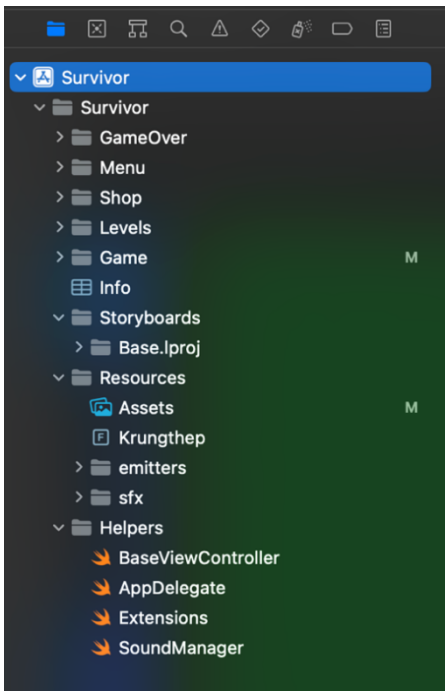


Рисунок 4.8 – Структура файлів проекту

В таблиці 4.1 наведено основні розроблені класи ігрового додатку та їх призначення

Таблиця 4.1 – Класи ігрового додатку

Клас	Призначення
GameOverViewController	Відображення фінального результату гри
GameViewController	Основний екран з грою
MenuViewController	Меню гри
ShopViewController	Магазин для покращення спорядження персонажа
LevelsViewController	Екран вибору рівнів
SoundManager	Клас для роботи зі звуками
ShopModel	Модель для збереження валюти гри та рівнів спорядження
Settings	Налаштування гри
LevelModel	Модель для збереження найкращого результату на кожному з рівнів
Player	Клас для обробки поведінки гравця
EnemyNode	Клас для обробки поведінки ворога
LevelXpNode	Кристали, що дозволяють поліпшувати рівень зброї гравця під час гри
CoinNode	Монети, які може збирати гравець
Crate	Ящик, який містить випадковий бонус
HealthNode	Бонус, який відновлює шкалу здоров’я гравця
MagnetNode	Бонус, що збирає всі кристали на рівні
Hud	Відображення інтерфейсу під час гри
LevelUpgradeHud	Інтерфейс, що відповідає за вибір покращення для зброї гравця
Joystick	Елемент управління гравцем
Camera	Камера

В таблиці 4.2 наведено види ігрового додатку та їх призначення

Таблиця 4.2 – Вид ігрового додатку

Вид	Призначення
ShopItemView	Відображення інформації про спорядження
UpgradeItemView	Вікно для покращення спорядження
LevelView	Вид, який відображає рівень гри
SettingsView	Вікно налаштувань

В таблиці 4.3 наведено сцени ігрового додатку та їх призначення

Таблиця 4.3 – Сцени ігрового додатку

Сцена	Призначення
GameScene	Основна ігрова сцена
GameOverScene	Сцена для відображення анімацій та результату гри
MenuScene	Фонова сцена в меню гри

Усі коди ігрового додатку наведені нижче в додатку Б.

4.3 Використання ігрового додатку

Після запуску гри на екрані з’являється меню гри (рис. 4.9). Меню відображає поточний обраний рівень, найкращий результат (найдовший час та кількість знищених ворогів). Крім цього, у гравця є можливість почати гру на поточному

рівні, змінити рівень гри, відкрити меню з налаштуваннями, а також перейти в магазин.

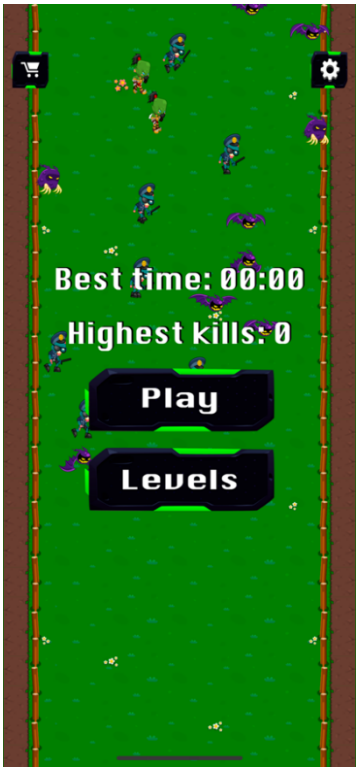


Рисунок 4.9 – Меню гри

Якщо гравець натиснув кнопку налаштувань, то з’являється меню з можливістю увімкнути/вимкнути звуки, музику або вібрацію телефону під час гри (рис. 4.10).

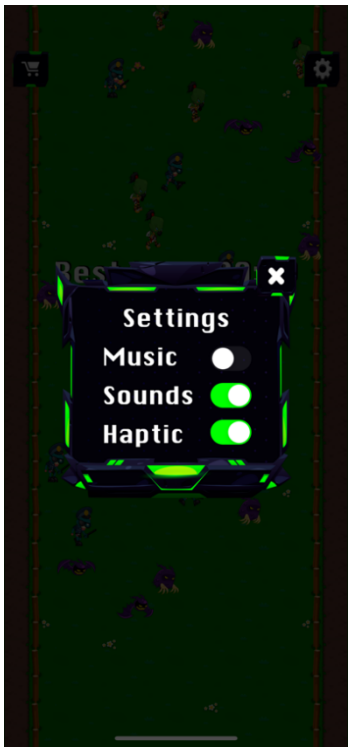


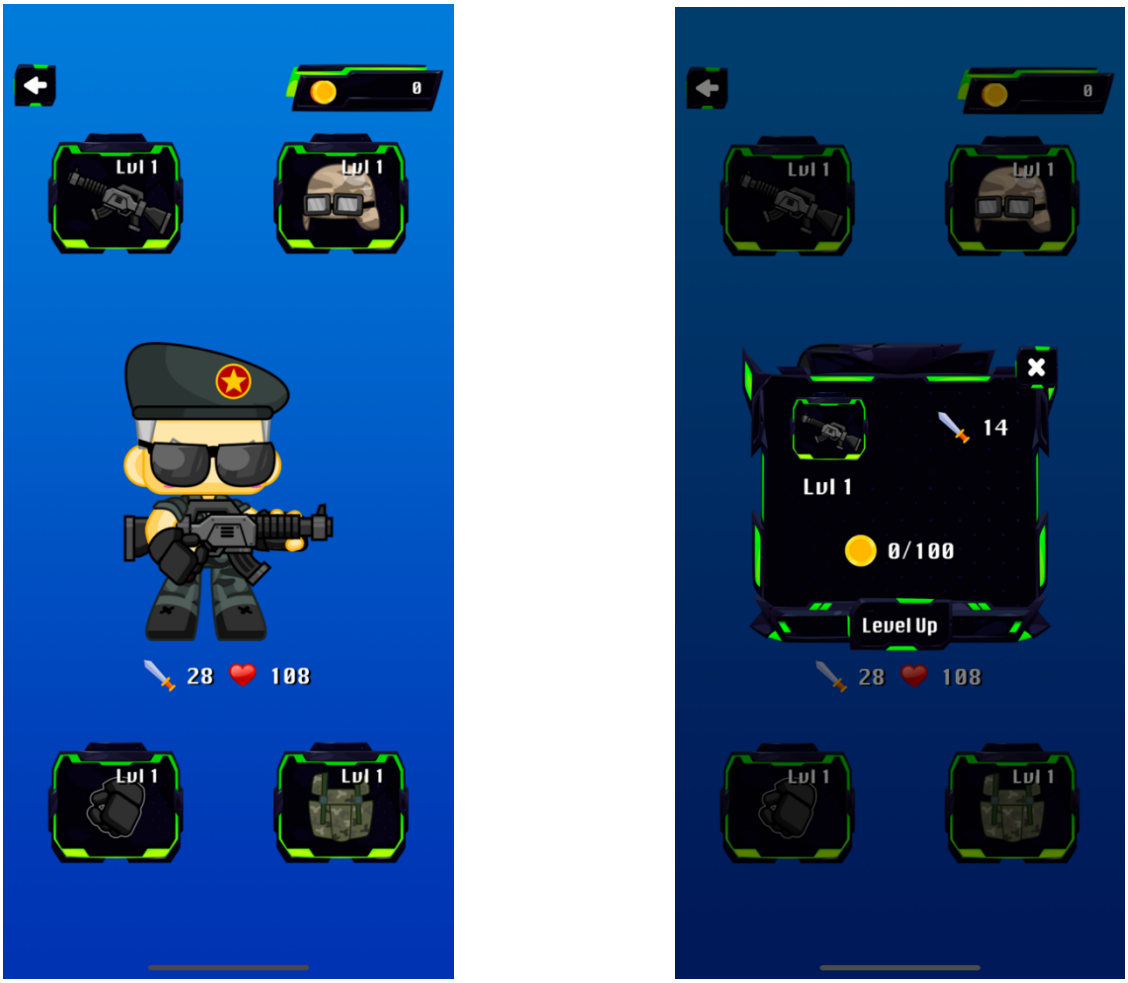
Рисунок 4.10 – Вікно налаштувань

При натисканні кнопки вибору рівня, гравець можна обрати один із чотирьох доступних рівнів (рис. 4.11). При цьому при зміні рівня на екрані меню буде відображатися обраний рівень та найкращий результат по цьому рівню.

а б в г

Рисунок 4.11 – Рівні гри: фабрика (а), селище (б), гори (в), замок (г)

У випадку, якщо гравець зайшов у магазин гри, на екрані буде відображено поточну кількість валюти в розпорядженні гравця, а також буде показано окремо рівень кожного спорядження гравця, а також загальні характеристики (сила шкоди та максимальний рівень здоров'я) (рис. 4.12 а). Окрім цього, якщо гравець натисне на якесь спорядження, то з'явиться відповідне вікно з можливістю перегляду поточної інформації про характеристику спорядження (рис. 4.12 б). За наявності достатньої кількості ігрової валюти гравець має можливість покращити спорядження, тим самим підвищивши загальні характеристики.



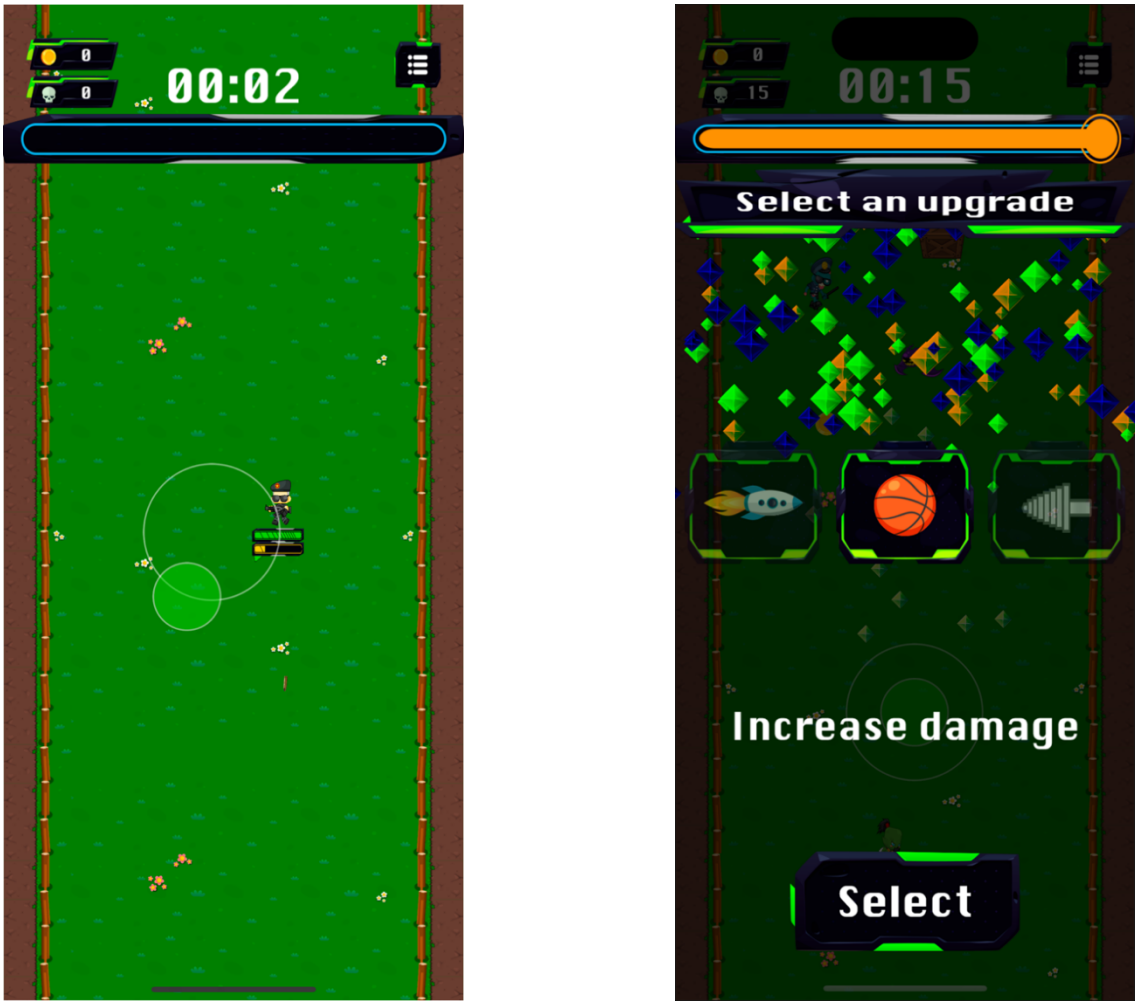
а б

Рисунок 4.12 – Спорядження гравця (а); покращення спорядження (б)

Якщо гравець натиснув кнопку "Play", то почнеться гра на обраному рівні. Метою гри є якомога довше протистояти ворогам, які постійно з'являються.

Керування гравцем відбувається за допомогою джойстика. При знищенні ворогів, є шанс підібрати кристали, які дають можливість заповнити шкалу досвіду. Коли ця шкала заповнена, гравець на вибір може обрати один з трьох запропонованих варіантів для збільшення потужності арсеналу. Окрім цього, на рівні інколи з'являється спеціальний ящик, що містить один з можливих бонусів: додаткову валюту, магніт, який збирає всі кристали на рівні, бонус, що частково відновлює шкалу здоров'я. У випадку, якщо шкала здоров'я гравця дорівнює нулю, то в такому випадку гравець переходить на екран результатів гри.

Ігровий процес зображено на рисунках 4.13-4.14.



а б

Рисунок 4.13 – Переміщення гравця локацією (а), вибір апгрейду для ігрової сесії (б)

а б

Рисунок 4.14 – Збір бонусу (а), знищення ворогів (б)

На екрані результатів гри відображається максимальний час, який вдалося протриматися гравцеві, а також кількість ворогів, що він зміг знищити (рис. 4.15). Якщо результат сесії перевершує найкращий час на цьому рівні, то відповідно зберігається новий рекорд рівня. Окрім цього, валюта гри, набрана під час сесії додається до поточної кількості. Гравець може почати обраний рівень спочатку, або повернутися до меню гри.

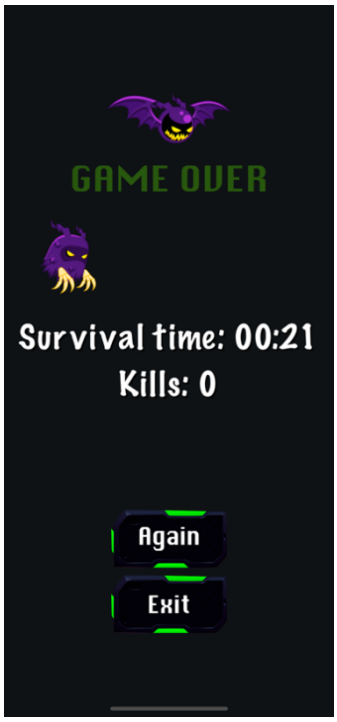


Рисунок 4.15 – Фінальний результат гри

Розроблений додаток реалізує весь запланований на етап постановки задачі перелік функціональних можливостей, що проілюстровано на рисунках підрозділу.

ВИСНОВКИ

Метою кваліфікаційної роботи є розробка мобільного аркадного шутера «Survivor» для ОС iOS для покращення тактичних, логічних навичок гравця.

Були проаналізовані актуальні джерела інформації по тематиці шутерів, що дозволяють стверджувати актуальність розроблюваного додатку, а також були визначені основні особливості та механіки шутерів.

Було сформовано мету роботи, задачі, визначені засоби реалізації поставленої задачі. У якості засобів реалізації було обрано мову програмування Swift, фреймворк для створення ігор SpriteKit, фреймворк UIKit для створення інтерфейсу, IDE XCode.

Протягом етапу моделювання додатку було створено діаграму нотації IDEF0 для відображення функціонування системи, виконано декомпозицію головного процесу на підпроцеси. Розроблена діаграма варіантів використання дозволила краще зрозуміти необхідний функціонал ігрового додатку.

Основні етапи розроблення проекту представлені в пояснювальній записці. Було описано проект на фазах ініціалізації та розробки. Під час фази розробки було розроблено структури WBS та OBS.

У результаті було створено мобільний аркадний шутер «Survivor». Були розроблені основні екрани додатку, налаштовані обмеження для позиції та розміру елементів інтерфейсу, розроблено візуальні ефекти, налаштовано ігрові та фонові сцени додатку. Були створені спрайти, налаштовано ігрова логіка та основні ігрові механіки.

Розроблений ігровий додаток може бути доданий в майбутньому в магазин додатків AppStore.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Number of smartphone subscriptions worldwide from 2016 to 2021, with forecasts from 2022 to 2027 [Електронний ресурс] // Statista Research Department. – 2022. – Режим доступу до ресурсу: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. (дата звернення: 02.10.2022);

2. Digi-capital projects worldwide game revenues to hit $200b by 2023 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.gamesindustry.biz/articles/2019-03-22-digi-capital-projectsworldwide-game-revenues-to-hit-usd200b-by-2023.

3. Best of Play 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/editorial_collection/promotion_topic_bestof2019_xfn_hub?pli=1;

4. Game design: next level: підручник. Gingko Press Inc., 2019. 240 с

5. Gregory J. Game Engine Architecture, Second Edition / Jason Gregory., 2018. – 272 с.

6. C. W. Totten. Level design: processes and experiences: підручник. A K Peters/CRC Press; 1st Edition, 2016. 408 с.

7. Hotline Miami в Steam: веб-сайт. URL: https://store.steampowered.com/app/219150/Hotline_Miami/?l=russian

8. 20 Minutes Till Dawn в Steam: веб-сайт URL: https://store.steampowered.com/app/1966900/20_Minutes_Till_Dawn/

9. Brawl Stars в App Store: веб-сайт URL: https://apps.apple.com/us/app/brawl-stars/id1229016807

10. Blokdyk G. IDEF. 3rd ed. 2019. 280 с.

11. Unhelkar B. Software Engineering with UML. Auerbach Publications, 2020. 426 p.

12. Rumpe B. Modeling with UML. Springer Cham, 2016. 281 p.

13. Farook F. UIKit Apprentice / Fahim Farook., 2021. – 827 с.

14. Swift Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://www.swift.org/about/.

15. Morrow B. Swift Apprentice / Ben Morrow., 2021. – 575 с.

16. Fix R. Expert Swift / Ray Fix., 2021. – 461 с.

17. Swift vs. Objective-C: A Look at iOS Programming Languages [Електронний ресурс] – Режим доступу до ресурсу: https://www.upwork.com/resources/swift-vs-objective-c-a-look-at-ios-programming-languages.

18. Spritekit [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/spritekit/.

19. Gauchat J. UIKit for Masterminds / John Gauchat.. – 1231 с.

20. Моделювання та аналіз систем. IDEF-технології. Підручник-практикум / 91 Черемних С.В., Семенов І.О., Ручкін В.С., 2006. 188 с.

21. Gomma H. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures. Cambridge University Press, 2014. 578 p.

22. Laso-Marsetti F. Model-View-Controller (MVC) in iOS – A Modern Approach [Електронний ресурс] / Felipe Laso-Marsetti. – 2019. – Режим доступу до ресурсу: https://www.kodeco.com/1000705-model-view-controller-mvc-in-ios-a-modern-approach#toc-anchor-005.

23. Strawn J. Design Patterns by Tutorials / J. Strawn, J. Greene., 2019. – 364 с.

24. Begbie C. 2D Apple Games by Tutorials. 2nd ed. 2016. 671 p.

ДОДАТОК А

ПЛАНУВАННЯ РОБІТ

для розробки кваліфікаційної роботи магістра

«Мобільний аркадний шутер «Survivor» для ОС iOS»

А.1 ІДЕНТИФІКАЦІЯ МЕТИ ІТ-ПРОЕКТУ

Метою проекту є розробка аркадного шутера «Survivor» для ОС iOS.

Деталізація мети методом SMART:

S – Метою проекту є розробка аркадного шутера «Survivor» для ОС iOS для цікавого проводження вільного часу.

M – Ігровий додаток має чіткий набір функцій, властивостей та задач для реалізації проекту.

A – Розробка ігрового додатку можлива за наявності засобів для розробки та ігрових механік.

R – Розробка додатку буде виконуватись на основі існуючих ігрових механік.

T – На розробку такого проекту потрібно – 6 тижнів, враховуючи, що в середньому буде використовуватись 6 годин в день.

А.2 ПЛАНУВАННЯ ЗМІСТУ СТРУКТУРИ РОБІТ ІНФОРМАЦІЙНОЇ СИСТЕМИ

WBS (рис. А.2.1) є ієрархічною структурою, що показує подальший розподіл проекту на фази, кінцеві результати і пакети робіт. Інструмент спрямований на детальне планування, оцінку вартості, визначення та розподіл персональної відповідальності виконавців та інші - тобто, на основні роботи і результати, що визначають зміст проекту.

Рисунок А.2.1 – WBS діаграма

Після того, як була створена WBS структура проекту, наступним необхідним етапом є розробка OBS (рис.А.3.2). Організаційна структура представляє собою графічне відображення учасників проекту та їх відповідальних осіб, які задіяні в реалізації проекту.

Рисунок А.2.2 – OBS діаграма

А.3 ПОБУДОВА КАЛЕНДАРНОГО ГРАФІКУ ВИКОНАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Діаграма Ганта – це горизонтальна діаграма з тимчасової шкалою, яка використовується для ілюстрації плану робіт за проектом з прив'язкою до часу.

За допомогою діаграм Ганта керівники проектів і менеджери по продукту розбивають проекти на робочі завдання для зручності управління, підтримують порядок в роботі і роблять залежності між завданнями наочними.

Діаграми Ганта дозволяють спростити складові проекти. За допомогою цього засобу можна досить наочно і зручно для узагальнення представити велику кількість даних. Завдяки цій гістограмі велика кількість зацікавлених осіб, команд або їх учасників не стане проблемою для запису завдань, як і часті зміни обсягу роботи.

Розглянемо створену діаграму Ганта до заданої інформаційної системи (рис.А.2.3).

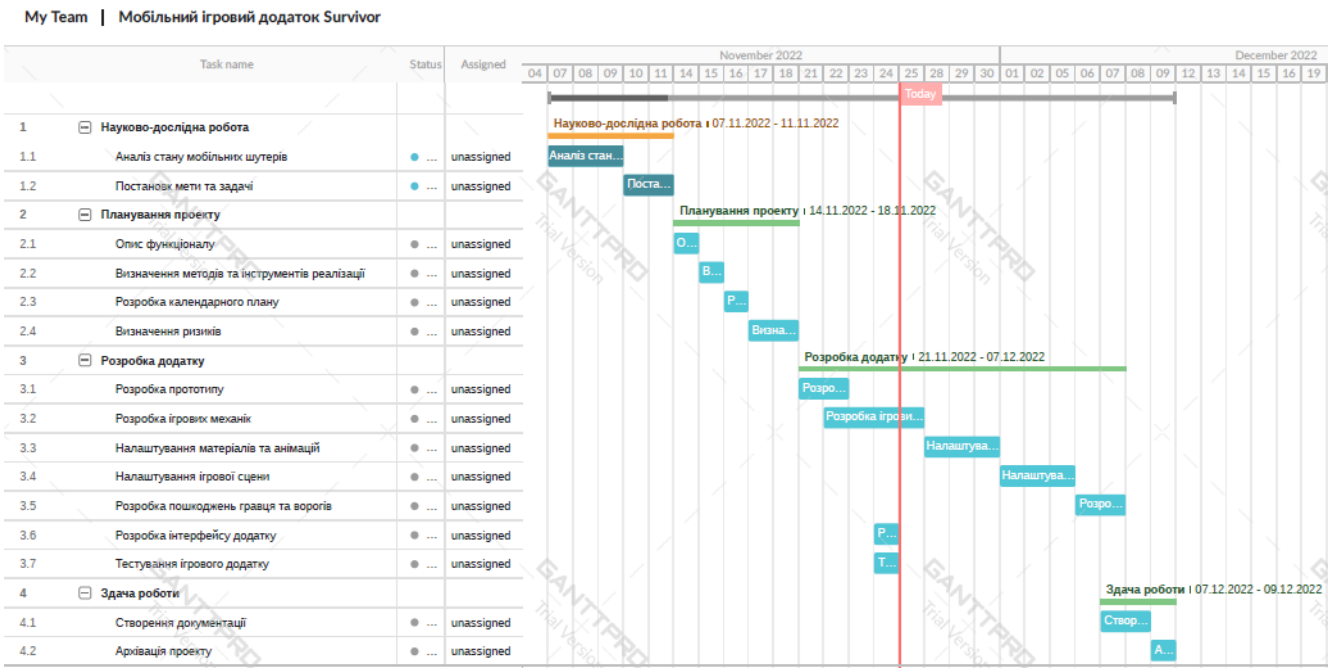


Рисунок А.2.3 – Діаграма Ганта(1)

А.4 ПЛАНУВАННЯ РИЗИКІВ ПРОЕКТУ

Ризиком є ймовірна подія, яка у випадку її виникнення може як негативно, так і позитивно вплинути на конкретний проект. Управління ризиком – це процес зміни ризиків та реагування на події під час виконання проекту. При реалізації проекту важливою частиною є моніторинг ризиків. Отже, в даному випадку можна виділити деякі ризики.

Класифікація ризиків:

1. За імовірністю виникнення:

– слабо ймовірнісні;
– мало ймовірнісні;
– імовірні;
– досить імовірні;
– майже імовірні.

2. За величиною втрат:

– мінімальна;
– низька;
– середня;
– висока;
– максимальна.

На основі цієї інформації була проведена класифікація ризиків для даного проекту, що показана в таблиці А.4.1

Таблиця А.4.1 – Класифікація ризиків

N	Назва ризику	Ймовірність	Величина втрат
1	Розширення обсягу робіт	3	3
2	Відставання від графіку	2	3

N	Назва ризику	Ймовірність	Величина втрат
3	Проблеми з технікою	1	2
4	Пошкодження файлів	2	5
5	Збої в роботі ПЗ	3	5
6	Помилки в роботі додатку	2	2
7	Непередбачувані обставини	5	5

На основі класифікації ризиків була складена матриця ризиків проекту, що представлена в таблиці А.4.2.

Таблиця А.4.2 – Матриця ризиків

	Величина втрат				
Ймовірність					R7
			R1		R5
		R6	R2		R4
		R3			

Після оцінки ризиків було складено план реагування на ризики (табл. А.4.3).

Таблиця А.4.3 – Реакція на ризики

Ризики проекту	Реакція на ризик
Розширення обсягу робіт	При плануванні графіку робіт варто додати додатковий час враховуючи можливі труднощі
Відставання від графіку	
Проблеми з технікою	Віднести техніку на ремонт, подбати про запасну техніку на час ремонту, щоб не переставати виконувати роботу
Пошкодження файлів	Робити запасні копії в хмарному сховищі
Збої в роботі ПЗ	Перевстановити програму, провести діагностику на пошкоджені файли
Помилки в роботі додатку	Перевантажити додаток, робити частіше збереження даних
Непередбачувані обставини	Подбати про доступ в інтернет, наявність альтернативних шляхів живлення техніки

ДОДАТОК Б

MenuViewController.swift:

```
import UIKit
import SpriteKit

class MenuViewController: BaseViewController {

    @IBOutlet weak var soundButton: UIButton!
    @IBOutlet weak var startButton: UIButton!
    @IBOutlet weak var levelsButton: UIButton!
    @IBOutlet weak var menuView: SKView!
    @IBOutlet weak var bestTimeLabel: UILabel!
    @IBOutlet weak var highestKillsLabel: UILabel!
    @IBOutlet weak var shopButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()
        startButton.addTarget(self, action: #selector(startButtonTapped), for:
.touchUpInside)
        levelsButton.addTarget(self, action: #selector(levelsButtonTapped), for:
.touchUpInside)
        soundButton.addTarget(self, action: #selector(toggleMute(_:)), for:
.touchUpInside)
        shopButton.addTarget(self, action: #selector(shopButtonTapped), for:
.touchUpInside)

        if !SoundManager.sharedInstance.isMuted {
            SoundManager.sharedInstance.startPlaying()
        }

        addMenuScene()
    }

    @objc func toggleMute(_ sender: Any) {
        let settingsView = SettingsView(frame: .init(origin: .zero, size:
view.frame.size))
        view.addSubview(settingsView)
        settingsView.animateIn()
        if Settings.shared.isAllowedVibration {
            Vibration.medium.vibrate()
        }
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        if let view = menuView, let scene = view.scene as? MenuScene {
            scene.changeBGtexture(with: Level.shared.getBGforLevel(Level.shared.level))
        }
        let time = Level.shared.getTime(for: Level.shared.level)
        let kills = Level.shared.getKills(for: Level.shared.level)
        bestTimeLabel.text = "Best time: \(Level.shared.stringFromTimeInterval(interval:
time))"
        highestKillsLabel.text = "Highest kills: \(kills)"
    }


    func addMenuScene() {
        let sceneNode = MenuScene(size: view.frame.size)
```

```
        sceneNode.scaleMode = .aspectFill

        if let view = menuView {
            view.presentScene(sceneNode)

//            view.showsFPS = true
//            view.showsNodeCount = true
//            view.showsPhysics = true
            view.ignoresSiblingOrder = true
        }
    }

    @objc func shopButtonTapped() {
        let storyboard  = UIStoryboard.init(name: "Main", bundle: nil)
        let shopVC = storyboard.instantiateViewController(withIdentifier:
"ShopViewController") as! ShopViewController
        navigationController?.pushViewController(shopVC, animated: true)
    }

    @objc func startButtonTapped() {
        let storyboard  = UIStoryboard.init(name: "Main", bundle: nil)
        let levelsVC = storyboard.instantiateViewController(withIdentifier:
"GameViewController") as! GameViewController
        navigationController?.pushViewController(levelsVC, animated: true)
    }

    @objc func levelsButtonTapped() {
        let storyboard  = UIStoryboard.init(name: "Main", bundle: nil)
        let levelsVC = storyboard.instantiateViewController(withIdentifier:
"LevelsViewController") as! LevelsViewController
        navigationController?.pushViewController(levelsVC, animated: true)
    }
}
```

GameViewController.swift:

```
import UIKit
import SpriteKit
import GameplayKit

class GameViewController: BaseViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        startGame()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        if Settings.shared.isAllowedSFX {
            SoundManager.sharedInstance.startPlaying()
        }
    }

    func startGame() {

        let sceneNode = GameScene(size: view.frame.size)
```

```
        sceneNode.scaleMode = .aspectFill
        sceneNode.gameProtocol = self

        if let view = self.view as! SKView? {
            view.presentScene(sceneNode)

//            view.showsFPS = true
//            view.showsNodeCount = true
//            view.showsPhysics = true
            view.ignoresSiblingOrder = true
        }

    }

}
extension GameViewController: GameProtocol {

    func restartGame() {
        startGame()
    }

    func gameOver(kill: Int, time: TimeInterval, coins: Int) {
        let storyboard  = UIStoryboard.init(name: "Main", bundle: nil)
        let gameOverVC = storyboard.instantiateViewController(withIdentifier:
"GameOverViewController") as! GameOverViewController
        gameOverVC.gameProtocol = self
        gameOverVC.kills = kill
        gameOverVC.coins = coins
        gameOverVC.time = time
        gameOverVC.modalTransitionStyle = .coverVertical
        navigationController?.pushViewController(gameOverVC, animated: true)
    }


    func backButtonTapped() {
        self.navigationController?.popViewController(animated: true)
    }

}
```

GameOverViewController.swift:

```
import UIKit
import SpriteKit

class GameOverViewController: BaseViewController {

    @IBOutlet weak var batImage: UIImageView!
    @IBOutlet weak var killsLabel: UILabel!
    @IBOutlet weak var survivalTimeLabel: UILabel!
    @IBOutlet weak var nextLevelButton: UIButton!
    @IBOutlet weak var ghostImage: UIImageView!
    @IBOutlet weak var gameOverView: SKView!
    @IBOutlet weak var gameOverLabel: UILabel!
    @IBOutlet weak var menuButton: UIButton!

    var kills: Int?
    var time: TimeInterval?
    var coins: Int?

    let level = Level.shared
    weak var gameProtocol: GameProtocol?
```

```
    override func viewDidLoad() {
        super.viewDidLoad()

        menuButton.addTarget(self, action: #selector(menuButtonTapped), for:
.touchUpInside)
        nextLevelButton.addTarget(self, action: #selector(levelButtonTapped), for:
.touchUpInside)

        guard let kills = kills, let time = time, let coins = coins else { return }
        Shop.shared.coins += coins
        killsLabel.text = "Kills: \(kills)"
        survivalTimeLabel.text = "Survival time: " +
Level.shared.stringFromTimeInterval(interval: time)
        if  level.getKills(for: level.level) < kills {
            level.setKills(for: level.level, kills: kills)
        }
        if  level.getTime(for: level.level) < time {
            level.setTime(for: level.level, time: time)
        }
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        gameOverLabel.pulsate()
        addGameOverScene()
    }

    func addGameOverScene() {
        let sceneNode = GameOverScene(size: view.frame.size)

        sceneNode.scaleMode = .aspectFill
        sceneNode.gameOverProtocol = self

        gameOverView.allowsTransparency = true
        gameOverView.showsFPS = true
        gameOverView.showsNodeCount = true
        gameOverView.showsPhysics = true
        gameOverView.presentScene(sceneNode)
        gameOverView.ignoresSiblingOrder = false
    }

    @objc func levelButtonTapped() {
        gameProtocol?.restartGame()
        navigationController?.popViewController(animated: true)
    }

    @objc func menuButtonTapped() {
        navigationController?.viewControllers.remove(at: 1)
        navigationController?.popViewController(animated: true)
    }

}

extension GameOverViewController: GameOverProtocol {
    func removeView() {
        gameOverView.isHidden = true
    }
}
```

ShopViewController.swift:

```
import UIKit

class ShopViewController: BaseViewController {

    @IBOutlet weak var backButton: UIButton!
    @IBOutlet weak var coinsLabel: UILabel!
    @IBOutlet weak var attackLabel: UILabel!
    @IBOutlet weak var healthLabel: UILabel!
    @IBOutlet weak var playerImageView: UIImageView!

    var upgradeView: UpgradeItemView!

    var items: [ShopItemView] = []
    var itemPosition: [(x:CGFloat,y:CGFloat)] = [(0.1,0.125),(0.1,0.75),(0.6,
0.125),(0.6,0.75)]

    override func viewDidLoad() {
        super.viewDidLoad()

        backButton.addTarget(self, action: #selector(backButtonTapped), for:
.touchUpInside)

        coinsLabel.text = Shop.shared.formatNumber(Shop.shared.coins)

        createItemViews()
        addItems()

        attackLabel.text = "\(Shop.shared.items[0].stat + Shop.shared.items[1].stat)"
        healthLabel.text = "\(Shop.shared.items[2].stat + Shop.shared.items[3].stat)"
    }

    func createItemViews() {
        for i in 0..<4 {
            guard let item:ShopItemView = ShopItemView.instanceFromNib() else { return }

            item.levelItemLabel.text = "Lvl \(Shop.shared.getLvl(for: i))"
            item.shopItemImage.image = .init(named: Shop.shared.getItemImage(for: i))
            item.levelItemLabel.font = UIFont(name: GameFont, size: view.frame.width *
0.04)

            let tapGesture: UITapGestureRecognizer = .init(target: self,
action:#selector(showUpgradeView))
            tapGesture.numberOfTapsRequired = 1
            tapGesture.numberOfTouchesRequired = 1

            item.shopItemImage.tag = i
            item.shopItemImage.isUserInteractionEnabled = true
            item.shopItemImage.addGestureRecognizer(tapGesture)

            items.append(item)
        }
    }

    @objc func showUpgradeView(_ sender: UITapGestureRecognizer) {
        guard let upgradeView = UpgradeItemView.instanceFromNib(), let tag =
sender.view?.tag  else { return }
        upgradeView.frame = .init(origin: .zero, size: view.frame.size)
        view.addSubview(upgradeView)
        upgradeView.shopProtocol = self
```

```
        upgradeView.configureButtons()
        upgradeView.configureView(item: Shop.shared.items[tag], index: tag)
        upgradeView.animateIn()
    }

    func addItems() {
        for i in 0 ..< items.count {
            items[i].frame = CGRect(x:view.frame.width * itemPosition[i].x,
                                    y: view.frame.height * itemPosition[i].y,
                                    width: view.frame.width * 0.3,
                                    height: view.frame.width * 0.3)
            view.addSubview(items[i])
            items[i].updateConstraints(with: view.frame.width * 0.05)
        }
    }


    @objc func backButtonTapped() {
        navigationController?.popViewController(animated: true)
    }

}
extension ShopViewController: ShopProtocol {
    func updateUI() {
        for i in 0..<items.count {
            items[i].levelItemLabel.text = "Lvl \(Shop.shared.getLvl(for: i))"
        }

        coinsLabel.text = Shop.shared.formatNumber(Shop.shared.coins)

        attackLabel.text = "\(Shop.shared.items[0].stat + Shop.shared.items[1].stat)"
        healthLabel.text = "\(Shop.shared.items[2].stat + Shop.shared.items[3].stat)"
    }
}
```

LevelsViewController.swift:

```
import UIKit
import CoreMedia


class LevelsViewController: BaseViewController, UIScrollViewDelegate {

    @IBOutlet weak var leftArrowButton: UIButton!
    @IBOutlet weak var rightArrowButton: UIButton!
    @IBOutlet weak var scrollView: UIScrollView!
    @IBOutlet weak var pageControl: UIPageControl!
    @IBOutlet weak var backButton: UIButton!

    var levels:[LevelView] = []
    var levelTuple: [(index: Int,lowerBound: CGFloat, upperBound: CGFloat)] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        createSlides()
        setupSlideScrollView(levels)
        configureGesturesAndSliding()

        scrollView.delegate = self
```

```
        backButton.addTarget(self, action: #selector(backButtonTapped), for:
.touchUpInside)

        pageControl.numberOfPages = levels.count

        pageControl.currentPage = Level.shared.level
        pageControl.isHidden = true
        scrollView.showsHorizontalScrollIndicator = false
        scrollView.contentOffset.x = view.frame.width * CGFloat(pageControl.currentPage)

        leftArrowButton.addTarget(self, action: #selector(arrowButtonTapped(_:)), for:
.touchUpInside)
        rightArrowButton.addTarget(self, action: #selector(arrowButtonTapped(_:)), for:
.touchUpInside)

        updateButtons()
    }

    func configureGesturesAndSliding() {
        let charactersOffset: CGFloat = CGFloat(1.0 / Double(levels.count-1))
        for i in 0..<levels.count-1 {
            levelTuple.append((i,CGFloat(i) * charactersOffset, CGFloat(i+1) *
charactersOffset))
        }
    }

    @objc func backButtonTapped() {
        self.navigationController?.popViewController(animated: true)
    }

    @objc func levelSelected(_ sender: UIButton) {
        print(sender.tag)
        if Level.shared.isLvlUnlocked(sender.tag) {
            print("unlocked \(sender.tag)")
            Level.shared.level = sender.tag
            updateButtons()
        }
    }

    @objc func changePage() {
        let x = CGFloat(pageControl.currentPage) * scrollView.frame.size.width
        scrollView.setContentOffset(CGPoint(x:x, y:0), animated: true)
    }

    @objc func arrowButtonTapped(_ sender: UIButton) {
        if sender == rightArrowButton {
            pageControl.currentPage += 1
        } else if sender == leftArrowButton {
            pageControl.currentPage -= 1
        }
        changePage()
    }

    func updateButtons() {
        levels.forEach {
            if Level.shared.level == $0.selectButton.tag {
                print("selected \($0.selectButton.tag)")
                $0.selectButton.isHidden = true
            } else {
                $0.selectButton.isHidden = false
            }
```

```
        }
    }

    func createSlides() {

        for i in 0...3 {
            let level:LevelView = Bundle.main.loadNibNamed("LevelView", owner: self,
options: nil)?.first as! LevelView
            level.levelImage.tag = i
            level.levelImage.image = .init(named: Level.shared.levels[i])
            level.levelLabel.image = .init(named: Level.shared.labels[i])
            level.selectButton.tag = i
            level.selectButton.addTarget(self, action: #selector(levelSelected(_:)), for:
.touchUpInside)
            levels.append(level)
        }
    }

    func setupSlideScrollView(_ levels: [LevelView]) {
        scrollView.frame = CGRect(x: 0, y: 0, width: view.frame.width, height:
view.frame.height)
        scrollView.contentSize = CGSize(width: view.frame.width * CGFloat(levels.count),
                                        height: view.frame.height * 0.8)
        scrollView.isPagingEnabled = true

        for i in 0 ..< levels.count {
            levels[i].frame = CGRect(x: view.frame.width * CGFloat(i), y: 0, width:
view.frame.width, height: view.frame.height)
            scrollView.addSubview(levels[i])
        }
    }

    func scrollViewDidScroll(_ scrollView: UIScrollView) {
        let pageIndex = round(scrollView.contentOffset.x/view.frame.width)
        pageControl.currentPage = Int(pageIndex)

        let maximumHorizontalOffset: CGFloat = scrollView.contentSize.width -
scrollView.frame.width
        let currentHorizontalOffset: CGFloat = scrollView.contentOffset.x

        // vertical
        let maximumVerticalOffset: CGFloat = scrollView.contentSize.height -
scrollView.frame.height
        let currentVerticalOffset: CGFloat = scrollView.contentOffset.y

        let percentageHorizontalOffset: CGFloat = currentHorizontalOffset /
maximumHorizontalOffset
        let percentageVerticalOffset: CGFloat = currentVerticalOffset /
maximumVerticalOffset

        let percentOffset: CGPoint = CGPoint(x: percentageHorizontalOffset, y:
percentageVerticalOffset)

        let charactersOffset: CGFloat = CGFloat(1.0 / Double(levels.count-1))
        for element in levelTuple {
            if pageControl.currentPage == levels.count - 1 {
                if percentageHorizontalOffset == 1.0  {
                    if element.index == Level.shared.level {
                        levels[pageControl.currentPage].levelLabel.isHidden = false
                        levels[pageControl.currentPage].levelLabel.show()
                    } else {
```

```
                        levels[pageControl.currentPage].levelLabel.isHidden = false
                        levels[pageControl.currentPage].selectButton.isHidden = false
                        levels[pageControl.currentPage].levelLabel.show()
                        levels[pageControl.currentPage].selectButton.show()
                    }
                } else {
                    for i in 0..<levels.count {
                        levels[i].levelLabel.isHidden = true
                        levels[i].selectButton.isHidden = true
                    }
                }
            } else if pageControl.currentPage == element.index {
                if percentageHorizontalOffset == element.lowerBound {
                    if element.index == Level.shared.level {
                        levels[pageControl.currentPage].levelLabel.isHidden = false
                        levels[pageControl.currentPage].levelLabel.show()
                    } else {
                        levels[pageControl.currentPage].levelLabel.isHidden = false
                        levels[pageControl.currentPage].selectButton.isHidden = false
                        levels[pageControl.currentPage].levelLabel.show()
                        levels[pageControl.currentPage].selectButton.show()
                    }
                } else {
                    for i in 0..<levels.count {
                        levels[i].levelLabel.isHidden = true
                        levels[i].selectButton.isHidden = true
                    }
                }
            }

            if percentOffset.x > element.lowerBound && percentOffset.x <=
element.upperBound {
                levels[element.index].levelImage.transform = CGAffineTransform(scaleX:
(element.upperBound-percentOffset.x)/charactersOffset, y: (element.upperBound-
percentOffset.x)/charactersOffset)
                levels[element.index+1].levelImage.transform = CGAffineTransform(scaleX:
percentOffset.x/element.upperBound, y: percentOffset.x/element.upperBound)
            }
        }
    }

    deinit {
        print("deinit called")
    }

}
```

SoundManager.swift:

```
import Foundation
import AVFoundation

class SoundManager : NSObject, AVAudioPlayerDelegate {
    static let sharedInstance = SoundManager()

    let MuteKey = "MuteKey"
    private(set) var isMuted = false
    var audioPlayer : AVAudioPlayer?
    var trackPosition = 0

    //Music: http://www.bensound.com/royalty-free-music
    static private let tracks = [
```

```
        "main"
    ]

    private override init() {
        //This is private so you can only have one Sound Manager ever.
        trackPosition = Int(arc4random_uniform(UInt32(SoundManager.tracks.count)))

        let defaults = UserDefaults.standard

        isMuted = defaults.bool(forKey: MuteKey)
    }

    public func startPlaying() {
        if !isMuted && (audioPlayer == nil || audioPlayer?.isPlaying == false) {
            let soundURL = Bundle.main.url(forResource:
SoundManager.tracks[trackPosition], withExtension: "mp3")

            do {
                audioPlayer = try AVAudioPlayer(contentsOf: soundURL!)
                audioPlayer?.delegate = self
            } catch {
                print("audio player failed to load")
                startPlaying()
            }

            audioPlayer?.prepareToPlay()
            audioPlayer?.play()
            audioPlayer?.volume = 0.2
            trackPosition = (trackPosition + 1) % SoundManager.tracks.count
        } else {
            print("Audio player is already playing!")
        }
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool) {
        startPlaying()
    }

    func toggleMute() -> Bool {
        isMuted = !isMuted

        let defaults = UserDefaults.standard
        defaults.set(isMuted, forKey: MuteKey)
        defaults.synchronize()

        if isMuted {
            audioPlayer?.stop()
        } else {
            startPlaying()
        }

        return isMuted
    }
```

LevelModel.swift:

```
import Foundation

class Level {

    let key = "currentLevel"
```

```
static let shared = Level()

var levels = ["village_lvl","factory_lvl","mountain_lvl", "castle_lvl"]
var labels = ["village_label","factory_label","mountain_label", "castle_label"]
private var levelBg = ["village_bg","factory_bg","mountain_bg","castle_bg"]

var level: Int {
    get {
        return UserDefaults.standard.integer(forKey: key)
    }
    set {
        UserDefaults.standard.set(newValue, forKey: key)
    }
}


func unlockLvl(_ level: Int) {
    UserDefaults.standard.set(true, forKey: "level_\(level)")
}

func isLvlUnlocked(_ level: Int) -> Bool {
    return UserDefaults.standard.bool(forKey: "level_\(level)")
}

func getBGforLevel(_ level: Int) -> String {
    return levelBg[level]
}

func getKills(for level: Int) -> Int {
    return UserDefaults.standard.integer(forKey: "kills_\(level)")
}

func setKills(for level: Int, kills: Int) {
    UserDefaults.standard.set(kills,forKey: "kills_\(level)")
}

func getTime(for level: Int) -> Double {
    return UserDefaults.standard.double(forKey: "kills_\(level)")
}

func setTime(for level: Int, time: Double) {
    UserDefaults.standard.set(time,forKey: "time_\(level)")
}

init() {
    UserDefaults.standard.register(defaults: [key : 0])
    for i in 0...3 {
        UserDefaults.standard.register(defaults: [
            "level_\(i)":true,
            "kills_\(i)":0,
            "time_\(i)":0.0,
        ])
    }
}

func formatNumber(_ n: Int) -> String {
    let num = abs(Double(n))
    let sign = (n < 0) ? "-" : ""

    switch num {
```

```
        case 1_000_000_000...:
            var formatted = num / 1_000_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)B"

        case 1_000_000...:
            var formatted = num / 1_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)M"

        case 1_000...:
            var formatted = num / 1_000
            formatted = formatted.reduceScale(to: 2)
            return "\(sign)\(formatted)K"

        case 0...:
            return "\(n)"

        default:
            return "\(sign)\(n)"
        }
    }


    func stringFromTimeInterval(interval: TimeInterval) -> String {

        let ti = NSInteger(interval)

        let seconds = ti % 60
        let minutes = (ti / 60) % 60

        return String(format: "%0.2d:%0.2d",minutes,seconds)
    }
}
```

ShopModel.swift:

```
import Foundation
import CoreGraphics

enum ItemType {
    case attack, deffense
}

class Shop {

    let key = "coins"

    static let shared = Shop()

    var items: [Item] = []
    private let itemsImages =
["weapon_shop_frame","gloves_shop_frame","hat_shop_frame","armor_shop_frame"]


    var health: CGFloat {
        return CGFloat(items[3].stat + items[2].stat)
    }

    var damage: CGFloat {
        return CGFloat(items[0].stat + items[1].stat)
    }
```

```
    var coins: Int {
        get {
            return UserDefaults.standard.integer(forKey: key)
        }
        set {
            UserDefaults.standard.set(newValue, forKey: key)
        }
    }


    func setLvl(for item: Int, lvl: Int) {
        UserDefaults.standard.set(lvl, forKey: "item_\(item)_shop")
    }

    func getLvl(for item: Int) -> Int {
        return UserDefaults.standard.integer(forKey: "item_\(item)_shop")
    }

    func getItemImage(for item: Int) -> String {
        return itemsImages[item]
    }


    init() {
        UserDefaults.standard.register(defaults: [key : 0])
        for i in 0...3 {
            UserDefaults.standard.register(defaults: [
                "item_\(i)_shop":1,
            ])
            items.append(.init(name: itemsImages[i], lvl: getLvl(for: i), type: i <= 1 ?
.attack : .deffense))
        }
    }

    func formatNumber(_ n: Int) -> String {
        let num = abs(Double(n))
        let sign = (n < 0) ? "-" : ""

        switch num {
        case 1_000_000_000...:
            var formatted = num / 1_000_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)B"

        case 1_000_000...:
            var formatted = num / 1_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)M"

        case 1_000...:
            var formatted = num / 1_000
            formatted = formatted.reduceScale(to: 2)
            return "\(sign)\(formatted)K"

        case 0...:
            return "\(n)"

        default:
            return "\(sign)\(n)"
        }
```

```
    }
}

struct Item {
    var name: String
    var lvl: Int
    var type: ItemType
    var stat: Int {
        switch type {
        case .attack:
            return 10 + 4 * lvl
        case .deffense:
            return 50 + 4 * lvl
        }
    }
    var price: Int {
        return 100 * lvl * lvl
    }
}
```

Settings.swift:

```
import Foundation
import UIKit
import AudioToolbox
import AVFAudio

class Settings {

    static let shared = Settings()

    let gameFont = "Krungthep"

    let keySFX = "SFX"
    let keyVibration = "Vibration"
    let keyHighscore = "Highscore"
    let bonusKey = "Bonus"

    var isAllowedSFX: Bool {
        get {
            return UserDefaults.standard.bool(forKey: keySFX)
        }
        set {
            UserDefaults.standard.set(newValue, forKey: keySFX)
        }
    }

    var isBonusCollected: Bool {
        get {
            return UserDefaults.standard.bool(forKey: bonusKey)
        }
        set {
            UserDefaults.standard.set(newValue, forKey: bonusKey)
        }
    }

    var isAllowedVibration: Bool {
        get {
            return UserDefaults.standard.bool(forKey: keyVibration)
        }
        set {
```

```
            UserDefaults.standard.set(newValue, forKey: keyVibration)
        }
    }

    var highscore: Int {
        get {
            return UserDefaults.standard.integer(forKey: keyHighscore)
        }
        set {
            UserDefaults.standard.set(newValue, forKey: keyHighscore)
        }
    }

    var brightness: CGFloat {
        get {
            return UIScreen.main.brightness
        }
        set {
            UIScreen.main.brightness = newValue
        }

    }


    init() {
        UserDefaults.standard.register(defaults: [
            bonusKey: false,
            keySFX : true,
            keyVibration: true,
            keyHighscore: 0,
        ])
    }
}

struct Manager
{
   ///AVAudio Player
    static var player: AVAudioPlayer?
}
```

MenuScene.swift:

```
import SpriteKit
import GameplayKit

class MenuScene: SKScene, SKPhysicsContactDelegate {
    var background: SKSpriteNode!
    var player: Player?
    var enemies: [Enemy] = []

    private var lastUpdateTime : TimeInterval = 0
    var enemySpawnInterval: TimeInterval = 0.5
    var currentEnemySpawnInterval: TimeInterval = 0.0

    override func sceneDidLoad() {
        self.lastUpdateTime = 0
    }

    override func didMove(to view: SKView) {
```

```
        setUpPhysics()
        addBackground()
    }

    private func setUpPhysics() {
        physicsWorld.contactDelegate = self
        physicsWorld.gravity = CGVector(dx: 0.0, dy: 0.0)
    }

    override func update(_ currentTime: TimeInterval) {

        if (self.lastUpdateTime == 0) {
            self.lastUpdateTime = currentTime
        }
        let dt = currentTime - self.lastUpdateTime
        self.lastUpdateTime = currentTime

        currentEnemySpawnInterval += dt
        if currentEnemySpawnInterval >= enemySpawnInterval {
            currentEnemySpawnInterval = 0
            spawnEnemy()
        }

        enemies.forEach {
            $0.position.y -= 1
            if $0.position.y <= -frame.height / 2 {
                $0.removeFromParent()
                let enemy = $0
                enemies.removeAll(where: { $0 == enemy })
            }
        }

    }

    //MARK: - Game Objects

    func addPlayer() {
        player = .init(texture: nil, color: .red, size: .init(width: size.width * 0.05,
height: size.width * 0.075))
        player?.position = .zero
        player?.setPhysicsBody()
        addChild(player!)
    }

    func spawnEnemy() {
        let enemyNames = ["bat","orc", "ghost", "zombie"]
        let name = enemyNames.randomElement()!
        let enemy: Enemy = .init(moveTextureName: name,
                                 size: .init(width:  name == "bat" ? frame.width * 0.2 :
frame.width * 0.1,
                                             height: frame.width * 0.12))

        enemy.position = .init(x: CGFloat.random(in: frame.width * 0.1 ... frame.width *
0.9),
                               y: frame.height)
        enemy.configurePhysicsBody()
        enemy.addMoveAnimation()
        addChild(enemy)
        enemies.append(enemy)
    }
```

```
    func addBackground() {
        background = .init(texture: .init(imageNamed: "village_bg"),
                           size: frame.size)
        background.position.y = frame.height / 2
        background.position.x = frame.width / 2
        background.zPosition = -1000
        addChild(background)
    }

    func changeBGtexture(with name: String) {
        background.texture = .init(imageNamed: name)
    }

    deinit {
        print( "gamescene was deallocated")
    }
}
```

GameOverScene.swift:

```
import SpriteKit
import GameplayKit

protocol GameOverProtocol: AnyObject {
    func removeView()
}

class GameOverScene: SKScene {
    var background: SKShapeNode!
    var enemies: [Enemy] = []

    weak var gameOverProtocol: GameOverProtocol?

    override func didMove(to view: SKView) {
        setUpPhysics()
        backgroundColor = .clear
        addBackground()
        for i in 0..<10 {
            for j in 0..<10 {
                spawnEnemy(pos: .init(x: 0 - CGFloat(i) * (size.width * 0.125),
                                      y: frame.height - (CGFloat(j) * size.width * 0.3)))
            }
        }
        enemies.forEach{
            addChild($0)
        }
        if Settings.shared.isAllowedSFX {
            run(.playSoundFileNamed("gameOver", waitForCompletion: true))
        }
    }

    private func setUpPhysics() {
        physicsWorld.gravity = CGVector(dx: 0.0, dy: 0.0)
    }

    override func update(_ currentTime: TimeInterval) {
        enemies.forEach {
            if $0.position.x >= frame.width * 0.8 {
                if background.action(forKey: "fading") == nil {
```

```
                    background.run(.sequence([
                        .fadeOut(withDuration: 1.5),
                        .run { [weak self] in
                            guard let self = self else { return}
                            self.gameOverProtocol?.removeView()
                        }
                    ]),withKey: "fading")
                }
            }
            $0.position.x += frame.width * 0.025
            if $0.position.x >= frame.width * 1.1 {
                $0.removeFromParent()
                let enemy = $0
                enemies.removeAll(where: { $0 == enemy })
            }
        }

    }

    //MARK: - Game Objects
    func spawnEnemy(pos: CGPoint) {
        let enemy: Enemy = .init(moveTextureName: "ghost",
                                 size: .init(width: size.width * 0.3,
                                             height: size.width * 0.45))
        enemy.position = pos
        enemy.addMoveAnimation()
        enemies.append(enemy)
    }

    func addBackground() {
        background = .init(rectOf: frame.size)
        background.fillColor = .black
        background.strokeColor = .black
        background.position.y = frame.height / 2
        background.position.x = frame.width / 2
        background.zPosition = -1000
        addChild(background)
    }

    deinit {
        print( "gameoverscene was deallocated")
    }
}
```

GameScene.swift:

```
import SpriteKit
import GameplayKit

protocol GameProtocol: AnyObject {
    func gameOver(kill: Int, time:TimeInterval, coins: Int)
    func backButtonTapped()
    func restartGame()
}

class GameScene: SKScene {

    weak var gameProtocol: GameProtocol?

    var gameCamera: Camera = Camera()
    var hud: Hud!
```

```
    var levelUpHud: LevelUpgradeHud!
    var player: Player?
    var enemies: [Enemy] = []
    var xpNodes: [LevelXpNode] = []
    var backgrounds: [SKSpriteNode] = []


    lazy var analogJoystick: AnalogJoystick = {
        let js = AnalogJoystick(diameter: frame.width * 0.3, colors: nil, images:
(substrate: .init(named: "base"), stick: .init(named: "stick")))
        js.position = CGPoint(x: size.width * 0.4 - js.radius,
                              y: -size.height * 0.4 + js.radius)
        js.zPosition = 1000
        js.alpha = 0.5
        return js
    }()

    lazy var gameState: GKStateMachine = .init(states: [
        GameOver.init(scene: self),
        Playing.init(scene: self),
        Paused.init(scene: self)
    ])

    private var lastUpdateTime : TimeInterval = 0
    var enemySpawnInterval: TimeInterval = 3.0
    var currentEnemySpawnInterval: TimeInterval = 0.0
    var currentCrateSpawnInterval: TimeInterval = 10.0
    var crateSpawnInterval: TimeInterval = 10.0

    override func sceneDidLoad() {
        self.lastUpdateTime = 0
    }

    override func didMove(to view: SKView) {

        setUpPhysics()

        configureCamera()
        addPlayer()
        addHud()
        addBackground(isGameStart: true)

        setupControlJoystick()

        gameState.enter(Playing.self)
    }

    private func setUpPhysics() {
        physicsWorld.contactDelegate = self
        physicsWorld.gravity = CGVector(dx: 0.0, dy: 0.0)
    }

    override func update(_ currentTime: TimeInterval) {

        guard gameState.currentState is Playing else { return }

        if (self.lastUpdateTime == 0) {
            self.lastUpdateTime = currentTime
        }
        let dt = currentTime - self.lastUpdateTime
        self.lastUpdateTime = currentTime
```

```
        if let player = player {

            currentEnemySpawnInterval += dt
            if currentEnemySpawnInterval >= enemySpawnInterval {
                currentEnemySpawnInterval = 0
                for _ in 0 ..< Int.random(in: 1...10) {
                    spawnEnemy(playerPosY: player.position.y)
                }
            }

            player.update(with: dt, enemies: enemies)
            gameCamera.setDestination(destination: .init(x: 0, y: player.position.y))
            gameCamera.update()
            enemies.forEach {
                $0.update(player: player)
            }
            xpNodes.forEach {
                $0.update(playerPosition: player.position)
            }
            if backgrounds.last!.contains(player.position) {
                addBackground(toTop: false)
            } else if backgrounds.first!.contains(player.position) {
                addBackground(toTop: true)
            }
            currentCrateSpawnInterval += dt
            if currentCrateSpawnInterval >= crateSpawnInterval {
                currentCrateSpawnInterval = 0
                spawnCrate(playerPosY: player.position.y)
            }
        }

        hud.updateTime(dt)
    }

    //MARK: - Game Objects

    func addPlayer() {
        let player: Player = .init(texture: .init(imageNamed: "player_0"),
                                   color: .clear,
                                   size: .init(width: size.width * 0.075, height:
size.width * 0.1))
        player.position = .zero
        player.setPhysicsBody()
        addChild(player)

        player.addWeapon(type: .shuriken)

        self.player = player
    }

    func spawnEnemy(playerPosY: CGFloat ) {
        let enemyNames = ["bat","orc", "ghost", "zombie"]
        let name = enemyNames.randomElement()!
        let enemy: Enemy = .init(moveTextureName: name,
                                 size: .init(width:  name == "bat" ? frame.width * 0.2 :
frame.width * 0.1,
                                             height: frame.width * 0.12))
        let heightShift = CGFloat.random(in: frame.height * 0.4 ... frame.height * 0.7)
        enemy.position = .init(x: CGFloat.random(in: frame.width * -0.45 ... frame.width *
0.45),
```

```
                                y: Bool.random() ? playerPosY + heightShift : playerPosY -
heightShift)
        enemy.configurePhysicsBody()
        enemy.addMoveAnimation()
        addChild(enemy)
        enemies.append(enemy)
    }

    func addHud() {
        hud = .init(size: frame.size)
        hud.zPosition = 1000
        hud.backButtonHandler = { [weak self] in
            guard let self = self else { return }
            self.gameProtocol?.backButtonTapped()
        }

        gameCamera.addChild(hud)
    }

    func addLevelUpHud() {
        levelUpHud = .init(size: frame.size)
        levelUpHud.zPosition = 1100
        levelUpHud.selectButtonHandler = { [weak self] weapon in
            guard let self = self else { return }
            self.hud.resetLevelProgressBar()
            self.levelUpHud.hideHud()

            guard let player = self.player else {
                return
            }

            switch weapon {
            case .mainWeapon:
                player.increaseLevel()
            case .shuriken:
                if let shuriken = player.shuriken {
                    shuriken.increaseLevel()
                } else {
                    player.addWeapon(type: .shuriken)
                }
            case .lightning:
                if let lightning = player.lightning {
                    lightning.increaseLevel()
                } else {
                    player.addWeapon(type: .lightning)
                }
            case .poisonousField:
                if let poisonousField = player.poisonousField {
                    poisonousField.increaseLevel()
                } else {
                    player.addWeapon(type: .poisonousField)
                }
            case .ball:
                if let ball = player.ball {
                    ball.increaseLevel()
                } else {
                    player.addWeapon(type: .ball)
                }
            case .drill:
                if let drill = player.drill {
                    drill.increaseLevel()
```

```
                } else {
                    player.addWeapon(type: .drill)
                }
            case .rocket:
                if let rocket = player.rocket {
                    rocket.increaseLevel()
                } else {
                   player.addWeapon(type: .rocket)
                }
            }

            self.gameState.enter(Playing.self)
        }
        gameCamera.addChild(self.levelUpHud)
    }


    func addBackground(isGameStart:  Bool = false, toTop: Bool = false) {
        if isGameStart {
            for i in 0..<3 {
                let background: SKSpriteNode = .init(texture: .init(imageNamed:
Level.shared.getBGforLevel(Level.shared.level)),
                                                     size: frame.size)
                background.position.y = frame.height * (1.5 - CGFloat(i))
                background.zPosition = -1000
                backgrounds.append(background)
                addChild(background)
            }
        } else {
            if toTop {
                let background: SKSpriteNode = .init(texture: .init(imageNamed:
Level.shared.getBGforLevel(Level.shared.level)),
                                                     size: frame.size)
                background.position.y = backgrounds.first!.position.y + frame.height
                background.zPosition = -1000
                backgrounds.insert(background, at: 0)
                addChild(background)
            } else {
                let background: SKSpriteNode = .init(texture: .init(imageNamed:
Level.shared.getBGforLevel(Level.shared.level)),
                                                     size: frame.size)
                background.position.y = backgrounds.last!.position.y - frame.height
                background.zPosition = -1000
                backgrounds.append(background)
                addChild(background)
            }
        }
    }

    func configureCamera() {
        camera = gameCamera
        gameCamera.position = .init(x: frame.midX,
                                    y: frame.midY)
        gameCamera.setDestination(destination: .init(x: frame.midX,
                                                     y: frame.midY))
        gameCamera.addBorder(frame: frame)
        addChild(gameCamera)
    }

    func showDamage(_ value: CGFloat, enemyPos: CGPoint) {
        let dmg = value.rounded()
```

```
        let damage = SKLabelNode(text: "\(dmg)")
        damage.fontSize = 20
        damage.zPosition = 500
        damage.fontColor = .orange
        damage.fontName = GameFont

        damage.position = enemyPos

        addChild(damage)

        damage.run(.sequence([
            .move(by: .init(dx: CGFloat.random(in: -20...20),
                            dy: CGFloat.random(in: -20...20)),
                  duration: 0.2),
            .removeFromParent()
        ]))
    }

    func spawnCrate(playerPosY: CGFloat) {
        let crate: CrateNode = .init(texture: .init(imageNamed: "crate"),
                                     color: .clear,
                                     size: .init(width: frame.width * 0.1,
                                                 height: frame.width * 0.1))
        crate.configurePhysicsBody()
        crate.position = .init(x: CGFloat.random(in:frame.width * -0.4 ... frame.width *
0.4),
                               y: CGFloat.random(in: playerPosY - frame.height / 2 ...
playerPosY + frame.height / 2))
        print(crate.position)
        addChild(crate)
    }

    //MARK: - Joystick setup
    func setupControlJoystick() {
        gameCamera.addChild(analogJoystick)
        analogJoystick.trackingHandler = { [unowned self] data in

            guard let player = player else { return }
            if player.position.x + data.velocity.x * 0.05 > frame.width * 0.4 ||
player.position.x + data.velocity.x * 0.05 < -frame.width * 0.4 {
                player.position.y += data.velocity.y * 0.05
                player.updateDirection(data: data.velocity)
            } else {
                player.position.y += data.velocity.y * 0.05
                player.position.x += data.velocity.x * 0.05
                player.updateDirection(data: data.velocity)
            }
            player.toLeft = data.velocity.x > 0 ? true : false
            if player.action(forKey: player.walkActionKey) == nil {
                player.addMoveAnimation(toLeft: player.toLeft)
            }
        }
        analogJoystick.stopHandler = { [unowned self] in
            self.player?.removeAllActions()
        }
    }

    deinit {
        print( "gamescene was deallocated")
    }
}
```

SettingsView.swift:

```
import UIKit
import SwiftUI

class SettingsView: UIView {

    lazy var musicLabel: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 1
        lbl.textColor = .white
        lbl.font = .init(name: settings.gameFont, size: 30)
        lbl.text = "Music"

        lbl.translatesAutoresizingMaskIntoConstraints = false

        return lbl
    }()

    lazy var settingsLabel: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 1
        lbl.textColor = .white
        lbl.font = .init(name: settings.gameFont, size: 30)
        lbl.text = "Settings"

        lbl.translatesAutoresizingMaskIntoConstraints = false

        return lbl
    }()

    lazy var sfxLabel: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 1
        lbl.textColor = .white
        lbl.font = .init(name: settings.gameFont, size: 30)
        lbl.text = "Sounds"

        lbl.translatesAutoresizingMaskIntoConstraints = false

        return lbl
    }()

    lazy var vibrationLabel: UILabel = {
        let lbl = UILabel()
        lbl.numberOfLines = 1
        lbl.textColor = .white
        lbl.font = .init(name: settings.gameFont, size: 30)
        lbl.text = "Haptic"

        lbl.translatesAutoresizingMaskIntoConstraints = false

        return lbl
    }()

    lazy var musicSwitch: UISwitch = {
        let swtch = UISwitch()
        swtch.onTintColor = .green
        swtch.addTarget(self, action: #selector(toggleSwitch(_:)), for: .valueChanged)
        return swtch
```

```
}()

lazy var sfxSwitch: UISwitch = {
    let swtch = UISwitch()
    swtch.onTintColor = .green
    swtch.addTarget(self, action: #selector(toggleSwitch(_:)), for: .valueChanged)
    return swtch
}()

lazy var vibrationSwitch: UISwitch = {
    let swtch = UISwitch()
    swtch.onTintColor = .green
    swtch.addTarget(self, action: #selector(toggleSwitch(_:)), for: .valueChanged)
    return swtch
}()

lazy var backButton: UIButton = {
    let btn = UIButton()
    btn.setBackgroundImage(.init(named: "close_button"), for: .normal)

    btn.addTarget(self, action: #selector(animateOut), for: .touchUpInside)
    btn.translatesAutoresizingMaskIntoConstraints = false

    return btn
}()


lazy var shadowBackground: UIView = {
    let view = UIView()
    view.backgroundColor = .black
    view.alpha = 0.8
    view.translatesAutoresizingMaskIntoConstraints = false
    return view
}()

lazy var container: UIImageView = {
    let view = UIImageView(image: .init(named: "popup_view"))
    view.translatesAutoresizingMaskIntoConstraints = false
    view.isUserInteractionEnabled = true
    return view
}()

let settings = Settings.shared

override init(frame: CGRect) {
    super.init(frame: frame)
    configureView()
}

required init?(coder: NSCoder) {
    fatalError("init(coder:) has not been implemented")
}

func configureView() {

    musicSwitch.isOn = !SoundManager.sharedInstance.isMuted
    vibrationSwitch.isOn = settings.isAllowedVibration
    sfxSwitch.isOn = settings.isAllowedSFX

    addSubview(shadowBackground)
    NSLayoutConstraint.activate([
```

```
    shadowBackground.leadingAnchor.constraint(equalTo: self.leadingAnchor),
    shadowBackground.trailingAnchor.constraint(equalTo: self.trailingAnchor),
    shadowBackground.topAnchor.constraint(equalTo: self.topAnchor),
    shadowBackground.bottomAnchor.constraint(equalTo: self.bottomAnchor)
])

addSubview(container)
NSLayoutConstraint.activate([
    container.centerXAnchor.constraint(equalTo: self.centerXAnchor),
    container.centerYAnchor.constraint(equalTo: self.centerYAnchor),
    container.widthAnchor.constraint(equalToConstant: 300),
    container.heightAnchor.constraint(equalToConstant: 300)
])

let stackView = UIStackView()
stackView.alignment = .center
stackView.distribution = .equalSpacing
stackView.spacing = 10
stackView.axis = .vertical
stackView.translatesAutoresizingMaskIntoConstraints = false
addSubview(stackView)
NSLayoutConstraint.activate([
    stackView.centerXAnchor.constraint(equalTo: self.centerXAnchor),
    stackView.centerYAnchor.constraint(equalTo: self.centerYAnchor)
])

let labelsStack = UIStackView()
labelsStack.alignment = .leading
labelsStack.distribution = .equalSpacing
labelsStack.spacing = 10
labelsStack.axis = .vertical

labelsStack.addArrangedSubview(musicLabel)
labelsStack.addArrangedSubview(sfxLabel)
labelsStack.addArrangedSubview(vibrationLabel)

let switchStack = UIStackView()
switchStack.alignment = .center
switchStack.distribution = .equalSpacing
switchStack.spacing = 15
switchStack.axis = .vertical

switchStack.addArrangedSubview(musicSwitch)
switchStack.addArrangedSubview(sfxSwitch)
switchStack.addArrangedSubview(vibrationSwitch)

let horStackView = UIStackView()
horStackView.alignment = .center
horStackView.distribution = .equalSpacing
horStackView.spacing = 20
horStackView.axis = .horizontal

stackView.addArrangedSubview(settingsLabel)
stackView.addArrangedSubview(horStackView)
horStackView.addArrangedSubview(labelsStack)
horStackView.addArrangedSubview(switchStack)


container.addSubview(stackView)

container.addSubview(backButton)
```

```
        NSLayoutConstraint.activate([
            backButton.trailingAnchor.constraint(equalTo: container.trailingAnchor),
            backButton.topAnchor.constraint(equalTo: container.topAnchor),
            backButton.widthAnchor.constraint(equalToConstant: 50),
            backButton.heightAnchor.constraint(equalToConstant: 50)
        ])

    }

    @objc func toggleSwitch(_ sender: UISwitch) {
        switch sender {
        case musicSwitch:
            SoundManager.sharedInstance.toggleMute()
        case sfxSwitch:
            settings.isAllowedSFX = sender.isOn
        case vibrationSwitch:
            settings.isAllowedVibration = sender.isOn
        default:
            break
        }
        if Settings.shared.isAllowedVibration {
            Vibration.medium.vibrate()
        }
    }

    @objc func animateOut() {
        UIView.animate(withDuration: 1.5, delay: 0, usingSpringWithDamping: 0.7,
                       initialSpringVelocity: 1, options: .curveEaseIn, animations: {
            self.container.transform = CGAffineTransform(translationX: 0 , y: -
self.frame.height)
            self.alpha = 0
        }) { (complete) in
            if complete {
                self.removeFromSuperview()
            }
        }
    }

    @objc func animateIn() {
        self.container.transform = CGAffineTransform(translationX: 0 , y: -
self.frame.height)
        UIView.animate(withDuration: 0.75, delay: 0 , usingSpringWithDamping: 0.5,
                       initialSpringVelocity: 1, options: .curveEaseIn, animations: {
            self.container.transform = .identity
            self.alpha = 1
        })
    }
}
```

ShopItemView.swift:

```
import UIKit

class ShopItemView: UIView {

    @IBOutlet weak var shopItemImage: UIImageView!
    @IBOutlet weak var levelItemLabel: UILabel!

    var levelCaption: String? {
        get { return levelItemLabel?.text }
        set { levelItemLabel.text = newValue }
    }
```

```
    var itemImage: UIImage? {
        get { return shopItemImage.image }
        set { shopItemImage.image = newValue }
    }

    func updateConstraints(with offset: CGFloat) {
        levelItemLabel.translatesAutoresizingMaskIntoConstraints = false

        levelItemLabel.topAnchor.constraint(equalTo: shopItemImage.topAnchor, constant:
offset * 1.25).isActive = true
        levelItemLabel.trailingAnchor.constraint(equalTo: shopItemImage.trailingAnchor,
constant: -offset).isActive = true
    }

    class func instanceFromNib() -> ShopItemView? {
        guard let view = UINib(nibName: "ShopItemView", bundle:
nil).instantiate(withOwner: nil, options: nil)[0] as? ShopItemView else { return nil }
            return view
     }
}
```

UpgradeItemView.swift:

```
import UIKit


protocol ShopProtocol: AnyObject {
    func updateUI()
}

class UpgradeItemView: UIView {


    @IBOutlet weak var container: UIView!

    @IBOutlet weak var statValueLabel: UILabel!
    @IBOutlet weak var statImageView: UIImageView!
    @IBOutlet weak var levelUpButton: UIButton!
    @IBOutlet weak var priceLabel: UILabel!
    @IBOutlet weak var itemImage: UIImageView!
    @IBOutlet weak var closeButton: UIButton!
    @IBOutlet weak var levelLabel: UILabel!

    var index: Int?
    var shop = Shop.shared

    weak var shopProtocol: ShopProtocol?

    func configureButtons() {
        closeButton.addTarget(self, action: #selector(animateOut), for: .touchUpInside)
        levelUpButton.addTarget(self, action: #selector(levelUpButtonPressed), for:
.touchUpInside)
    }

    func configureView(item: Item, index: Int) {
        self.index = index
        statValueLabel.text = "\(item.stat)"
        statValueLabel.font = .init(name: GameFont, size: 20)
```

```
        statImageView.image = .init(named: item.type == .attack ? "sword" : "heart")
        let currentCoins = shop.formatNumber(shop.coins)
        let price = shop.formatNumber(item.price)
        priceLabel.text = "\(currentCoins)/\(price)"
        priceLabel.font = .init(name: GameFont, size: 20)

        levelLabel.text = "Lvl \(item.lvl)"
        levelLabel.font = .init(name: GameFont, size: 20)

        itemImage.image = .init(named: item.name)
    }

    @objc func levelUpButtonPressed() {
        guard let index = index else { return }
        if shop.coins >= shop.items[index].price {
            shop.coins -= shop.items[index].price
            shop.items[index].lvl += 1
            shop.setLvl(for: index, lvl: shop.items[index].lvl)
            configureView(item: shop.items[index], index: index)
            shopProtocol?.updateUI()
        }
    }

    @objc func animateOut() {
        UIView.animate(withDuration: 1.5, delay: 0, usingSpringWithDamping: 0.7,
                       initialSpringVelocity: 1, options: .curveEaseIn, animations: {
            self.container.transform = CGAffineTransform(translationX: 0 , y: -
self.frame.height)
            self.alpha = 0
        }) { (complete) in
            if complete {
                self.removeFromSuperview()
            }
        }
    }

    @objc func animateIn() {
        self.container.transform = CGAffineTransform(translationX: 0, y: -
self.frame.height)
        self.alpha = 1
        UIView.animate(withDuration: 0.5, delay: 0 , usingSpringWithDamping: 0.7,
                       initialSpringVelocity: 1, options: .curveEaseIn, animations: {
            self.container.transform = .identity
            self.alpha = 1
        })
    }


    class func instanceFromNib() -> UpgradeItemView? {
        guard let view = UINib(nibName: "UpgradeItemView", bundle:
nil).instantiate(withOwner: nil, options: nil)[0] as? UpgradeItemView else { return nil }
        return view
    }
}
```

CoinNode.swift:

```
import SpriteKit
```

```
class CoinNode: SKSpriteNode {

    var rotationFrames: [SKTexture] {
        var arr: [SKTexture] = []
        for i in 0...5 {
            arr.append(.init(imageNamed: "coin_\(i)"))
        }
        return arr
    }

    let rotationActionKey = "rotationAction"

    func configurePhysicsBody() {
        physicsBody = .init(circleOfRadius: self.frame.width / 2)
        physicsBody?.categoryBitMask = PhysicsCategory.coin
        physicsBody?.contactTestBitMask  = PhysicsCategory.lootZone
        physicsBody?.collisionBitMask = 0
    }

    func addRotationAnimation() {
        removeAction(forKey: rotationActionKey)

        let walkingAction = SKAction.repeatForever(
            SKAction.animate(with:rotationFrames,
                             timePerFrame: 0.1,
                             resize: false,
                             restore: true))

        run(walkingAction,withKey: rotationActionKey)
    }

}
```

Crate.swift:

```
import SpriteKit

enum BonusType: CaseIterable {
    case coin,health, magnet
}

class CrateNode: SKSpriteNode {

    var touches: Int = 0

    func configurePhysicsBody() {
        self.zPosition = 150
        physicsBody = .init(circleOfRadius: self.frame.width / 2)
        physicsBody?.categoryBitMask = PhysicsCategory.crate
        physicsBody?.contactTestBitMask  = PhysicsCategory.mainWeapon |
PhysicsCategory.shuriken | PhysicsCategory.drill | PhysicsCategory.lightning |
PhysicsCategory.poisonousField | PhysicsCategory.rocket
        physicsBody?.collisionBitMask = PhysicsCategory.player
    }

    func updateTouches(_ completion: @escaping (BonusType) -> Void) {
        touches += 1
        if touches == 2 {
            let bonus = CGFloat.random(in: 0...1.0)
            var type:BonusType = .coin
            switch bonus {
```

```
            case 0...0.8:
                type = .coin
            case 0.81...0.9:
                type = .magnet
            default:
                type = .health
            }

            self.physicsBody?.categoryBitMask = PhysicsCategory.none
            completion(type)
            self.removeFromParent()
        }
    }

}
```

EnemyNode.swift:

```
import SpriteKit
import CoreGraphics

class Enemy: SKSpriteNode {

    var health: CGFloat = 50
    var damage: CGFloat = 10.0
    var isDead: Bool = false

    var moveTextureName: String


    init(health: CGFloat = 50, damage: CGFloat = 10, moveTextureName: String,size: CGSize)
{
        self.health = health
        self.damage = damage
        self.moveTextureName = moveTextureName
        super.init(texture: .init(imageNamed: moveTextureName + "_0"),color: .clear, size:
size)
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    var walkFrames: [SKTexture] {
        var count = 0
        var arr: [SKTexture] = []
        while UIImage(named: "\(moveTextureName)_\(count)") != nil {
            count += 1
        }
        for i in 0..<count {
            arr.append(.init(imageNamed: "\(moveTextureName)_\(i)"))
        }
        return arr
    }

    var deathFrames: [SKTexture] {
        var count = 0
        var arr: [SKTexture] = []
        while let image = UIImage(named: "\(moveTextureName)Death_\(count)") {
            arr.append(.init(image: image))
            count += 1
```

```
        }
        return arr
    }

    let movementActionKey = "movementActionKey"
    let deathActionKey = "deathAction"

    let zombieSpeed: CGFloat = 70.0

    func configurePhysicsBody() {
        physicsBody = .init(circleOfRadius: frame.height / 2)
        physicsBody?.allowsRotation = false
        physicsBody?.categoryBitMask = PhysicsCategory.enemy
        physicsBody?.collisionBitMask = PhysicsCategory.player | PhysicsCategory.enemy
        physicsBody?.contactTestBitMask = PhysicsCategory.shuriken |
PhysicsCategory.player | PhysicsCategory.lightning | PhysicsCategory.mainWeapon |
PhysicsCategory.poisonousField | PhysicsCategory.drill | PhysicsCategory.ball |
PhysicsCategory.rocket
    }

    func hit(with value: CGFloat,completion: @escaping (Bool) -> Void) {
        if value > health {
            isDead = true
            physicsBody?.categoryBitMask = PhysicsCategory.none
            addDeathAnimation()
            completion(true)
        } else {
            health -= value
            run(.highlight(duration: 0.5))
            completion(false)
        }
    }

    func update(player: Player) {
        guard !isDead else {
            physicsBody?.velocity = .zero
            return
        }
        let targetPosition = player.position
        let currentPosition = self.position

        let angle = atan2(currentPosition.y - targetPosition.y, currentPosition.x -
targetPosition.x) + CGFloat(Double.pi)

        let velocityX = zombieSpeed * cos(angle)
        let velocityY = zombieSpeed * sin(angle)

        let newVelocity = CGVector(dx: velocityX, dy: velocityY)
        physicsBody?.velocity = newVelocity
        xScale = newVelocity.dx > 0 ? 1 : -1
    }


    func addMoveAnimation() {
        let walkingAction = SKAction.repeatForever(
            SKAction.animate(with:walkFrames,
                             timePerFrame: 0.1,
                             resize: false,
                             restore: false))

        run(walkingAction,withKey: movementActionKey)
```

```
    }

    func spawnCrystal() -> LevelXpNode? {
        let spawnChance = CGFloat.random(in: 0 ... 1)
        print("chance:\(spawnChance)")
        if spawnChance > 0.2 {
            let xpNode: LevelXpNode = .init(size:.init(width: 15,
                                                       height: 15))
           return xpNode
        } else {
            return nil
        }
    }

    func addDeathAnimation() {
        let walkingAction: SKAction =  .sequence([
            .animate(with:deathFrames,
                     timePerFrame: 0.1,
                     resize: false,
                     restore: false),
            .removeFromParent()
        ])

        run(walkingAction,withKey: deathActionKey)
    }
}
```

Player.swift:

```
import SpriteKit

class Player: SKSpriteNode {

    var shuriken: SpinningShuriken?
    var lightning: LightningNode?
    var poisonousField: PoisonousField?
    var drill: Drill?
    var ball: Ball?
    var rocket: Rocket?

    var lootZone: SKShapeNode

    var weapons: [WeaponType] = [.mainWeapon]

    var level = 1
    var damage: CGFloat = Shop.shared.damage
    var multiplier: CGFloat = 1.0
    var bulletQuantity: Int = 1
    var weaponDamage: CGFloat {
        return multiplier * damage
    }
    var health:CGFloat = Shop.shared.health
    var currentHealth: CGFloat {
        set {
            if newValue > maxHealth {
                health = maxHealth
            } else {
                health = newValue
            }
        }
        get {
```

```
        return health
    }
}
let maxHealth:CGFloat = Shop.shared.health
var toLeft = true {
    willSet {
        if newValue != toLeft {
            addMoveAnimation(toLeft: newValue)
        }
    }
}

var currentReloadInterval: TimeInterval = 0.0
var reloadInterval: TimeInterval = 2

var directionNode: SKSpriteNode
var healthBar : SKCropNode
var emptyHealthBar: SKSpriteNode
var reloadBar : SKCropNode
var emptyReloadBar: SKSpriteNode

var leftWalkFrames: [SKTexture] {
    var arr: [SKTexture] = []
    for i in 0...7 {
        arr.append(.init(imageNamed: "player_\(i)"))
    }
    return arr
}

var rightWalkFrames: [SKTexture] {
    var arr: [SKTexture] = []
    for i in 0...7 {
        arr.append(.init(imageNamed: "playerRight_\(i)"))
    }
    return arr
}

var deathFrames: [SKTexture] {
    var arr: [SKTexture] = []
    for i in 0...4 {
        arr.append(.init(imageNamed: "playerDeath_\(i)"))
    }
    return arr
}

let walkActionKey = "walkAction"
let deathActionKey = "deathAction"

var directionAngle: CGFloat = .pi


//MARK: - Init

override init(texture: SKTexture?, color: UIColor, size: CGSize) {
    //HealthBar
    let healthBaseImage = Textures.healthFrame
    let healthCoverImage = Textures.healthIndicator
    let barSize: CGSize = .init(width: size.width * 1.5, height: size.height * 0.3)
    let filledHealthImage  = SKSpriteNode(imageNamed: healthCoverImage)
    filledHealthImage.size.width = barSize.width
    filledHealthImage.size.height = barSize.height
```

```
        healthBar = SKCropNode()
        healthBar.addChild(filledHealthImage)
        healthBar.maskNode = SKSpriteNode(color: .green,
                                          size: CGSize(width: filledHealthImage.size.width
* 2,
                                                       height:
filledHealthImage.size.height))
        healthBar.maskNode?.position = .init(x: -filledHealthImage.size.width / 2, y: 0)
        healthBar.maskNode?.xScale = 0
        healthBar.zPosition = 210

        emptyHealthBar = SKSpriteNode.init(imageNamed: healthBaseImage)
        emptyHealthBar.size.width = barSize.width
        emptyHealthBar.size.height = barSize.height
        emptyHealthBar.name = "levelProgress"
        emptyHealthBar.zPosition = 200

        //ReloadBar
        let reloadBaseImage = Textures.reloadFrame
        let reloadCoverImage = Textures.reloadIndicator
        let filledReloadImage  = SKSpriteNode(imageNamed: reloadCoverImage)
        filledReloadImage.size.width = barSize.width
        filledReloadImage.size.height = barSize.height

        reloadBar = SKCropNode()
        reloadBar.addChild(filledReloadImage)
        reloadBar.maskNode = SKSpriteNode(color: .white,
                                          size: CGSize(width: filledReloadImage.size.width
* 2,
                                                       height:
filledReloadImage.size.height))
        reloadBar.maskNode?.position = .init(x: -filledReloadImage.size.width / 2, y: 0)
        reloadBar.maskNode?.xScale = 0
        reloadBar.zPosition = 210

        emptyReloadBar = SKSpriteNode.init(imageNamed: reloadBaseImage)
        emptyReloadBar.size.width = barSize.width
        emptyReloadBar.size.height = barSize.height
        emptyReloadBar.name = "levelProgress"
        emptyReloadBar.zPosition = 200

        directionNode = .init(texture: .init(imageNamed: "directionNode"),
                              color: .green,
                              size: .init(width: 10, height: 10))
        directionNode.position.x = size.height * 1.25
        directionNode.zRotation = -.pi / 2

        lootZone = .init(circleOfRadius: size.height)
        lootZone.physicsBody = .init(circleOfRadius: size.height)
        lootZone.physicsBody?.categoryBitMask = PhysicsCategory.lootZone
        lootZone.physicsBody?.contactTestBitMask = PhysicsCategory.xpNode |
PhysicsCategory.coin | PhysicsCategory.health | PhysicsCategory.magnet
        lootZone.physicsBody?.collisionBitMask = 0
        lootZone.strokeColor = .clear

        super.init(texture: texture, color: color, size: size)


        healthBar.position = .init(x: 0, y: -size.height * 0.7 )
```

```
        emptyHealthBar.position = .init(x: 0, y: -size.height * 0.7)
        reloadBar.position = .init(x: 0, y: -size.height * 0.7 -
emptyReloadBar.frame.height)
        emptyReloadBar.position = .init(x: 0, y: -size.height * 0.7 -
emptyReloadBar.frame.height)
        healthBar.maskNode?.xScale = 1.0

        addChild(emptyHealthBar)
        addChild(healthBar)

        addChild(emptyReloadBar)
        addChild(reloadBar)
        addChild(directionNode)
        addChild(lootZone)
        print(maxHealth)
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func addWeapon(type: WeaponType) {
        switch type {
        case .mainWeapon:
            break
        case .shuriken:
            let shuriken = SpinningShuriken(size: .init(width: size.height, height:
size.height), playerDamage: damage)
            addChild(shuriken)
            self.shuriken = shuriken
            weapons.append(.shuriken)
        case .lightning:
            let lightning = LightningNode(size: size, playerDamage: damage)
            lightning.position = .zero
            addChild(lightning)
            self.lightning = lightning
            weapons.append(.lightning)
        case .poisonousField:
            let poisonousField: PoisonousField = .init(size: .init(width: frame.height *
2,
                                                                   height: frame.height *
2), playerDamage: damage)
            addChild(poisonousField)
            self.poisonousField = poisonousField
            weapons.append(.poisonousField)
        case .ball:
            let ball: Ball = .init(size: .init(width: size.height * 1.25,
                                               height: size.height * 1.25),
                                   playerDamage: damage,
                                   parentScene: self.parent!)

            addChild(ball)
            self.ball = ball
            weapons.append(.ball)
        case .drill:
            let drill: Drill = .init(size: size,
                                     playerDamage: damage,
                                     parentScene: self.parent!)
            self.drill = drill
            addChild(drill)
            weapons.append(.drill)
```

```
        case .rocket:
            let rocket: Rocket = .init(size: size,
                                       playerDamage: damage,
                                       parentScene: self.parent!)
            self.rocket = rocket
            addChild(rocket)
            weapons.append(.rocket)
        }
    }


    //MARK: - Updating player

    func increaseLevel() {
        level += 1
        if level > 7 {
            multiplier *= 1.25
        } else {
            bulletQuantity += 1
            reloadInterval -= 0.1
            multiplier *= 1.25
        }
        // print("mainWeapon lvl \(level)")
    }

    func attackWithMainWeapon(dt: TimeInterval) {
        currentReloadInterval += dt
        reloadBar.maskNode?.xScale = currentReloadInterval / reloadInterval
        if currentReloadInterval >= reloadInterval {
            currentReloadInterval = 0

            for i in 0..<bulletQuantity {
                run(.wait(forDuration: Double(i) * 0.2)) { [weak self] in
                    guard let self = self  else { return }
                    let newVelocity: CGVector = .init(dx: -500 * cos(self.directionAngle),
dy: -500 * sin(self.directionAngle))

                    let bullet = SKSpriteNode(texture: .init(imageNamed: "bullet"), color:
.cyan, size: .init(width: 20, height: 10))
                    if Settings.shared.isAllowedSFX {
                        bullet.run(.playSoundFileNamed("rifle", waitForCompletion: true))
                    }
                    bullet.physicsBody = .init(rectangleOf: bullet.size)
                    bullet.physicsBody?.categoryBitMask = PhysicsCategory.mainWeapon
                    bullet.physicsBody?.collisionBitMask = 0
                    bullet.physicsBody?.contactTestBitMask = PhysicsCategory.enemy
                    bullet.zRotation = self.directionAngle + .pi
                    bullet.physicsBody?.velocity = newVelocity
                    bullet.position = self.position
                    bullet.run(.wait(forDuration: 3.0)) {
                        bullet.removeFromParent()
                    }
                    self.parent?.addChild(bullet)
                }
            }
        }
    }

    func update(with dt: TimeInterval, enemies: [Enemy]) {
        let filteredEnemies = enemies.filter( {
```

```
            let distance = sqrt(pow($0.position.x - self.position.x,2) + pow($0.position.y
- self.position.y,2))
            if distance <= 400 {
                return true
            } else {
                return false
            }
        })

        attackWithMainWeapon(dt: dt)

        if let drill = drill {
            drill.updateDrills(dt: dt)
        }

        if let ball = ball {
            ball.updateBall(dt: dt)
        }

        if let lightning = lightning {
            if filteredEnemies.count > 0 {
                lightning.updateLightning(dt:dt, enemies: filteredEnemies)
            }
        }

        if let shuriken = shuriken {
            shuriken.updateShurikens(dt:dt)
        }

        if let rocket = rocket {
            rocket.updateRocket(dt: dt, target: filteredEnemies.randomElement())
        }
    }

    func updateDirection(data: CGPoint) {
        let angle = atan2(data.y, data.x) + CGFloat(Double.pi)
        directionAngle = angle
        directionNode.zRotation = directionAngle + .pi / 2
        directionNode.position.x  = -frame.height * 1.25 * cos(angle)
        directionNode.position.y  = -frame.height * 1.25 * sin(angle)
    }

    func updateHealth(with value: CGFloat) {
        currentHealth += value
        if currentHealth >= maxHealth {
            healthBar.maskNode?.xScale = 1.0
        } else if currentHealth <= 0 {
            self.physicsBody?.categoryBitMask = PhysicsCategory.none
            healthBar.maskNode?.xScale = 0.0
        } else {
            healthBar.maskNode?.xScale = currentHealth / maxHealth
        }
    }

    //MARK: - Physics body and animation

    func setPhysicsBody() {
        self.physicsBody = .init(circleOfRadius: self.frame.width / 2)
        physicsBody?.allowsRotation = false
        physicsBody?.categoryBitMask = PhysicsCategory.player
```

```
        physicsBody?.contactTestBitMask = PhysicsCategory.enemy | PhysicsCategory.coin |
PhysicsCategory.xpNode | PhysicsCategory.health | PhysicsCategory.magnet
        physicsBody?.collisionBitMask = 0
    }

    func addMoveAnimation(toLeft: Bool) {
        removeAction(forKey: walkActionKey)
        if action(forKey: walkActionKey) != nil {
            removeAction(forKey: walkActionKey)
        } else {
            let walkingAction = SKAction.repeatForever(
                SKAction.animate(with:toLeft ? leftWalkFrames : rightWalkFrames,
                                 timePerFrame: 0.1,
                                 resize: false,
                                 restore: true))

            run(walkingAction,withKey: walkActionKey)
        }
    }

    func addDeathAnimation() {
        removeAction(forKey: deathActionKey)
        removeAction(forKey: walkActionKey)
        let deathAction = SKAction.animate(with:deathFrames,
                                           timePerFrame: 0.15,
                                           resize: false,
                                           restore: false)

        run(deathAction,withKey: deathActionKey)
    }

    private func shouldMove(_ currentPosition: CGPoint, touchPosition: CGPoint) -> Bool {
        return abs(currentPosition.x - touchPosition.x) > self.frame.width / 2 ||
        abs(currentPosition.y - touchPosition.y) > self.frame.height / 2
    }

}
```

HealthNode.swift:

```
import SpriteKit


class HealthNode: SKSpriteNode {

    func configurePhysicsBody() {
        self.zPosition = 150
        physicsBody = .init(circleOfRadius: self.frame.width / 2)
        physicsBody?.categoryBitMask = PhysicsCategory.health
        physicsBody?.contactTestBitMask  = PhysicsCategory.lootZone
        physicsBody?.collisionBitMask = 0
    }
}
```

MagnetNode.swift:

```
import SpriteKit

class MagnetNode: SKSpriteNode {

    func configurePhysicsBody() {
```

```
        self.zPosition = 150
        physicsBody = .init(circleOfRadius: self.frame.width / 2)
        physicsBody?.categoryBitMask = PhysicsCategory.magnet
        physicsBody?.contactTestBitMask  = PhysicsCategory.lootZone
        physicsBody?.collisionBitMask = 0
    }
}
```

LevelXpNode.swift:

```
import SpriteKit


class LevelXpNode: SKSpriteNode {

    var xpValue: CGFloat = 30
    var xpGainActionKey = "xpGainAction"
    var isCollected = false

    init(size: CGSize) {
        super.init(texture: .init(imageNamed: Textures.xpCrystal),
                   color: .purple,
                   size: size)
        setPhysicsBody()
    }

    func collectedXP(by player: Player) {
        self.run(.sequence([
            .move(by: .init(dx: -100 * cos(player.directionAngle),
                            dy: -100 * sin(player.directionAngle)),
                          duration: 0.2),
            .run { [weak self] in
                guard let self = self else { return }
                self.isCollected = true
            }
        ]),
                 withKey: xpGainActionKey)
    }

    func update(playerPosition: CGPoint) {
        if isCollected {
            let angle = atan2(playerPosition.y - self.position.y, playerPosition.x -
self.position.x)
            self.zRotation = angle - .pi / 2
            self.position.x += 5 * cos(angle)
            self.position.y += 5 * sin(angle)
        }
    }

    func pauseAction(_ pause: Bool) {
        if let xpGainAction = action(forKey: xpGainActionKey) {
            if pause {
                xpGainAction.speed = 0
            } else {
                xpGainAction.speed = 1
            }
        }
    }

    func setPhysicsBody() {
        self.physicsBody = .init(rectangleOf: frame.size)
```

```
        physicsBody?.allowsRotation = false
        physicsBody?.categoryBitMask = PhysicsCategory.xpNode
        physicsBody?.collisionBitMask = 0
        physicsBody?.contactTestBitMask = PhysicsCategory.player |
PhysicsCategory.lootZone
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }


}
```

Hud.swift:

```
import SpriteKit
import Foundation

class Hud: SKNode {

    var coinLabel: SKLabelNode
    var killsLabel: SKLabelNode
    var killsFrame: SKSpriteNode
    var coinFrame: SKSpriteNode
    var timeLabel: SKLabelNode

    var levelProgressBar : SKCropNode
    var emptyLevelProgressBar: SKSpriteNode

    var backButton:SKSpriteNode

    var selectedButton: SKSpriteNode?

    var backButtonHandler: (() -> Void)?

    var survivalTime: TimeInterval = 0.0
    var coins: Int = 0
    var kills: Int = 0
    var currentLevelProgress: CGFloat = 0
    var levelProgress: CGFloat = 100

    init(size: CGSize) {
        let baseImage = Textures.progressFrame
        let coverImage = Textures.progressIndicator
        let healthBarSize: CGSize = .init(width: size.width, height: size.height * 0.05)
        let filledImage  = SKSpriteNode(imageNamed: coverImage)
        filledImage.size.width = healthBarSize.width
        filledImage.size.height = healthBarSize.height

        levelProgressBar = SKCropNode()
        levelProgressBar.addChild(filledImage)
        levelProgressBar.maskNode = SKSpriteNode(color: .white,
                                                 size: CGSize(width:
filledImage.size.width * 2,
                                                              height:
filledImage.size.height))
        levelProgressBar.maskNode?.position = .init(x: -filledImage.size.width / 2, y: 0)
        levelProgressBar.maskNode?.xScale = 0
        levelProgressBar.zPosition = 210
```

```
    emptyLevelProgressBar = SKSpriteNode.init(imageNamed: baseImage)
    emptyLevelProgressBar.size.width = healthBarSize.width
    emptyLevelProgressBar.size.height = healthBarSize.height
    emptyLevelProgressBar.name = "levelProgress"
    emptyLevelProgressBar.zPosition = 200

    coinLabel = .init(fontNamed: GameFont)
    coinLabel.text = "\(coins)"
    coinLabel.fontColor = .white
    coinLabel.fontSize = 15
    coinLabel.zPosition = 100

    killsLabel = .init(fontNamed: GameFont)
    killsLabel.text = "\(kills)"
    killsLabel.fontColor = .white
    killsLabel.fontSize = 15
    killsLabel.zPosition = 100

    coinFrame = .init(texture: .init(imageNamed: Textures.coinFrame),
                      size: .init(width: size.width * 0.2,
                                  height: size.width * 0.07))
    killsFrame = .init(texture: .init(imageNamed: Textures.killFrame),
                      size: .init(width: size.width * 0.2,
                                  height: size.width * 0.07))

    backButton = .init(texture: .init(imageNamed: Textures.backButton),
                       size: .init(width: size.width * 0.1,
                                   height: size.width * 0.1))

    timeLabel = .init(fontNamed: GameFont)
    timeLabel.fontSize = size.width * 0.1
    timeLabel.fontColor = .white
    super.init()

    coinFrame.position = .init(x: -size.width * 0.35,
                               y: size.height * 0.45)

    killsFrame.position = .init(x: -size.width * 0.35,
                                y: coinFrame.position.y - killsFrame.frame.height - 5)

    coinLabel.position = .init(x: coinFrame.frame.width * 0.15,
                               y: -coinFrame.frame.height * 0.25)
    killsLabel.position = .init(x: killsFrame.frame.width * 0.15,
                               y: -killsFrame.frame.height * 0.25)

    backButton.position = .init(x: size.width * 0.45 - backButton.frame.width / 2,
                                y: size.height * 0.4625 - backButton.frame.height / 2)
    levelProgressBar.position = .init(x: 0, y: size.height * 0.365)
    emptyLevelProgressBar.position = .init(x: 0, y: size.height * 0.365)
    timeLabel.position = .init(x: 0, y: size.height * 0.4)

    addChild(levelProgressBar)
    addChild(emptyLevelProgressBar)
    addChild(backButton)
    addChild(coinFrame)
    addChild(killsFrame)
    addChild(timeLabel)

    coinFrame.addChild(coinLabel)
    killsFrame.addChild(killsLabel)
}
```

```
    func updateTime(_ dt: TimeInterval) {
        survivalTime += dt
        timeLabel.text = stringFromTimeInterval(interval: survivalTime)
    }

    func updateCoins(with value: Int) {
        coins += value
        coinLabel.text = "\(formatNumber(coins))"
    }

    func updateKills(with value: Int) {
        kills += value
        killsLabel.text = "\(kills)"
    }

    func updateLevelProgressBar(with value: CGFloat, completion: @escaping (Bool) -> Void)
{

        guard currentLevelProgress <= levelProgress else {
            return
        }

        currentLevelProgress += value
        let progress = currentLevelProgress / levelProgress
        if progress > 1 {
            levelProgressBar.maskNode?.run(.scaleX(to: progress, duration: 0.2)) {
                completion(true)
            }
        } else {
            levelProgressBar.maskNode?.run(.scaleX(to: progress, duration: 0.2)) {
                completion(false)
            }
        }
    }

    func resetLevelProgressBar() {
        currentLevelProgress = 0
        levelProgress *= 1.3
        levelProgressBar.maskNode?.run(.scaleX(to: 0, duration: 0.2))
    }


    func touchBegan(_ touch: UITouch) {
        if backButton.contains(touch.location(in: self)) {
            selectedButton = backButton
            selectedButton?.alpha = 0.5
        }
    }
    //MARK: - Handling touches
    func touchMoved(_ touch: UITouch) {
        guard let selectedButton = selectedButton else {
            return
        }
        if selectedButton.contains(touch.location(in: self)) {
            selectedButton.alpha = 0.5
        } else {
            selectedButton.alpha = 1.0
        }
    }
```

```
    func touchEnded(_ touch: UITouch) {
        guard let selectedButton = selectedButton else {
            return
        }
        if selectedButton == backButton {
            if backButton.contains(touch.location(in: self)) {
                handleBackButtonAction()
            }
        }
        self.selectedButton?.alpha = 1.0
        self.selectedButton = nil
    }

    func handleBackButtonAction() {
        backButtonHandler?()
    }

    func formatNumber(_ n: Int) -> String {
        let num = abs(Double(n))
        let sign = (n < 0) ? "-" : ""

        switch num {
        case 1_000_000_000...:
            var formatted = num / 1_000_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)B"

        case 1_000_000...:
            var formatted = num / 1_000_000
            formatted = formatted.reduceScale(to: 1)
            return "\(sign)\(formatted)M"

        case 1_000...:
            var formatted = num / 1_000
            formatted = formatted.reduceScale(to: 2)
            return "\(sign)\(formatted)K"

        case 0...:
            return "\(n)"

        default:
            return "\(sign)\(n)"
        }
    }

    func stringFromTimeInterval(interval: TimeInterval) -> String {

      let ti = NSInteger(interval)

      let seconds = ti % 60
      let minutes = (ti / 60) % 60

      return String(format: "%0.2d:%0.2d",minutes,seconds)
    }


    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}
```

LevelUpgradeHud.swift:

```
import SpriteKit

enum WeaponType: String, CaseIterable {
    case mainWeapon,shuriken="weight",lightning,poisonousField,ball,drill,rocket
}

class LevelUpgradeHud: SKNode {

    var background: SKShapeNode

    var selectButton: SKSpriteNode
    var selectLabel: SKSpriteNode

    var upgrades: [SKSpriteNode] = []
    var upgradeDescription: SKLabelNode
    var weaponUpgrades: [WeaponType]  = []
    var selectedUpgradeIndex: Int? = nil

    var selectedUpgrade: SKSpriteNode?
    var selectedButton: SKSpriteNode?
    var selectButtonHandler: ((WeaponType) -> Void)?

    var upgradeEffects: [SKEmitterNode] = []


    init(size:CGSize) {

        background = .init(rectOf: size)
        background.fillColor = .black
        background.strokeColor = .black
        background.alpha = 0.7
        background.zPosition = 100

        selectButton = .init(texture: .init(imageNamed: "select_button"),
                             size: .init(width: size.width * 0.5,
                                         height: size.height * 0.1))
        selectButton.alpha = 0.6
        selectButton.zPosition = 110

        selectLabel = .init(texture: .init(imageNamed: Textures.upgradeFrame),
                            size: .init(width: size.width, height: size.width * 0.15))
        selectLabel.zPosition = 110

        for i in 0..<3 {
            let upgrade: SKSpriteNode = .init(texture: nil,
                                              color: .red,
                                              size: .init(width: size.width * 0.3,
                                                          height: size.width * 0.3))
            upgrade.position = .init(x: size.width * (-0.33 + CGFloat(i) * 0.33),
                                     y: 0)
            upgrade.zPosition = 110
            upgrades.append(upgrade)
        }

        upgradeDescription = .init(fontNamed: GameFont)
        upgradeDescription.zPosition = 110
        upgradeDescription.numberOfLines = 5
        upgradeDescription.position.y = -size.height * 0.25
```

```
        super.init()

        selectButton.position.y = -size.height * 0.4
        selectLabel.position.y = size.height * 0.3

        for i in 0..<3 {
            let emitter = SKEmitterNode(fileNamed: Emitter.levelUp)!
            emitter.position = selectLabel.position
            emitter.zPosition = 105
            emitter.particleColorBlendFactor = 1.0
            emitter.particleColorSequence = nil
            switch i {
            case 0:
                emitter.particleColor = .orange
            case 1:
                emitter.particleColor = .green
            default:
                emitter.particleColor = .blue
            }
            upgradeEffects.append(emitter)
        }
        upgrades.forEach {
            $0.setScale(0)
            addChild($0)
        }

        selectButton.setScale(0)
        selectLabel.setScale(0)
        background.setScale(0)

        addChild(selectButton)
        addChild(selectLabel)
        addChild(background)
        addChild(upgradeDescription)
    }

    func showHud() {
        generateUpgrades()
        background.setScale(1)
        selectLabel.run(.scale(to: 1, duration: 0.3))
        selectButton.run(.scale(to: 1, duration: 0.3))
        upgrades.forEach {
            $0.run(.scale(to: 1, duration: 0.3))
            $0.alpha = 0.5
        }

        upgradeEffects.forEach {
            addChild($0)
        }
    }

    func hideHud() {
        weaponUpgrades.removeAll()
        background.setScale(0)
        selectLabel.run(.scale(to: 0, duration: 0.3))
        selectButton.run(.scale(to: 0, duration: 0.3))
        upgrades.forEach {
            $0.run(.scale(to: 0, duration: 0.3))
        }
        upgradeDescription.removeFromParent()
        upgradeEffects.forEach {
```

```
            $0.removeFromParent()
        }
    }

    func generateUpgrades() {
        for _ in 0..<3 {
            while true {
                if let randomType = WeaponType.allCases.randomElement() {
                    if !weaponUpgrades.contains(randomType) {
                        weaponUpgrades.append(randomType)
                        break
                    }
                }
            }
        }
        for i in 0..<3 {
            upgrades[i].texture = .init(imageNamed:
"\(weaponUpgrades[i].rawValue)_upgrade")
        }
    }


    //MARK: - Handling touches

    func touchBegan(_ touch: UITouch) {
        if selectButton.contains(touch.location(in: self)), selectedUpgradeIndex != nil {
            selectedButton = selectButton
            selectedButton?.alpha = 0.5
        } else if let selectedUpgrade = upgrades.first(where: {
$0.contains(touch.location(in: self))}) {
            self.selectedUpgrade = selectedUpgrade
            selectedUpgrade.alpha = 0.5
        }
    }

    func touchMoved(_ touch: UITouch) {
        if let selectedUpgrade = selectedUpgrade {
            if selectedUpgrade.contains(touch.location(in: self)) {
                selectedUpgrade.alpha = 0.5
            } else {
                selectedUpgrade.alpha = 1.0
            }
        } else if let selectedButton = selectedButton {
            if selectedButton.contains(touch.location(in: self)) {
                selectedButton.alpha = 0.5
            } else {
                selectedButton.alpha = 1.0
            }
        }
    }

    func touchEnded(_ touch: UITouch) {
        if let selectedUpgrade = selectedUpgrade {
            if selectedUpgrade.contains(touch.location(in: self)) {
                selectedUpgradeIndex = upgrades.firstIndex(of: selectedUpgrade)
                upgradeDescription.text = updateText(type:
weaponUpgrades[selectedUpgradeIndex!])
                selectButton.alpha = 1
            }
            selectedUpgrade.alpha = 1
            upgrades.forEach{
```

```
                if selectedUpgrade != $0 {
                    $0.alpha = 0.5
                }
            }

        } else if let selectedButton = selectedButton {
            if selectedButton.contains(touch.location(in: self)) {
                handleSelectButtonAction()
            }
            selectedButton.alpha = 1.0
        }
        selectedButton = nil
        selectedUpgrade = nil
    }

    func handleSelectButtonAction() {
        guard let selectedUpgradeIndex = selectedUpgradeIndex else {
            return
        }
        selectButtonHandler?(weaponUpgrades[selectedUpgradeIndex])
    }

    func updateText(type: WeaponType) -> String {
        switch type {
        case .mainWeapon:
             return "Increase number \n of bullets"
        case .shuriken:
            return "Increase number \n of spinning weights"
        case .lightning:
            return "Increase number \n of strikes"
        case .poisonousField:
            return "Increase radius \n of field"
        case .ball:
            return "Increase damage"
        case .drill:
            return "Increase number \n of drills"
        case .rocket:
            return "Increase damage"
        }
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

}
```

Camera.swift:

```
import SpriteKit

class Camera: SKCameraNode {

    private var destination : CGPoint!

    private let easing : CGFloat = 0.9

    public func updatePosition(point: CGPoint) {
        position = point
        destination = point
    }
```

```
    func addBorder(frame: CGRect) {
        let border = SKShapeNode(rectOf: frame.size)
        border.fillColor = .clear
        border.physicsBody = .init(edgeLoopFrom: .init(x: -frame.width / 2,
                                                       y: -frame.height / 2,
                                                       width: frame.width,
                                                       height: frame.height))
        border.physicsBody?.friction = 0
        border.physicsBody?.categoryBitMask = PhysicsCategory.border
        border.physicsBody?.contactTestBitMask = PhysicsCategory.drill |
PhysicsCategory.ball
        border.physicsBody?.collisionBitMask = PhysicsCategory.drill |
PhysicsCategory.ball
        addChild(border)
    }

    public func setDestination(destination : CGPoint) {
        self.destination = destination
    }

    public func update() {
        let distance = sqrt(pow((destination.x - position.x), 2) + pow((destination.y -
position.y), 2))

        if(distance > 1) {
            let directionX = (destination.x - position.x)
            let directionY = (destination.y - position.y)

            position.x += directionX * easing
            position.y += directionY * easing
        } else {
            position = destination
        }
    }
}
```

Joystick.swift:

```
import SpriteKit

//MARK: AnalogJoystickData
public struct AnalogJoystickData: CustomStringConvertible {
    var velocity = CGPoint.zero,
    angular = CGFloat(0)

    mutating func reset() {
        velocity = CGPoint.zero
        angular = 0
    }

    public var description: String {
        return "AnalogStickData(velocity: \(velocity), angular: \(angular))"
    }
}

//MARK: - AnalogJoystickComponent
open class AnalogJoystickComponent: SKSpriteNode {
    private var kvoContext = UInt8(1)
    var borderWidth = CGFloat(0) {
        didSet {
```

```
            redrawTexture()
        }
    }

    var borderColor = UIColor.black {
        didSet {
            redrawTexture()
        }
    }

    var image: UIImage? {
        didSet {
            redrawTexture()
        }
    }

    var diameter: CGFloat {
        get {
            return max(size.width, size.height)
        }

        set(newSize) {
            size = CGSize(width: newSize, height: newSize)
        }
    }

    var radius: CGFloat {
        get {
            return diameter * 0.5
        }

        set(newRadius) {
            diameter = newRadius * 2
        }
    }

    //MARK: - DESIGNATED
    init(diameter: CGFloat, color: UIColor? = nil, image: UIImage? = nil) {
        super.init(texture: nil, color: color ?? UIColor.black, size: CGSize(width:
diameter, height: diameter))
        addObserver(self, forKeyPath: "color", options: NSKeyValueObservingOptions.old,
context: &kvoContext)
        self.diameter = diameter
        self.image = image
        redrawTexture()
    }

    required public init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    deinit {
        removeObserver(self, forKeyPath: "color")
    }

    open override func observeValue(forKeyPath keyPath: String?, of object: Any?, change:
[NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {
        redrawTexture()
    }

    private func redrawTexture() {
```

```
        guard diameter > 0 else {
            print("Diameter should be more than zero")
            texture = nil
            return
        }

        let scale = UIScreen.main.scale
        let needSize = CGSize(width: self.diameter, height: self.diameter)
        UIGraphicsBeginImageContextWithOptions(needSize, false, scale)
        let rectPath = UIBezierPath(ovalIn: CGRect(origin: CGPoint.zero, size: needSize))
        rectPath.addClip()

        if let img = image {
            img.draw(in: CGRect(origin: CGPoint.zero, size: needSize), blendMode: .normal,
alpha: 1)
        } else {
            color.set()
            rectPath.fill()
        }

        let needImage = UIGraphicsGetImageFromCurrentImageContext()!
        UIGraphicsEndImageContext()
        texture = SKTexture(image: needImage)
    }
}

//MARK: - AnalogJoystickSubstrate
open class AnalogJoystickSubstrate: AnalogJoystickComponent {
    // coming soon...
}

//MARK: - AnalogJoystickStick
open class AnalogJoystickStick: AnalogJoystickComponent {
    // coming soon...
}

//MARK: - AnalogJoystick
open class AnalogJoystick: SKNode {
    var trackingHandler: ((AnalogJoystickData) -> ())?
    var beginHandler: (() -> Void)?
    var stopHandler: (() -> Void)?
    var substrate: AnalogJoystickSubstrate!
    var stick: AnalogJoystickStick!
    private var tracking = false
    private(set) var data = AnalogJoystickData()

    var disabled: Bool {
        get {
            return !isUserInteractionEnabled
        }

        set(isDisabled) {
            isUserInteractionEnabled = !isDisabled

            if isDisabled {
                resetStick()
            }
        }
    }
```

```
    var diameter: CGFloat {
        get {
            return substrate.diameter
        }

        set(newDiameter) {
            stick.diameter += newDiameter - diameter
            substrate.diameter = newDiameter
        }
    }

    var radius: CGFloat {
        get {
            return diameter * 0.5
        }

        set(newRadius) {
            diameter = newRadius * 2
        }
    }

    init(substrate: AnalogJoystickSubstrate, stick: AnalogJoystickStick) {
        super.init()
        self.substrate = substrate
        substrate.zPosition = 0
        addChild(substrate)
        self.stick = stick
        stick.zPosition = substrate.zPosition + 1
        addChild(stick)
        disabled = false
        let velocityLoop = CADisplayLink(target: self, selector: #selector(listen))
        velocityLoop.add(to: RunLoop.current, forMode: RunLoop.Mode(rawValue:
RunLoop.Mode.common.rawValue))
    }

    convenience init(diameters: (substrate: CGFloat, stick: CGFloat?), colors: (substrate:
UIColor?, stick: UIColor?)? = nil, images: (substrate: UIImage?, stick: UIImage?)? = nil)
{
        let stickDiameter = diameters.stick ?? diameters.substrate,
        jColors = colors ?? (substrate: nil, stick: nil),
        jImages = images ?? (substrate: nil, stick: nil),
        substrate = AnalogJoystickSubstrate(diameter: diameters.substrate, color:
jColors.substrate, image: jImages.substrate),
        stick = AnalogJoystickStick(diameter: stickDiameter / 2, color: jColors.stick,
image: jImages.stick)
        self.init(substrate: substrate, stick: stick)
    }

    convenience init(diameter: CGFloat, colors: (substrate: UIColor?, stick: UIColor?)? =
nil, images: (substrate: UIImage?, stick: UIImage?)? = nil) {
        self.init(diameters: (substrate: diameter, stick: nil), colors: colors, images:
images)
    }

    required public init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    @objc func listen() {

        if tracking {
```

```
            trackingHandler?(data)
        }
    }

    //MARK: - Overrides
    open override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {

        if touches.first != nil {//, stick == atPoint(touch.location(in: self)) {
            tracking = true
            beginHandler?()
        }
    }

    open override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
        for touch: AnyObject in touches {
            let location = touch.location(in: self)

            guard tracking else {
                return
            }

            let maxDistantion = substrate.radius,
            realDistantion = sqrt(pow(location.x, 2) + pow(location.y, 2)),
            needPosition = realDistantion <= maxDistantion ? CGPoint(x: location.x, y:
location.y) : CGPoint(x: location.x / realDistantion * maxDistantion, y: location.y /
realDistantion * maxDistantion)
            stick.position = needPosition
            data = AnalogJoystickData(velocity: needPosition, angular: -
atan2(needPosition.x, needPosition.y))
        }
    }
    open override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        resetStick()
    }
    open override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
        print("cancel")
        resetStick()
    }
    // CustomStringConvertible protocol
    open override var description: String {
        return "AnalogJoystick(data: \(data), position: \(position))"
    }
    // private methods
    func resetStick() {
        tracking = false
        let moveToBack = SKAction.move(to: CGPoint.zero, duration: TimeInterval(0.1))
        moveToBack.timingMode = .easeOut
        stick.run(moveToBack)
        data.reset()
        stopHandler?();
    }
}

typealias 🕹 = AnalogJoystick
```