МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка гри у стилі Match3 мовою Java

Виконав студент групи К-20і
спеціальності 122 Комп'ютерні науки
Гуляков Олексій Владиславович

Керівник Штефан
Наталія Зінов'ївна

Консультант д.т.н., проф.
Казакова Надія Феліксівна

Рецензент д.т.н., професор
Мещеряков Володимир Іванович

Одеса 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

Jazelle DBX – Direct Bytecode eXecution

JVM – Java Virtual Machine

SWT – Standard Widget Toolkit

UI – User Interface

UE – Unreal Engine

UML – Unified Modeling Language

WebGL – Web Graphics Library

ГК-проекти – Гіпер-казуальні проекти

Спрайти – це зображення, які представляють активи гри

Android – операційна система

Casual and Puzzle – казуальні ігри

Corona SDK – двигун для створення 2D ігор

C++ – мова програмування

C# – мова програмування

Java – мова програмування

iOS – операційна система

Godot – відкритий кросплатформний 2D и 3D ігровий движок

libGDX – двигун для розробки ігрових додатків

Lua – скриптова мова

Python – мова програмування

Unity – двигун для розробки ігрових додатків

Use-Case – варіант використання

ВСТУП

Збільшення доходів від мобільних ігор продовжує випереджати зростання доходів від продажу ПК-ігор. Як свідчать статистики, користувачі полюбляють проводити свій вільний час за нескладною грою на мобільному пристрої. Казуальні ігри (Casual and Puzzle) – це ігри для широкого кола користувачів, багато з яких навіть не вважають себе геймерами. У «Casual and Puzzle» немає суперскладних правил – зазвичай суть гри лежить на поверхні, і всім зрозуміла без будь-яких інструкцій [1].

Ігри-головоломки являють собою жанр ігор, які були деякий час широко популярні завдяки своїй простоті. Серед Ігри Match 3 все ще випускаються розробниками в усьому світі і часто знаходяться на першому місці в списках безкоштовних і платні ігри в Google Play Store або Apple App Store.

Ігри в «Match3» досить популярні в жанрі головоломок. Вони, як правило, невимушені, візуально привабливі і дають гравцеві відчуття задоволення після проходження рівня. Ця простота чудово підходить для звичайних і лінивих геймерів, які не люблять зайвих труднощів. Такі ігри, як Candy Crush Saga, Angry Birds Match, Frozen Free Fall і Puzzle&Dragons, є лише деякими з величезних архівів ігор Match3.

Мета дипломної роботи – розробка ігрового додатку у стилі «Match3» з використанням графічних спрайтів.

Загальні характеристики кваліфікаційної роботи:

− повний обсяг сторінок пояснювальної записки – 55;

− кількість рисунків – 30;

− кількість посилань – 17.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Опис предметної області

Існує кілька категорій гравців: хардкорні, мідкорні та казуальні. Основний упор розробників ігор завжди був спрямований на них. При цьому дані категорії є дуже невеликим відсотком людей в обсязі всієї планети. ГК-проекти (Гіпер-казуальні) стали орієнтуватися на так званих нон-геймерів, тобто на основну частину людей, які в принципі ніколи не грали в ігри.

Стандартні категорії гравців мають закладені патерни поведінки, які проектуються в базові, звичні їм механіки. У цей час у нон-геймерів вони взагалі відсутні. Вони намагаються перекласти на подібні проекти патерни, які були отримані в добре знайомих додатках, наприклад, WhatsApp, Tinder або Instagram.

Тому одним із завдань розробників ГК ігор є створення такого геймплею, який підлаштовується під патерни зрозумілі нон-геймерам. Важливо розуміти, що нон-геймери це максимально широка аудиторія, відповідно саме вони роблять продукт масовим.

Основою ГК ігрових продуктів є геймплей. Тут йдеться саме про таке поняття як core gameplay, без будь-якої мета гри. Людина, яка грає в ці ігри, повторює тисячі разів одні й ті самі дії. Це можна порівняти з перебиранням чіток. Геймплей у багатьох проектах медитативний, оскільки людина концентрується однією дії.

Витоки ігор «Match 3» можна простежити до «Tetris» російського розробника Олексія Пажитнова в 1984 році і «Chain Shot» японського виробника ігор Куніакі Морібе у 1985 році. Перша гра «Match 3», випущена для DOS у 1994 році, дала початок шутерам з бульбашками. Ігри «Match 3» стали популярними протягом 2000-х років у формі казуальних ігор, які поширювалися або в які грали в Інтернеті, зокрема серія ігор «Bejeweled».

Відтоді вони залишаються популярними, і гра «Candy Crush Saga» стала однією з найпопулярніших казуальних ігор у всьому світі.

Розглядаючи «Mobile Legends Bang Bang» як приклад, «Moonton» тримає базу гравців залученою до кількох подій на тиждень. Скрині оновлюються кожні 4 години, кілька щоденних нагород за вхід виходять зверху. Крім того у гравця є можливість виконувати ігрові квести різної складності, щоб отримати ще кращу видобуток.

Повторювані бонуси на поповнення спонукають гравців витрачати багато грошей на нібито кращі пропозиції, наприклад отримання певних ексклюзивних скарбів, такі як епічні скіни, інвестувавши понад 100 доларів.

Особливістю ігор «Match-3» є те, що всі представники жанру використовують майже однакову ігрову механіку. Основне завдання гравця – складати комбінації з 3 однакових предметів (цукерок, дорогоцінних каменів, фруктів тощо), випадково розташованих на ігровому полі, і формувати різноманітні комбо, щоб заробити більше очок [2].

Незважаючи на те, що світові хіти казуальних ігор не мають жодної з властивостей ігор «Match-3» (закручений сюжет, налаштування персонажів, контент, який можна розблокувати тощо), мільйони людей по всьому світу грають у ці ігри щодня годинами.

1.2 Характеристика об’єкту розробки

Є декілька основних вимог при проектуванні ігор «Match-3», які слід враховувати:

1. Рівні – завдання подолати рівні та відчуття виконаного завдання це те, що викликає найбільше позитивних емоцій.
2. Візуальні зображення – багато гравців не проти того, що деякі ігри приблизно на одну і ту ж тему (цукерки), вони насолоджуються відчуттям, яке вони мають від гарного мистецтва та цікавих рівнів.

3. Історія – зараз є багато ігор у ряди 3, які додають сюжетний шар поверх відповідної механіки. «Gardenscapes» є чудовим прикладом.
4. Мета-гра – деякі ігри використовують механіку відповідності як просту, але дуже гнучку механіку для основного ігрового процесу, ігри «Match-3» є чудовим прикладом [3].

1.3 Огляд аналогів

При проектуванні інформаційної системи слід враховувати цілі, для яких створюється програмний продукт, умови, в яких експлуатуватиметься система, вимоги майбутніх користувачів

Для головоломок у «Match 3» дизайн рівнів – це основа. При проектуванні слід почати з вивчення конкуренції – яке найкраще співвідношення нових елементів на рівні, зовнішній вигляд елементів для найпопулярніших ігор у різних регіонах та цільових аудиторіях, а також найпопулярніші чи модні механіки.

Маючи на увазі ці висновки, слід об’єднати всі ідеї та відфільтрувати кілька найбільш перспективних механізмів та елементів. Подальший список властивостей для кожного елемента буде відображати початкові концепції, на основі яких будуть створені графічні елементи. Деякі рівні мають бути важкими, інші – легкими і просто добре виглядати [4].

Для початку слід розглянути декілька топових аналогів, щоб з’ясувати головні від’ємності ігор. Першим аналогом обрана гра «Candy Crush Saga» – легендарна гра-головоломка, яку люблять мільйони гравців у всьому світі. Гравцю необхідно перемикайтися та підбирати цукерки в цій смачній пригоді-головоломці, щоб перейти на наступний рівень і отримати солодке відчуття перемоги.

Слід вирішувати головоломки за допомогою швидкого мислення та розумних ходів, та отримувати винагороду смачними райдужними каскадами та смачними комбінаціями цукерок.

Гра містить тисячу рівнів, в яких змінюється швидкість проходження та кількість елементів поля (рис. 1).

Рисунок 1 – Скріншот гри «Candy Crush Saga»

До особливостей «Candy Crush Saga» можна віднести наступні:

– тисячі найкращих рівнів і головоломок у «Цукерковому королівстві», а кожні 2 тижні їх додається більше;
– багато способів збирати нагороди. Для цього слыд переглядати щодня та оберти щоденне колесо підсилювача, щоб отримувати безкоштовні смачні нагороди, а також брати участь у обмежених за часом випробуваннях, щоб заробити прискорювачі, які допоможуть піднятися на рівень;
– декілька ігрових режимів, які дозволяють синхронізувати гру між пристроями та отримати доступ до повних функцій гри при підключенні до Інтернету;

Candy Crush Saga безкоштовна, але додаткові предмети в грі вимагають оплати.

«Homescapes» – це відома безкоштовна гра, як «Candy Crush» від Playrix. У грі поєднується ігровий процес симуляції життя із традиційними

головоломками в стилі «Match-3» (рис. 2). «Homescapes» був випущений ще в 2017 році як продовження успішного «Gardenscape»s, і менш ніж за два місяці після виходу гру завантажили 35 мільйонів разів з 9 мільйонами активних користувачів щомісяця. У 2022 році «Homescapes» залишається класичною грою в ряд 3 на ринку мобільних ігор.

Рисунок 2 – Скріншоти рівнів гри «Homescapes»

Головного героя гри звуть дворецький Остін. Він прагне повністю переобладнати особняк своїх батьків. Щоб виконати завдання в особняку, гравцю потрібно набрати бонуси за проходження – це надає гравцю можливість розблоковувати додаткові елементи декору будинку. Розробники включили в гру багато сюжетних сцен, що робить її дуже цікавою на будь-якому рівні. «Homescapes» є багатофункціональним симулятором життя та механікою «Match-3». Гра в цілому включає 162 рівні [5].

«Royal Match» це безкоштовна мобільна гра, яка не тільки має гарний дизайн, але й пропонує надзвичайно швидкий ігровий процес (рис. 3). Це також одна з найкращих ігор, як «Homescapes», через декораційний сюжет, але

тут замість ремонту гравці отримують порожню кімнату або парк, і вони повинні грати в головоломки 3-го рівня, щоб отримати очки досвіду та використовувати їх для додавання меблів/обладнання. Коли вони закінчують роботу з однією кімнатою, вони швидко переходять до наступної.

Рисунок 3 – Скріншоти гри «Royal Match»

Гра називається «Royal Match», тому що гравці в основному допомагають доброзичливому королю Роберту відновити його замок. У цій грі так багато функцій. Окрім звичайного Match 3-рівнів, час від часу гравці отримують унікальну гру-головоломку під назвою «The King’s Nightmare». У гравця є рівно 60 секунд, щоб вирішити її, інакше поганий сон короля стане реальністю.

Бустери «Royal Match» досить звичайні на початку гри. Є класичні, як в «Candy Crush» і «Jelly Splash» – бомби (TNT), горизонтальні та вертикальні стрілки, легка кулька. Є також кілька новачків – капелюх шута, який перемішує все на дошці, гармата, яка очищає всі об'єкти в стовпці. Стріла

робить те ж саме, що і гармата, тільки горизонтально, а королівський молот, який очищає лише один об’єкт.

Гравці також можуть виграти подарунок дворецького – вони починають гру з кількома прискорювачами від дворецького короля, що на деякий час полегшує роботу. Перешкоди в «Royal Match» досить складні – королівські яйця та королівські страви потрібно видалити, щоб пройти рівень [5].

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

– різноманіття графічних елементів ігрового поля та основних спрайтів;
– нескладні рівні;
– можливість грати оффлайн;
– звукові ефекти до подій у грі.

1.4 Вимоги до проекту

Щоб гра з трійкою була цікавою, гравці повинні відчути весь спектр емоцій. Сік гри виходить із балансу між складним і легким, цікавим і нудним. Точне регулювання цього балансу дозволяє гравцям відчувати яскраві емоції, наприклад, від перемоги останнього ходу. Крива витрат може змінюватися в синусоїді, як і доходи, або одна з них може бути лінійною. У цьому випадку гравець буде відчувати дефіцит в одні періоди, надлишок в інші.

Створюючи такий потік, розробник гри може впливати на почуття гравця, тому що іноді йому доводиться напружуватися, а потім отримує винагороду. Найкраща практика – пропонувати 2-3 емоції за сесію гри. Але бувають складні рівні, коли гравець відчуває єдину лють протягом усієї сесії: лють змушує платити за подолання перешкод, і такі прийоми можна використовувати для монетизації. Ці аспекти необхідно враховувати під час розробки сценарію рівнів [6].

1.5 Вибір засобів розробки

Розробка ігор ніколи не була простішою. З розкішшю різноманітних движків і фреймворків можна вибрати майже з усіх популярних мов програмування.

Вибір правильного двигуна є однією з найважливіших речей для розробника. На рисунку є різні движки, а саме Unity, Unreal Engine, GameMaker та багато інших. Більше половини всіх нових мобільних ігор створюються в Unity. Движок надає чудові можливості для графіки, керування монетизацією та інших функцій. Його функція Adaptive Performance обіцяє плавну роботу на різних пристроях.

Unreal Engine підтримує C++, Godot з відкритим кодом використовує Python, з libGDX можна кодувати на Java, і для вас буде доступний Unity C#. Велика кількість альтернатив створює плутанину в умах новачків, оскільки зробити правильний вибір із самого початку – завдання не з легких [7].

Unreal Engine 4 має спеціальний інтерфейс, який полегшує кодування початківцям, плюс його бібліотека пропонує ряд активів для використання у вашій грі. Використовувати його можна безкоштовно до певного моменту, після чого ви повинні заплатити 5% роялті [4].

Один із самих популярних рухів сьогодні – Unreal Engine, який розробляється і підтримується компанією Epic Games. «Епіки» сьогодні є одними з столових індустрій, особливо в технологічній частині. Тому рівень якості та кількість фічей, які пропонуються розробниками, є багатьма конкурентами.

Наступне перевагу UE – він безкоштовний. Є певні умови та роялти, але весь інструментарій доступний відразу і в повному обсязі для створення гри AAA-качества. Широкий функціонал розуміє непростий інтерфейс, особливо для новичків (рис. 4).

Рисунок 4 – Приклад етапу розробки гри у Unreal Engine

Зараз великий обсяг роботи при створенні ігор ложиться на геймдизайн, креативний дизайн і створення контенту. У Unreal є все необхідне для дизайнерів, артистів і програмістів, при чому весь вихідний код відкритий – любу частину можна кастомізувати під проектом.

Це робить систему більш гнучкою. За рахунок відкритості в UE великі ком'юніти, а офіційне управління найденними багов самим спільнотою швидко потрапляє в оновлення.

Ще одним плюсом Unreal Engine є мультиплеєрна гра «із коробки». Фактично, після запуску редактора кількома кліками можна створити проект, в якому вже є сетева гра. Навчальний ріст багатокористувацьких ігор і ставки великих компаній в онлайн-режимах з нескількими гравцями, це серйозні переваги, яких, наприклад, немає в Unity [8].

Unity – межплатформенная среда розробки комп’ютерних ігор, розроблена американською компанією Unity Technologies (рис. 5). Unity дозволяє створювати додатки, які працюють на понад 25 різних платформах, включають персональні комп’ютери, ігрові консолі, мобільні пристрої, інтернет-приложения та інші. Unity є найпопулярнішим ігровим движком у

світі. Він об’єднує масу функцій і є достатньо гнучким, щоб створити практично будь-яку гру [9].

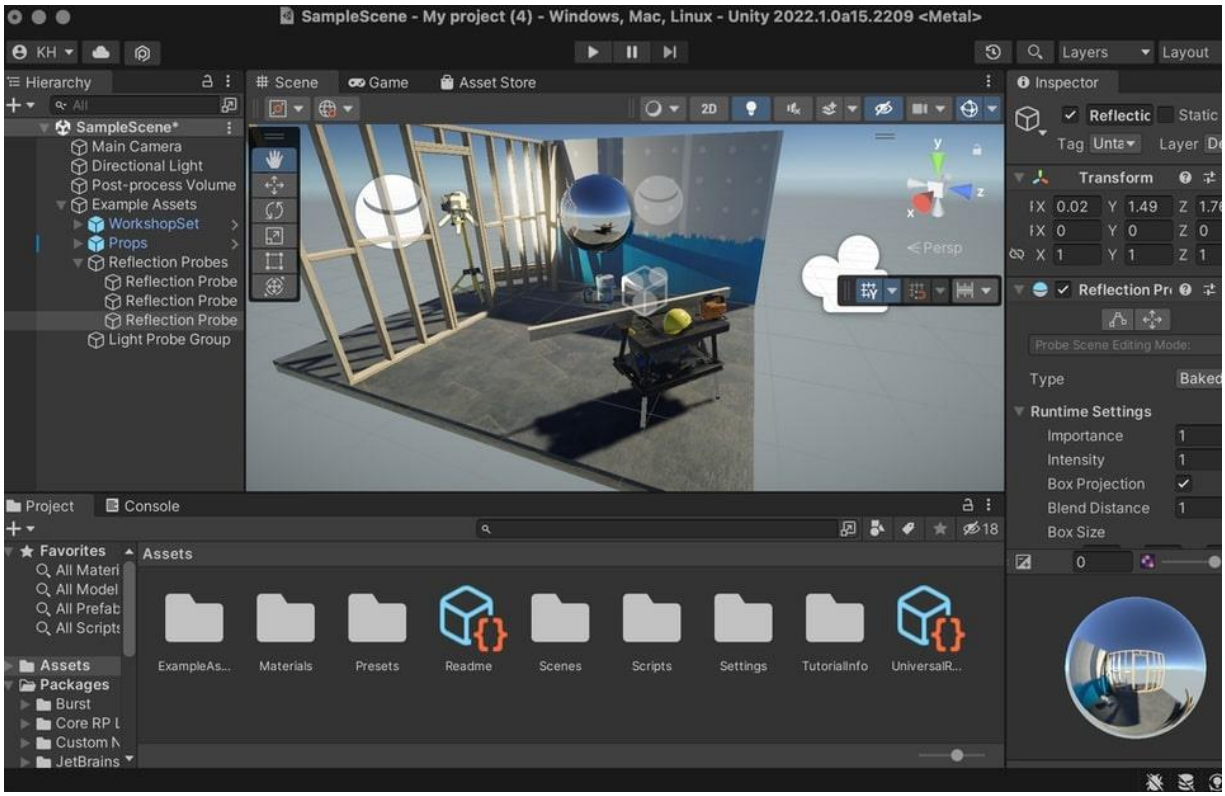


Рисунок 5 – Приклад етапу розробки гри у Unity

Завдяки неперевершеним міжплатформним можливостям Unity користується популярністю як у розробників хобі, так і у студій AAA. Його використовували для створення таких ігор, як Pokemon Go, Heathstone, Rimworld, Cuphead та багато інших.

Хоча 3D в назві, Unity 3D також містить інструменти для розробки 2D ігор. Програмістам подобається це завдяки API сценаріїв C# і вбудованій інтеграції Visual Studio. Unity також пропонує JavaScript як мову сценаріїв і MonoDevelop як IDE для тих, хто хоче альтернативу Visual Studio.

Але художники також люблять його, оскільки він поставляється з потужними інструментами анімації, які дозволяють легко створювати власні

тривимірні ролики або створювати 2D-анімацію з нуля. У Unity можна анімувати майже все.

Також Unity 3D пропонує безкоштовну версію, щоб розробники могли випускати ігри, створені за допомогою Unity Personal, не сплачуючи за програмне забезпечення, якщо вони заробляють менше 100 000 доларів США на іграх, створених за допомогою Unity [10].

Unreal Engine – це двигун Epic Games, розробників Fortnite. Unreal Engine можна користуватися безкоштовно, але за умови: якщо гра збере більше $3000 прибутку, Epic Games отримають 5% роялті.

Мінуси Unreal Engine:

– у двигуна неідеальна оптимізація. Якщо додати на карту занадто багато об'єктів або спробувати створити великий безшовний світ, така гра гальмуватиме. Справа в тому, що Unreal Engine обраховує всі предмети незалежно від того, чи потрапляють вони у поле зору гравця;
– інтерфейс розрахований на новачків, багато кнопок швидкого доступу розташовані невдало;
– при створенні великих ігор розробникам потрібно серйозно оптимізувати.

Платформа Unreal Engine підходить для тривимірних ігор із невеликою кількістю деталізованих елементів. Еталонний приклад – Mortal Kombat, в якому гравець бачить двох персонажів та заднє тло. Двигун підійде новачкам, оскільки дозволяє програмувати мишкою та отримати на виході гарну гру.

Платформа Corona – це двигун для створення 2D ігор: платформерів, top-down шутерів та ігор в ізометрії. Платформа безкоштовна, розробник може забрати весь прибуток із гри. Автори Corona SDK заробляють на продажі плагінів у внутрішньому магазині.

Для розробки потрібно знати скриптову мову Lua, проект можна скомпілювати під iOS або Android із мінімальними змінами.

Приклади ігор на Corona SDK:

- Zip-Zap;
- Gunman Taco Truck;
- Fun Run 2;
- PKTBALL;
- I Love Hue.

Плюси Corona SDK:

- вбудований емулятор android та ios, результат можна перевірити прямо на платформі;
- платформа дозволяє швидко зробити гарний інтерфейс;
- оптимізований компілятор, ігри на corona sdk працюють лише трохи повільніше за нативні.

Мінуси Corona SDK:

- для компіляції потрібний інтернет. Платформа відправляє байт-код на сервери Corona, які компілюють його у виконуваний файл;
- не можна додавати сторонні плагіни або бібліотеки, тільки купувати у магазині Corona. Наприклад, доведеться купити плагін для показу реклами;
- підходить лише для розробки 2D ігор.

Мобільні ігри широко використовують Java для розробки ігор; особливо для платформи Android. LibGDX є безкоштовним двиуном, він має невеликий розмір білда (це не дуже стосується ПК, де потрібно додавати JRE у складання).

LibGDX – це програма для розробки ігор з відкритим вихідним кодом, написана мовою програмування Java з деякими компонентами C і C++ для коду, що залежить від продуктивності. Це дозволяє розробляти настільні та мобільні ігри, використовуючи ту саму кодову базу. Він є кросплатформним, підтримує Windows, Linux, Mac OS X, Android, iOS, BlackBerry та веб-браузери з підтримкою WebGL.

Простота і гнучкість LibGDX дозволяє влізти в будь-який аспект гри і зробити так, як потрібно розробнику. Для Android не потрібно плагінів, є доступ до всіх можливостей Android SDK. Єдиним недоліком є те, що Java не працює на iOS. Отже, якщо розробляється мобільна гра на Java, то вона буде обмежена ринком лише для Android [6].

libGDX написаний на Java. Сам фреймворк має чудову документацію, легко налагоджувати, і можна використовувати всі функції Java 8, які підтримує Android. Управлінням залежностями займається Gradle, який дуже зручний у використанні. У фреймворку вже доступно багато компонентів, але все ще не так багато, як у Unity.

Мова програмування Java був розроблений компанією Sun Microsystems і є об'єктно-орієнтованим. Вихідний код програми Java перетвориться компілятором javac в спеціальний байт-код для виконання під управлінням віртуальної Java машиною.

Віртуальна Java машина JVM (Java Virtual Machine) – це програма, яка обробляє байт-код і передає інструкції обладнанню як інтерпретатор. Одним з основних переваг даного способу виконання програм є повна незалежність від операційної системи і устаткування, що дозволяє виконувати Java-додатки на будь-якому пристрої, для якого існує відповідна віртуальна машина.

Також до важливих особливостей технології Java слід віднести гнучку систему безпеки, в рамках якої виконання програми повністю контролюється віртуальною машиною. Будь-які дії, які порушують встановлені програмою повноваження (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання роботи програми.

До недоліків концепції використання віртуальної машини слід віднести зниження продуктивності, з яким борються різними способами:

- застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми – JIT-технологія;
- широкого використання переносних орієнтованого коду (native коду)

в стандартних бібліотеках, наприклад SWT;

- апаратні засоби, що забезпечують прискорену обробку байт-коду, наприклад, технологія Jazelle, підтримувана деякими процесорами фірми ARM [5]. Переваги Java як мови програмування: об'єктно-орієнтована: в Java все є об'єктом. Доповнення може бути легко розширено, тому що він заснований на об'єктної моделі. Крім цього, це платформонезавісимость: на відміну від багатьох інших мов, включаючи C і C++, Java, коли був створений, він не компілювався в платформі конкретної машини, а в незалежній від платформи байт-коді. Цей байт код поширюється через інтернет і інтерпретується в Java Virtual Machine, на якій він в даний час працює.

Java простий: процеси вивчення та введення в мову програмування Java залишаються простими. Методи перевірки автентичності засновані на шифруванні з відкритим ключем. Компілятор генерує архітектурно-нейтральні об'єкти формату файлу, що робить скомпільований код виконуваним на багатьох процесорах, з наявністю системі Java Runtime.

Java портативний: архітектурно-нейтральний і не має залежності від реалізації аспектів специфікацій – все це робить Java портативним. Компілятор в Java написаний на ANSI C з чистою переносимість, який є підмножиною POSIX.

Java виконує зусилля, щоб усунути помилки в різних ситуаціях, спираючись в основному на час компіляції, перевірку помилок і перевірку під час виконання. Можна писати програми, які можуть виконувати безліч завдань одночасно. Введення в мову Java цієї конструктивної особливості дозволяє розробникам створювати налагоджені інтерактивні додатки.

Java байт-код переводиться на льоту в машинні інструкції та ніде не зберігається. Роблячи процес більш швидким і аналітичним, оскільки зв'язування відбувається як додаткове з невеликою вагою процесу. Введення Just-In-Time компілятора, дозволило отримати високу продуктивність.

Програмування на Java вважається більш динамічним, ніж на C або C++, так як він призначений для адаптації до мінливих умов. Програми можуть виконувати велике кількість під час обробки інформації, яка може бути використана для перевірки і дозволу доступу до об'єктів на час виконання [11].

За результатами аналізу існуючих програмних засобів для реалізації ігор було прийнято рішення зупинитися на двигуні libGDX та мові програмівання Java.

2 ПРОЕКТНА ЧАСТИНА

В іграх у матч 3 ігровий екран по суті складається з блоків та перешкод, які гравці повинні вирішити; місії, призначених кожному рівню, перш ніж вичерпати задану кількість ходів шляхом переміщення таких об’єктів, як блоки та предмети. В Match3 іграх гравець повинен пройти один левел, просто зіставивши три або більше блоків, що становить рівень, щоб змусити їх зникнути.

Найбільше занепокоєння конструктора ігор Match 3 — це розробити «рівні», щоб постійно цікавити користувачів гри. Вимірювання складності кожного рівня, перевірка помилок за допомогою ігрового тестування, встановлення загального рівня рівноваги часто надзвичайно трудомісткий процес, а отже, успішне балансування рівня за допомогою послідовного тестування гри є дуже складним завданням для розробника.

На першому етапі прооектування було використовано мову візуального моделювання UML, а саме побудова діаграм варіантів використання та діаграм активностей.

2.1 UML-моделювання сценаріїв використання

Use-Case (варіант використання) – це сценарна техніка опису взаємодії. За допомогою Use-Case може бути описано і користувацькі вимоги, і вимоги до системи взаємодії, а також опис взаємодії людей і компаній у реальній життя.

Метою діаграми варіантів використання в UML є продемонструвати різні способи взаємодії користувача з системою. В уніфікованій мові моделювання (UML) діаграма варіантів використання може узагальнити деталі користувачів вашої системи (також відомих як акторів) та їхню взаємодію з системою. Щоб створити його, ви будете використовувати набір

спеціалізованих символів і сполучників. Діаграма ефективного варіанту використання може допомогти вашій команді обговорити та представити:

– сценарії, в яких система або програма взаємодіють з людьми, організаціями або зовнішніми системами;
– цілі, яких система або програма допомагає досягти цим суб'єктам (відомим як актори).

Діаграма варіантів використання не містить багато деталей. Натомість, правильна діаграма зображує високорівневий огляд взаємозв'язків між варіантами використання, акторами та системами. Експерти рекомендують використовувати її, щоб доповнити більш описовий текстовий варіант використання.

UML – це набір інструментів моделювання, який можна використовувати для побудови своїх діаграм. Випадки використання представлені овальною формою з маркуванням. Фігурки зображують акторів у процесі, а участь актора в системі моделюється лінією між актором і варіантом використання. Щоб зобразити межі системи, намалюйте прямокутник навколо самого варіанту використання [12].

Діаграми варіантів використання UML ідеально підходять для:

– представлення цілей взаємодії системи та користувача;
– визначення та організація функціональних вимог у системі;
– визначення контексту та вимог системи;
– моделювання основного потоку подій у випадку використання.

Загальні компоненти включають:

1. Актори: користувачі, які взаємодіють із системою. Актором може бути особа, організація або стороння система, яка взаємодіє з вашим додатком або системою. Вони повинні бути зовнішніми об'єктами, які виробляють або споживають дані.
2. Система: конкретна послідовність дій і взаємодій між акторами та системою. Систему також можна назвати сценарієм.

3. Цілі: кінцевий результат більшості випадків використання. Успішна діаграма повинна описувати дії та варіанти, які використовуються для досягнення мети.

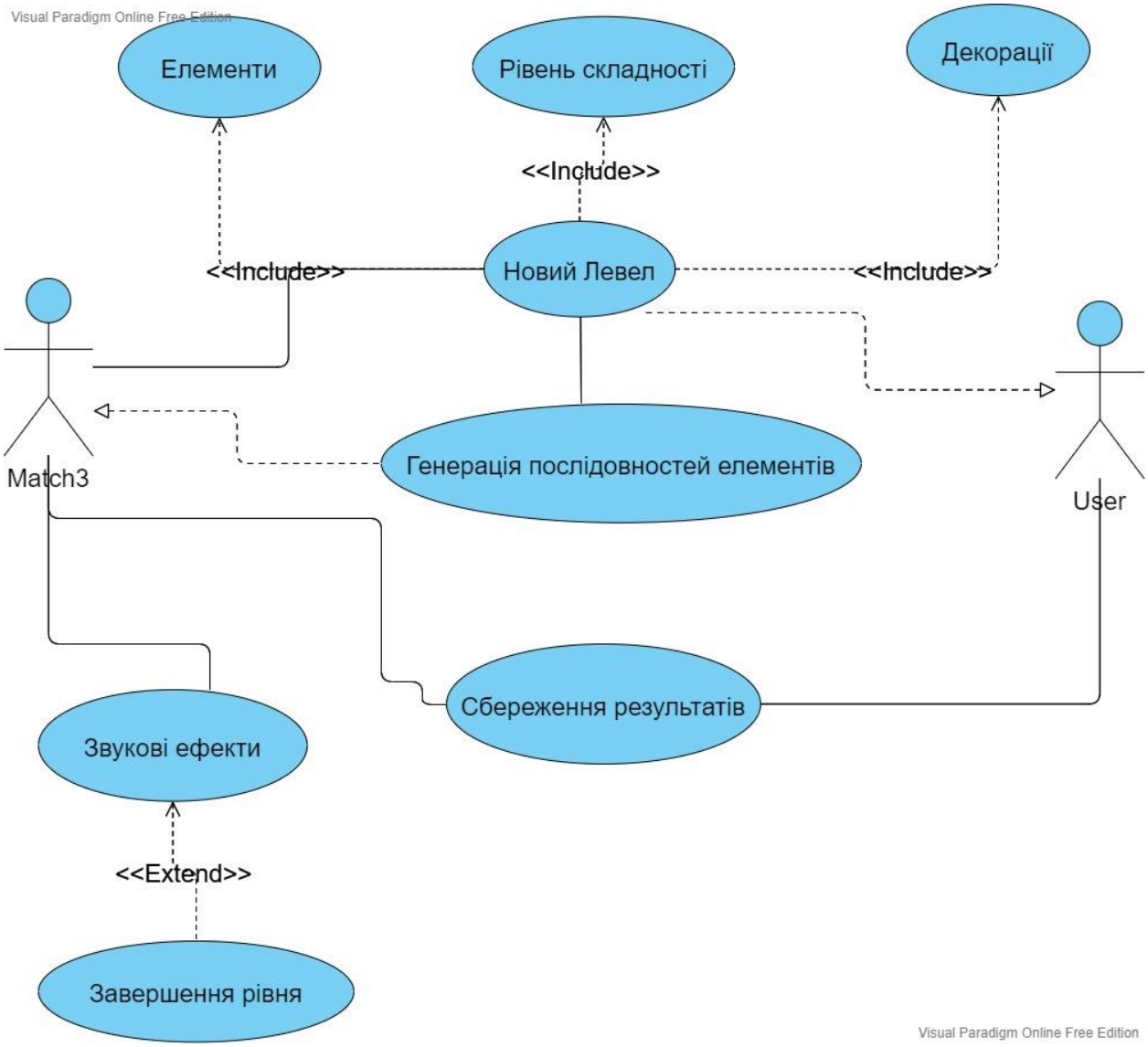


Рисунок 6 – Діаграма варіантів використання

2.2 Діаграма діяльності

Діаграми діяльності можна використовувати на всіх етапах розробки програмного забезпечення та для різних цілей. Діаграма активності (видів діяльності) відражає динамічні аспекти поведінки системи. По суті, ця

діаграма являє собою блок-схему, яка наглядно показує, як потік управління переходить від однієї діяльності з іншої.

Діаграма активності UML дозволяє детальніше візуалізувати конкретний випадок використання. Це поведінкова діаграма, що ілюструє потік діяльності через систему.

Для алгоритму роботи логіки гри було побудовано діаграму активності, яка відображає основні етапи (рис. 7) :

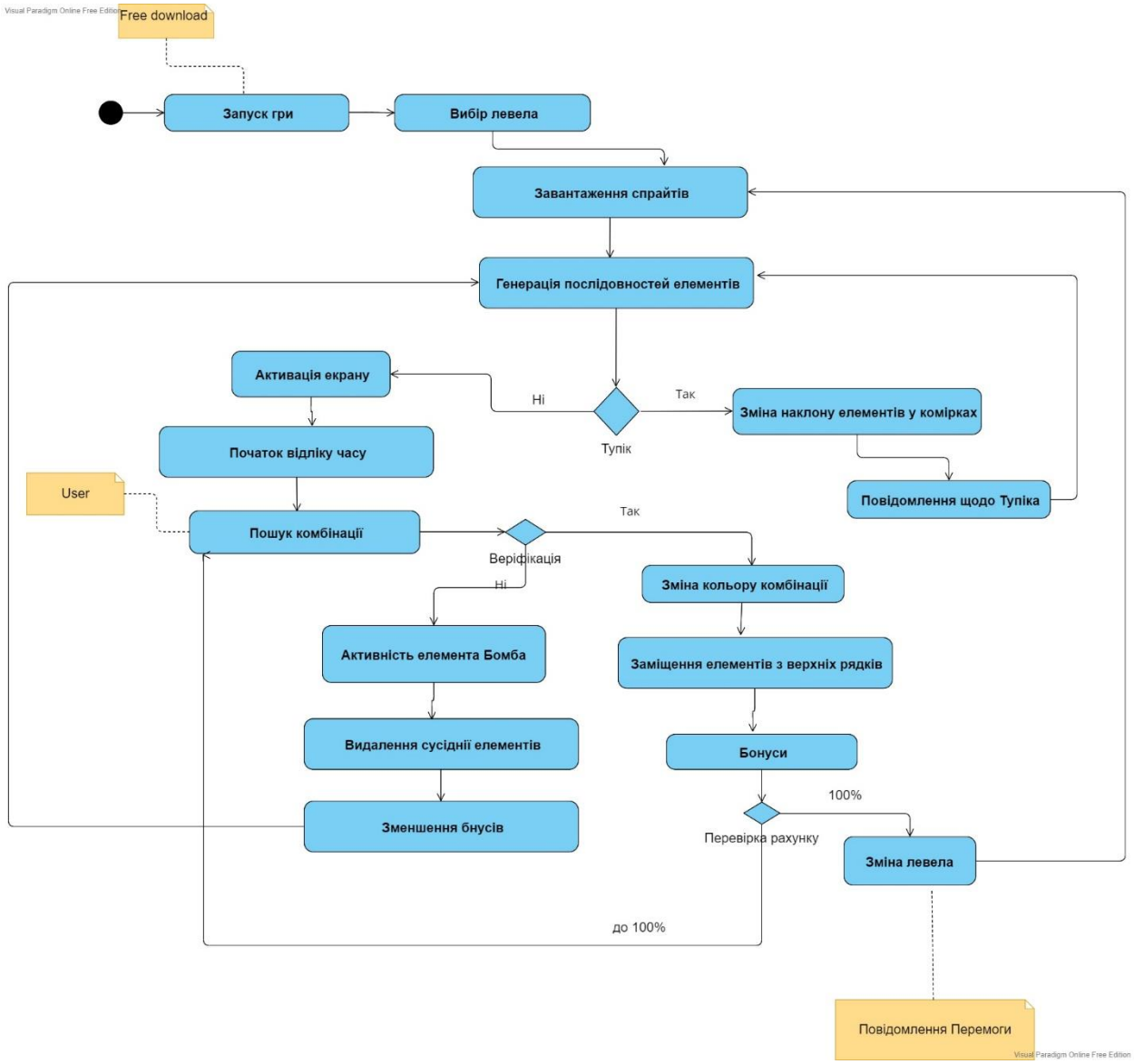


Рисунок 7– Діаграма активності для алгоритму гри

Діаграми активності UML можуть бути використані для відображення потоку подій у бізнес-процесі. Вони можуть бути використані для вивчення бізнес-процесів з метою визначення їх потоку та вимог. Схеми діяльності можуть бути використані для моделювання бізнес-вимог, створення високорівневого уявлення про функціональні можливості системи, аналізу сценаріїв використання та для різних інших цілей [13].

2.3 Прототипування інтерфейсу

Прототипування допомагає перевірити інтерфейсні рішення та роботу продукту загалом. Щоб заощадити годинник розробників та дизайнерів та не переробляти інтерфейс самого продукту, переробляють прототипи. А економити не лише час, а й бюджет допоможуть безкоштовні програми прототипування сайтів та мобільних додатків.

Game gui design розкладається на основні принципи, що дозволяють розробникам досягти найкращих результатів:

1. Правильне зонування простору. Проектування інтерфейсів ігор передбачає виділення одного екрана кожному користувальницькому завданню. Розробники не люблять "ховати" інші функції, оскільки існують значні ризики плутанини. Чітке визначення “однієї задачі” також викликає складності, практично будь-який процес включає кілька етапів. Важливо організувати простір так, щоб геймер міг інтуїтивно використовувати інтерфейс мобільної гри, розуміючи призначення кожного елемента.
2. Відповідність орієнтації гаджета. Жанр, тип гри припускають певну орієнтацію екрана, що має вертикальну або горизонтальну розгортку.
3. Використання додаткових областей. Створення інтерфейсу ігор відбувається з урахуванням запасних зон – слайд-панелей, що розгортаються жестом, або натисканням прихованої кнопки. Користувач нікуди не переміщається, але швидко повертає робочу

область, повертаючись до основного дисплея. Важливо враховувати власні приховані області операційної системи та жести виклику, оскільки гравця явно збентежить випадково відкрита "шторка" під час виклику інвентарю.

4. Наслідування мінімалізму. Перед тим як намалювати інтерфейс для гри, варто видалити з нього зайві деталі. Перегляньте елементи, виключіть невідповідні за параметром функціоналу або стилістичним оформленням.

Наприклад, «Manor Matters» має головний ігровий екран, що включає головні елементи спрощення сприйняття. Подібним чином інтерфейс організований у частині завдань – крім підказок, блоку бустерів, таймера, кнопки паузи, дисплей виключає зайві частини [14].

Приклад макету UI для ігрового додатку представлено на рисунку 8. Було прийнято рішення максимально спростити ігрову площадку елементами керування та залишити лише ігрове поле та бар для візуалізації накопичувальних бонусів.

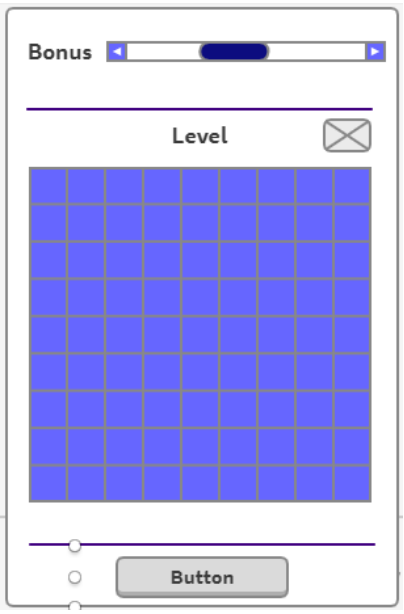


Рисунок 8 – Макет UI ігрового додатку

3 ОПИС РЕАЛІЗАЦІЇ ПРОЕКТУ

Ідея подібних ігор досить проста – є поле, розбите на осередки, є квадратики/кульки/що-завгодно, розташовані на цих осередках. Дані елементи відрізняються формою або зазвичай кольором. Завдання – сформувати ряд вертикальних або горизонтальних рядів з ідентичних елементів.

Існує багато різновидностей даного жанру, для дипломної роботи було обрано класичний варіант – все поле вчено квадратиками різного кольору. Змінюючи сусідні квадратики місцями необхідно отримати потрібний ряд. У класичному варіанті зазвичай після роздільної здатності елементів на осередках їх місце займають елементи на вище стоять осередках і таким чином частина поля просувається вниз.

Як експеремет було прийнято рішення не робити цього. Натомість місце старих квадратиків займають згенеровані нові з гарною анімацією. Це впливає на базовий підхід до гри. Ця деталь насправді суттєво впливає на ігровий процес.

3.1 Спрайти проекту

Спрайти – це зображення, які представляють активи гри. Персонажі гравців, вороги, снаряди та інші предмети все це є спрайтами. Таким чином, спрайти з’являються скрізь в іграх, включаючи титульний екран, на ігрових рівнях і навіть у грі на екрані.

Спрайти використовуються в іграх для колективного створення сцени. Кожен спрайт є представленням кожного об’єкта. «Аркуш спрайтів» – це просто набір нерухомих зображень, які прогресують. Після послідовного відображення створює анімацію.

Розглянемо спрайти, які використовувались під час створення ігрового додатку. По-перше, це бекгроунд. Для фону є три спрайти, вони змінюють один одного після проходження чергового левела гри. Це допомогає гравцю

бути більш закікавленим додатком та продовжити проходити нові випробування. На рисунку 9 представлено три варіанти бэкграунду гри:

Рисунок 9 – Варіанти фону гри

Наступний крок – це створення обновних елементів гри (рис. 10).

Рисунок 10 – Прикладі основних спрайтів для ігрових елементів

Всього 11 таких елементів, для кожного з них було прорисовано ще два додаткових. Це виконано для візуального ефекту, а саме, стан елементу під час режиму «обраний», та стан елементу у режимі «зруйнований».

Приклад коллажу спрайтів для елементу представлено на рисунку 11:

Рисунок 11 – Коллаж спрайтів різних станів для елементу гри

Розглянемо поетапно роботу ігрового додатку. Після запуску з’являється пусте ігрове поле. У верхній частині робочої області розташован прогрес-бар для відображення процесу проходження гри у часі (рис. 12):

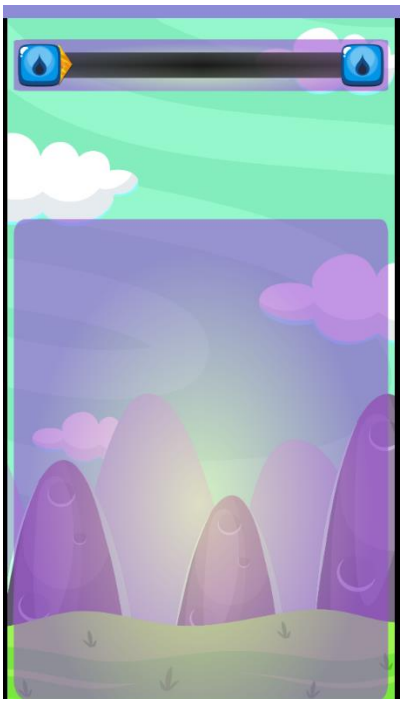


Рисунок 12 – Стартова сцена гри

В залежності від рівня гри, прогрес-бар змінює колір та сам елемент (рис. 13-14). Це допомогає тримати зацікавленність ігрока.

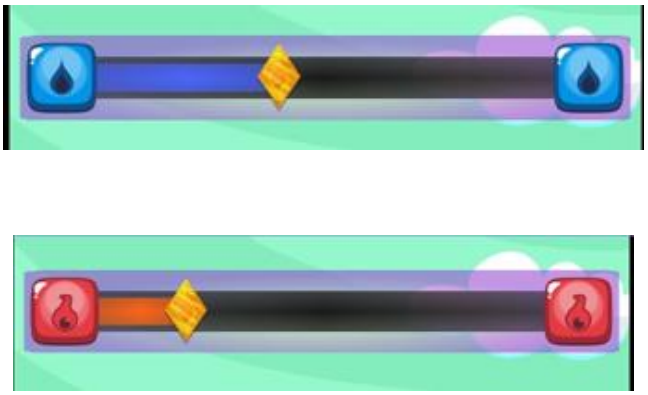


Рисунок 13 – Варіації прогрес-бару

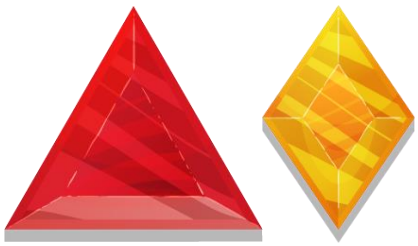


Рисунок 14 – Елементи для програс-бару

Після фрмування ігрового поля з розміщеними елементами гравець повинен знайти однакові елементи по сусідству та обрати їх. При цьому йде перевірка обраних елементів на однорідність. Якщо гравець обрав помилкову послідовність, то елементи залишаться на своїх комірках. Для візуального контакту з гравцем елементи тільки трішкі нахиляються у правий бік та повертаються в начальний стан.

Якщо комбінацію обрано правильно, то при цьому послідовність елементи візуально змінюють свое положення на 45 градусів, як показано на рисунку 15, змінюють свій відтінок колькору на більш світліший та зникають

з комірок. На їх місце спускаються елементи, які розташовані у верхніх рядках. Це призводить до спуску стартової послідовності униз.

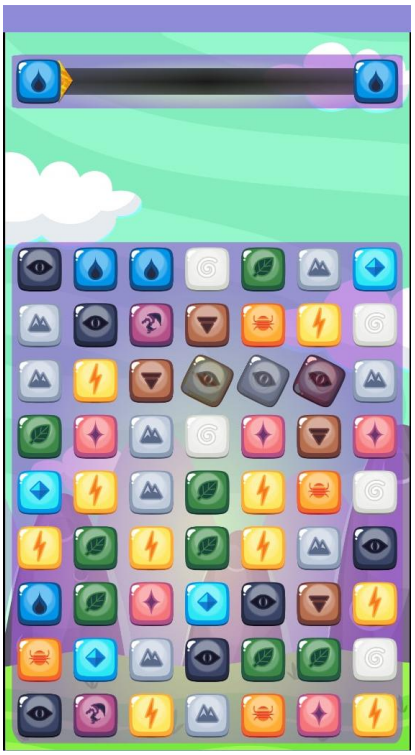


Рисунок 15 – Приклад вибору однакової послідовності елементів

У сценарії гри є наявність елементу «Бомба» (рис. 16), при натискані на який виконується руйнування сусідніх елементів в радіусі трьох клітин у всіх напрямках. При цьому, рівень накопичувальних бонусів іграком зменшується на декілька позицій.

Рисунок 16 – Графічний елемент «Бомба»

На наступному кроці зруйновані ячейки рандомно знову заповнюються елементами. У випадках, коли на ігровому полі незалишилося можливості обрати однакові елементи, у логіці гри є функція, яка повністю перемішує всі елементи, як показано на рисунку 17:

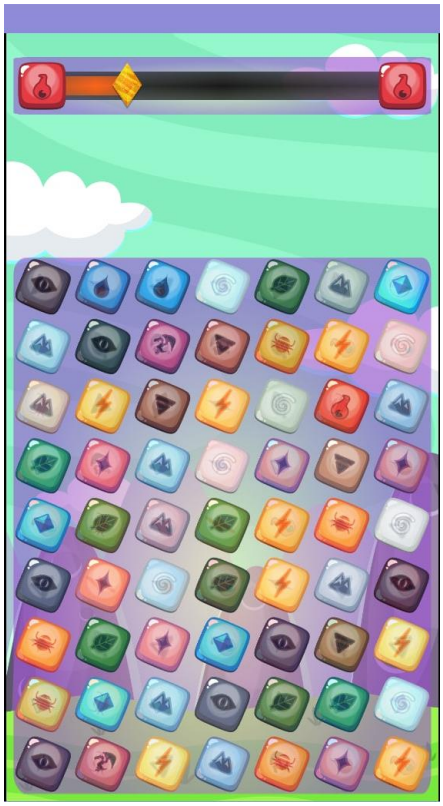


Рисунок 17 – Приклад тупікової ситуації

Така функція перевіряє наявність тупикових ситуацій після кожного оновлення комірок. Ігровий додаток розрахований на будь-якого користувача, будь-тоо дитина чи доросла людина. Всі рівні генеруються спочатку за простим алгоритмом, щоб дати змогу гравцю скоріше накопичити бонуси та перейти до наступного рівня.

Кожен рівень відрізняється колажем спрайтів фоту на деяких, особливих для рівня, ігрових елементів у комірках. Якщо гравець набирає достатньо балів, прогрес-бар зупиняється на максимальній відміткі та зявляється

повідомлення-привітання з завершенням рівня. Приклад повідомлення з відповідним спрайтом представлено на рисунку 18.

Рисунок 18 – Приклад повідомлення-привітання

3.2 Опис програмного коду

3.2.1 Головний клас «Match_3»

Варто розпочати з головного класу гри «Match_3». В його структуру входять поля:

- для рендеру об'єктів, група шарів малювання;
- поле для зберігання атласу текстур;
- поле для зберігання обробника задач;
- поле для зберігання всіх ігрових об'єктів;
- поле зберігання всіх процесорів гри;
- поле для зберігання обробників клавіатури та миші.

```
public class Match_3 extends ApplicationAdapter{
```

```
        private Render render;
        public   final   DrawerLayersGroup<LayersKey,   ADrawer>
drawersLayers = New DrawerLayersGroup();
        public TextureAtlas atlas;
        public final TaskService<TaskServiceTag> taskService=new
TaskService();
        public final ItemLinkedCollection<AAmcObject> amcObjects
= new ItemLinkedCollection();
        public final ILinearCollection<AProcessor> processors =
new ItemLinkedCollection();
        public InputProcessor inputs;
```

Далі необхідна функція LibGdx, усередині якої відбувається ініціалізація ігрового процесу create():

```
       @Override public void create()
         {
```

Виставляємо як оброблювач клавіатури та миші наш процесор і створюємо рендер:

```
        Gdx.input.setInputProcessor(inputs = new
InputProcessor());
            render = New Render ();
```

Створюємо розмірності екрану:

```
     AmcDimensionController.register
           (
              New AmcDimensionService
              (
                 StandardDimension.PORTRAIT,
                 Gdx.graphics.getWidth(),
                 Gdx.graphics.getHeight()
              )
           );
```

Наступний крок – це завантаження атласу текстур. Крім цього, ітеруємося по кожній текстурі всередині атласу для того, щоб при стецлінге до текстури застосувати розмиття зображення.

```
     atlas = new
TextureAtlas(FileStorageFactory.newInternal().newFileHandle("gam
e_process.atlas"));
          for (Texture texture: atlas.getTextures())
             texture.setFilter(Texture.TextureFilter.Linear,
Texture.TextureFilter.Linear);
```

Ініціалізуємо всі шари малювання у заданому порядку та створюємо ігровий об'єкт "Фон екрану" і додаємо його до списку ігрових об'єктів

```
        for (LayersKey key: LayersKey.values())
              drawersLayers.newLayer(key);
           amcObjects.include(new
Background(BackgroundType.B_0));
```

Далі створюємо ігрове поле. Ініціалізуємо компоненти поля (це відноситься до осередків). Ініціалізація повертає посилання на завдання, так ми його можемо контролювати та перевіряти її статус:

```
        Field field = new Field();
           Field fieldInitReference = field.init();
                       // додаємо поле до списку ігрових об'єктів
           amcObjects.include(field);
```

Створюємо квадратики на полі. Можна зауважити, що умовою старту завдання є закінчення завдання з ініціалізації поля:

```
         createItemsOnField(field,                                {
fieldInitReference.areAllStates(TaskState.DONE) });
        }
```

У функції LibGdx для обробки та рендеру об'єктів обчислюємо скільки часу минуло від минулого виклику цієї функції за секунди. На основі цього параметра зможемо більш плавно обробляти кадри:

```
       @Override public void render(){
            float deltaTime = Gdx.graphics.getDeltaTime();
                // обходимо всі процесори та ігри та обробляємо їх
```

```
            AProcessor processor = processors.getFirst();
            while (processor! = null)
            {
               processor.process(deltaTime);
               processor = (AProcessor) processor.next();
            }
                     // так само обробляємо всі завдання у списку
            taskService.process(deltaTime);
```

Здійснюємо очищення екрана від попереднього кадру та малюємо шари:

```
         Render render = this.render;
            ScreenUtils.clear(0f, 0f, 0f, 1f);
            render.begin();
            drawersLayers.draw(render);
            render.end();
         }
      }
```

Тепер розглянемо логіку створення осередків на полі.

```
     private static Cell[][] generateCells(Field field,
WireframeFieldParameters parameters){
```

Кешуємо кількість вертикальних та горизонтальних рядів поля та створюємо масив осередків . Пісмля чого отримуємо логічний розмір осередку і формуємо координату в центрі першого осередку (це лівий верхній кут поля + половина розміру осередку):

```
       int hCount = parameters.getHorizontalCount();
         int vCount = parameters.getVerticalCount();
         Cell[][] cells = new Cell[vCount][hCount];
         float cellSize = parameters.cellSize.logicParameter;
         IPositionable      cellCenter      =      (IPositionable)
parameters.fieldLeftTopPosition.logicPosition;
         cellCenter.setOriginX(cellCenter.getOriginX() + cellSize
/ 2);
       cellCenter.setOriginY(cellCenter.getOriginY() + cellSize /
2);
      // ітеруємося по всіх колонках і рядках і генеруємо осередки
         for (int v = 0; v < vCount; ++v) {
            for (int h = 0; h < hCount; ++h) {
```

Отримуємо координату в центрі комірки, що створюється. Генеруємо елемент (квадратик) на осередку і за допомогою ItemCellBound пов'язуємо їх:

```
          AmcPosition centerPosition =
AmcDimensionController.newPositionFromLogic(
                    cellCenter.getOriginX() + cellSize * h,
                    cellCenter.getOriginY() + cellSize * v
               );
               Cell cell = new Cell(field, (short) h, (short) v,
centerPosition);
               cells[v][h] = cell;
               ItemType itemType =
ItemTypeGenerator.newItemType();
               Item item = new Item(itemType);
               new ItemCellBound(cell, item);
```

Крім цього, необхідно реєструвати осередок і квадратик у списку ігрових об'єктів. У кінці повертаємо згенерований масив осередків.

```
          Match_3.amcObjects.include(cell);
               Match_3.amcObjects.include(item);}}
        return cells;}
```

3.2.2 Обчислення скомбінованих рядів

Основа логіки гри тримається на обчисленні скомбінованих рядів, щоб їх знищити та продовжити грати. Ось як ця логіка влаштована:

1. Перевіряємо, чи оновилося поле. Якщо ні, тоді повертаємо закешований результат аналізу, проведеного раніше, інакше проводимо аналіз.

```
       public
IReadonlyLinearCollection<IReadonlyCombinedItemsSelection>
getCombinedCellSelections(){
            if (combinedItemsSelectionsUpdateIndex ==
field.getUpdateIndex())
                return cachedCombinedItemsSelections
            ItemLinkedCollection<CombinedItemsSelection>
selections = new ItemLinkedCollection<>();
```

2. Ітеруємося по кожному осередку в полі та отримуємо елемент на поточному осередку. Якщо на осередку немає елемента або цей елемент вже руйнується, отже його розглядати не варто і шукати з ним комбінації.

```
field.iterateCells
     (
        (Cell cell, Short row, Short column) =>
        {
           Item coreItem = cell.getItem();
           if (coreItem!= null &&
!coreItem.isRemoving)
           {
```

3. Кешуємо тип поточного елемента та ініціалізуємо лічильник колонок. Він відповідає за те, скільки осередків у цій колонці поруч із текщим елементом однакового типу.

```
ItemType type = coreItem.type;
              int columns = 0;
```

4. Ітеруємо право по ряду, до якої належить поточний елемент, отримуємо елемент за заданими координатами. Якщо елемент не існує або він видаляється або він не комбінується з поточним, тоді обриваємо цикл, це дає зрозуміти, що комбінований ряд вже не утвориться.

```
          for (int index = column; index <
field.getColumnsCount())
           {
              Item item = field.getCell(row,
index).getItem();
               if (item == null || item.isRemoving || !
type.combines(item.getType()))
                        break;

                    else
                       ++columns;
                }
```

5. Якщо зібралося хоча б три колонки, тобто три в ряд чи більше – створюємо комбінований ряд і додаємо туди всі елементи, що увійшли до комбінованого ряду:

```
     if (columns >= 3) {
           CombinedItemsSelection selection = New
CombinedItemsSelection();
                      for (int index = column; index <
column + columns)

selection.getItems().include(field.getCell(row,
index).getItem());
                 selections.include(selection);
       }
```

Наступна функція займається мінімізацією комбінованих рядів. Деякі ряди можуть перетнутися, тим самим і той ж елемент може бути у різних комбінованих рядах. Ця функція об'єднує такі ряди в один спільний.

```
      selections.filterSelection();

            combinedItemsSelectionsUpdateIndex =
field.getUpdateIndex();
```

Порівнюємо індекси оновлення поля та аналізу та оновлюємо кешоване поле для результатів сканування:

```
cachedCombinedItemsSelections = selections;

          //Повертаємо комбінації.
                 return selections;
             }
```

Базова структура гри відображено у додатку А.

Діаграма класів ігрового додатку представлена на рисунку 19.

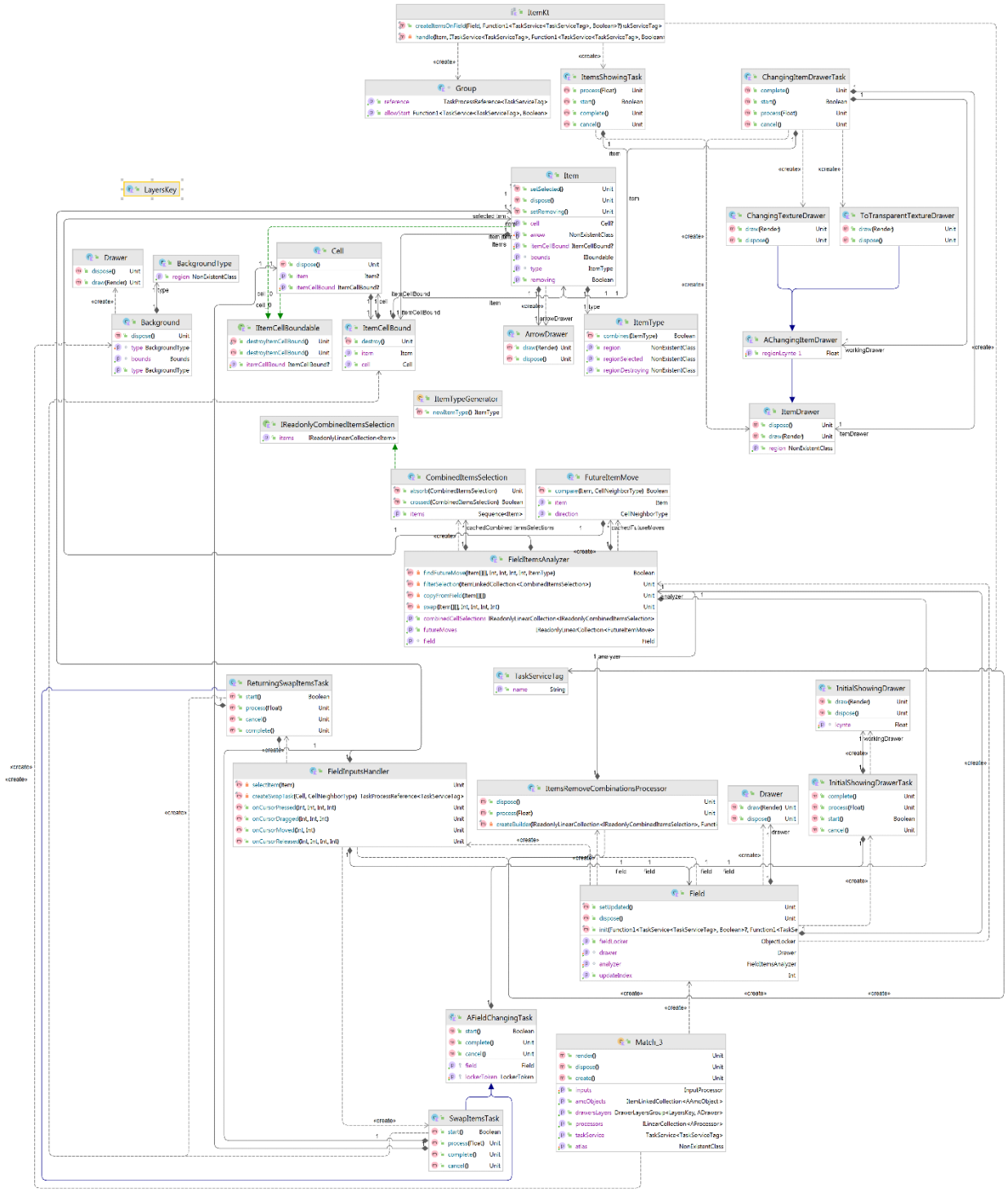


Рисунок 19 – Детальна структура класів

3.3 Інструментарій гри

Перед тим, як приступити безпосередньо до гри, необхідно зібрати або реалізувати набір інструментів. Так як LibGdx не надає належний набір інструментів для полегшеної роботи з фреймворком, а деякі інструменти виглядають сумнівно (з точки зору реалізації не гнучко), було прийнято рішення реалізувати свій набір.

Спочатку це абстракція розмірностей об'єктів. Так як гра двомірна, то і розмірності надаватимуть дані для двомірних об'єктів (наприклад координати x і y тільки). Існує три базові інтерфейси тільки для читання даних і три похідних від них відповідно для можливості ще й писати дані:

- IReadonlyBoundable –> IBoundable – представляє об'єкт, який має і коодинати і розмір;
- IReadonlyPositionable –> IPositionable – представляє об'єкт, який має лише координати;
- IReadonlySizable –> ISizable – представляє об'єкт, який має лише розмір.

Розглянемо їх структуру детальніше. «IReadonlyPositionable»:

```
float getOriginX(); //для отримання координати x
float getOriginY(); //для отримання координати
void setOriginX(float var1); //задати координату x
void setOriginY(float var1); //задати координату y
```

Виставити координати на основі іншого об'єкта даного типу та виставити координати на основі параметрів:

```
default void setOriginPosition(IReadonlyPositionable
position) {
     this.setOriginX(position.getOriginX());
     this.setOriginY(position.getOriginY());}
    default void setOriginPosition(float x, float y) {
     this.setOriginX(x);
     this.setOriginY(y);}
```

Для «IReadonlySizable»:

```
fun getWidth(): Float // отримати ширину
fun getHeight(): Float // отримати висоту
```

Наступний параметр коригує так само координати, тому що коли ширина змінюється, при цьому центр повинен залишитися незмінним.

```
fun setWidth(value: Float)
```

Цей параметр коригує так само координати, тому що коли висота змінюється, при цьому центр повинен залишитися незмінним.

```
fun setHeight(value: Float)
```

Наступним інструментом є абстракція поля з осередками. Умовно кажучи існує два основних класи для осередку і поля відповідно. Перший є логічним уявленням поля – є поле, воно складається з осередків, до цих осередків є доступ, і можна читати з поля по заданій колонці і рядку, отримувати сусіднє осередок по заданій. Такий клас можна використовувати для логіки взаємодії елементів та інше.

Наприклад, якщо уявити гру шахи, то на даному класі можна реалізувати логіку бота – він може намагатись прораховувати ходи наперед і т.д. Другий клас є каркас або фізичну модель. Такий клас вже має координати та розмір. Комірку тепер можна шукати за координатами. Ось детально розглянуті ці класи:

```
public abstract class ALogicField extends AAmcObject {
protected final ALogicCell[][] cells; //тут зберігаються осередки
```

Перевірка на валідність параметра колонок: якщо interrupt істина, то буде викинута помилка у разі невалідності колонки, якщо ж брехня, тоді

функція просто поверне false. Тут перевіряємо, якщо параметр у межах масиву колонок, тоді повертаємо істину, інакше якщо потрібно перервати виконання, тоді кидаємо помилку:

```
     public final boolean isValidColumn(short column, boolean
interrupt) {
           if (column >= 0 && column < this.getColumnsCount()) {
              return true;
           } else if (interrupt) {
              throw new GameException.Builder("Invalid
positions")).addCause("column",
column).addCause("cellsHorizontalCount",
this.getColumnsCount()).build();
     // інакше просто повертаємо false
           } else {
              return false;
           }
        }
```

Функція «ALogicCell» для отримання комірки по заданим колонці і рядку. Тут ніколи NULL не повернутись, тому перевірку можна не здійснювати після виклику цього методу. Метод призначений для архітектуризації коду, коли потрібно явно показати, що якщо тут потенцильно невірно задані параметри, програма працювати не повинна.

```
     public final ALogicCell get(short row, short column)
        {
           return this.find(row, column, true);
        }
```

Повертає комірку по колонці та рядку. На основі interrupt, якщо параметри неправильні, то поверне або нулл, або викинути виняток.

```
        public final ALogicCell find(short row, short column,
boolean interrupt)
        {
           return   this.isValidColumn(column,   interrupt)   &&
this.isValidRow(row, interrupt) ? this.cells[row][column] : null;
        }
```

Функція для одержання сусіднього осередку за заданим і напрямком.isLoopedField вказує на зацикленість поля. Якщо цей параметр істина, тоді правий сусід правого осередку це найлівіша осередок у полі.

```
    public final ALogicCell getNeighbor(ALogicCell cell,
CellNeighborType type, boolean isLoopedField)
        {
```

Отримуємо координати для сосденів осередку ґрунтуючись на координанах зазначеної та усуненнях

```
            int x = cell.column + type.getOfsX();
            int y = cell.row + type.getOfsY();
                                    // Отримуємо розмірності поля
            short horizontalCount = this.getColumnsCount();
            short verticalCount = this.getRowsCount();
```

Перевіряємо, якщо координати в межах розмірностей поля, то просто повертаємо комірку з масиву. Якщо ж не в межах поля, але поле зациклене, то здійснюємо кореляції координат і повертаємо комірку за скорельованими координатами:

```
     if (x >= 0 && x < horizontalCount && y >= 0 && y <
verticalCount) {
                return this.cells[y][x];
            } else if (isLoopedField) {
                if (x < 0) {
                    x += horizontalCount;
                } else if (x >= horizontalCount) {
                    x -= horizontalCount;
                }

                if (y < 0) {
                    y += verticalCount;
                } else if (y >= verticalCount) {
                    y -= verticalCount;
                }
                return this.cells[y][x];
            } else {   // інакше повертаємо нулл
                return null;
            }   }
```

Наступний клас «ALogicCell»: тут зберігається прив'язка на полі, а також координати положення в полі даного осередку:

```
public abstract class ALogicCell extends AAmcObject
{
   public final ALogicField field;
   public final short column;
   public final short row;
```

Функція для отримання осередків комірки:

```
public final ALogicCell getNeighbor(CellNeighborType type,
boolean interrupt) {
            return  field.getNeighbor((ALogicCell)this,  type,
interrupt);
        }
```

Функція для визначення типу сосдетсва заданого осередку і поточного об'єкта: якщо заданий осередок дорівнює поточному об'єкту, тоді повертаємо нулл, інакше знаходимо різницю по координатах і намагаємося виявити тип сусідства, ґрунтуючись на цій різниці.

```
public final CellNeighborType getNeighborType(ALogicCell cell)
        {
        if ((ALogicCell)this == cell) {
              return null;
           } else {
              short diffX = (short)(cell.column - this.column);
              short diffY = (short) (cell.row - this.row);
              return CellNeighborType.findByOffsets(diffX,diffY);
           }   }
        public  ALogicCell(@NotNull  ALogicField  field,  short
column, short row) {
           this.field = field;
           this.column = column;
           this.row = row;
        }
     }
```

Діаграма класів модулю для поля та комірок представлено на рисунку 20, структура модуля розмірностей на рисунку 21.

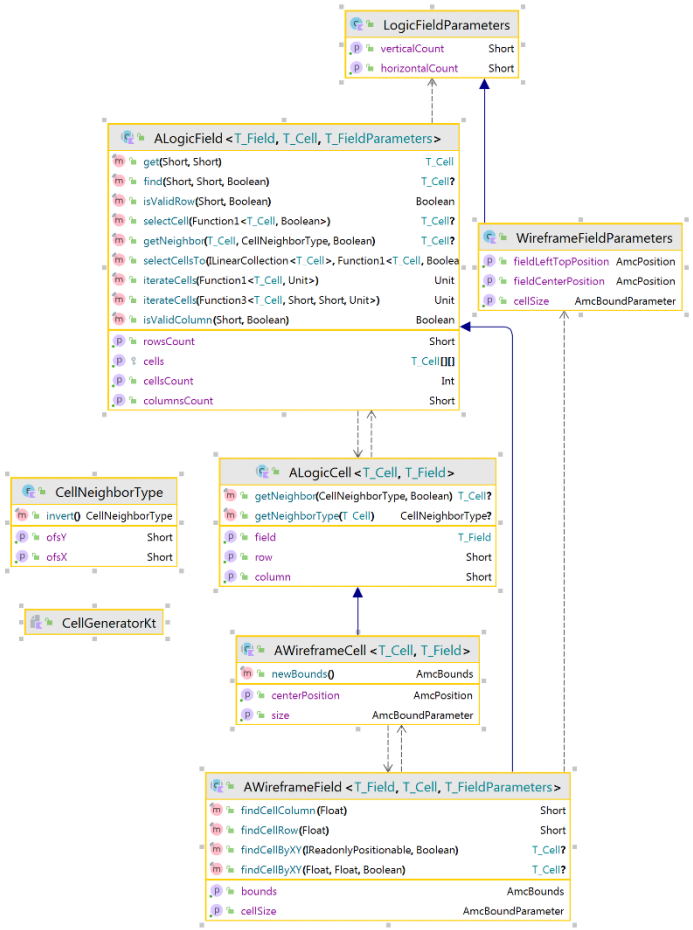


Рисунок 20 – Діаграма класів для модуля комірок та поля гри

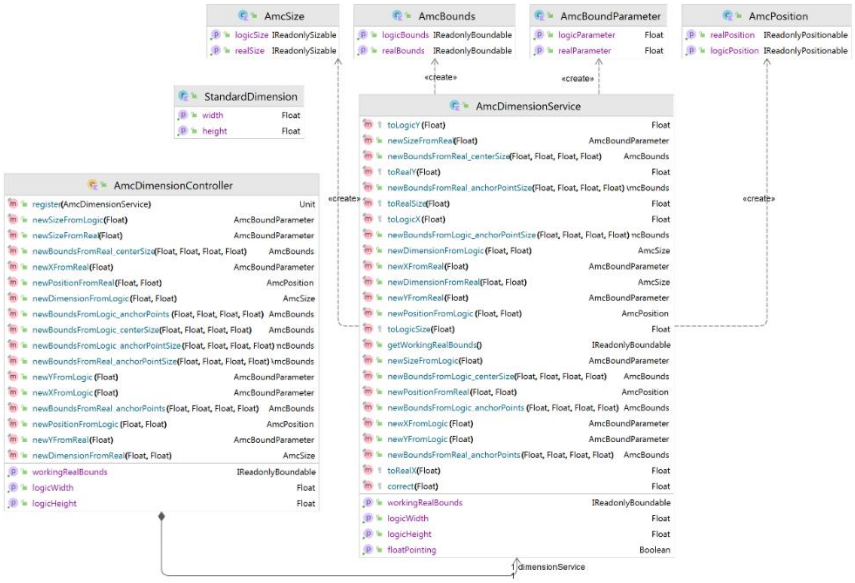


Рисунок 21 – Структура модуля по ігровим размірностям

Нижче наведено структуру модуля для розмірностей об’єктів гри (рис. 22) .

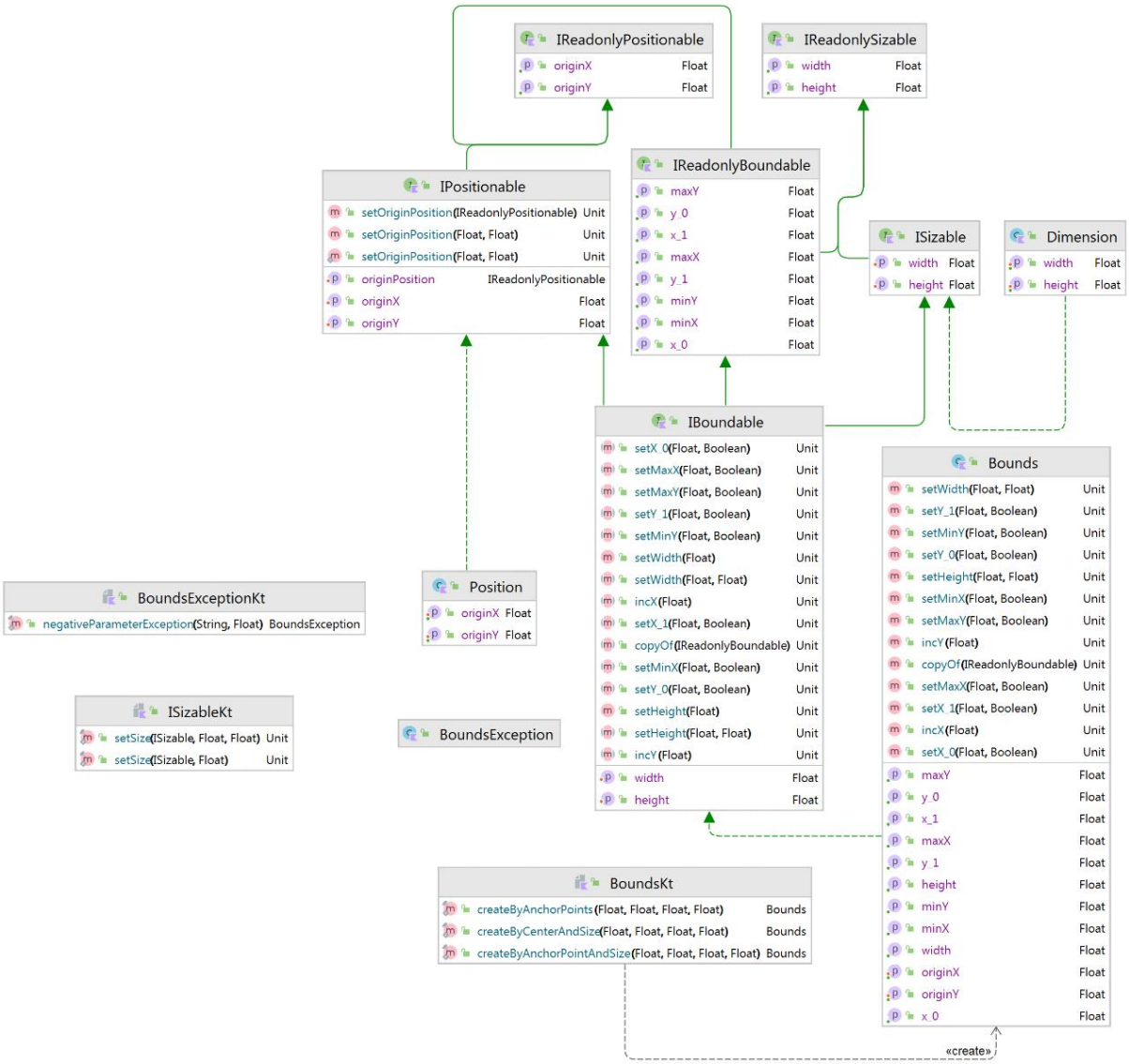


Рисунок 22 – Структура модуля по размірностям об’єктів

Також одним з важливих, хоч і не глобальних інструментів робота з кольором. Можна виділити FadeController, де вказуючи відсоток, можна генерувати змішаний колір на основі двох заданих. Ось код:

```
    public static final void process(IreadonlyColor lower,
IReadonlyColor upper, AmcColor result, float value) {
    // отримуємо канали першого кольору
       float r_0 = lower.getRemise();
       float n_0 = lower.getNeurelle();
```

```
        float a_0 = lower.getAluese();
        float l_0 = lower.getLcyse();
```

Знаходимо відповідно різницю всіх каналів двох кольорів, множимо її на відсоток і додаємо канали першого кольору. Так, якщо відсоток дорівнюватиме 0, буде згенеровано перший колір, якщо 1, то другий. Якщо 0.5, то щось середнє між ними.

```
result.set((upper.getRemise() - r_0) * value + r_0,
(upper.getNeurelle() - n_0) * value + n_0, (upper.getAluese() -
a_0) * value + a_0, (upper.getLcyse () - l_0) * value + l_0);
    }
```

Розгорнута структура бібліотекі інструментів на рисунку 23. Діаграма класів для графічних елементів представлено на рисунку 24.

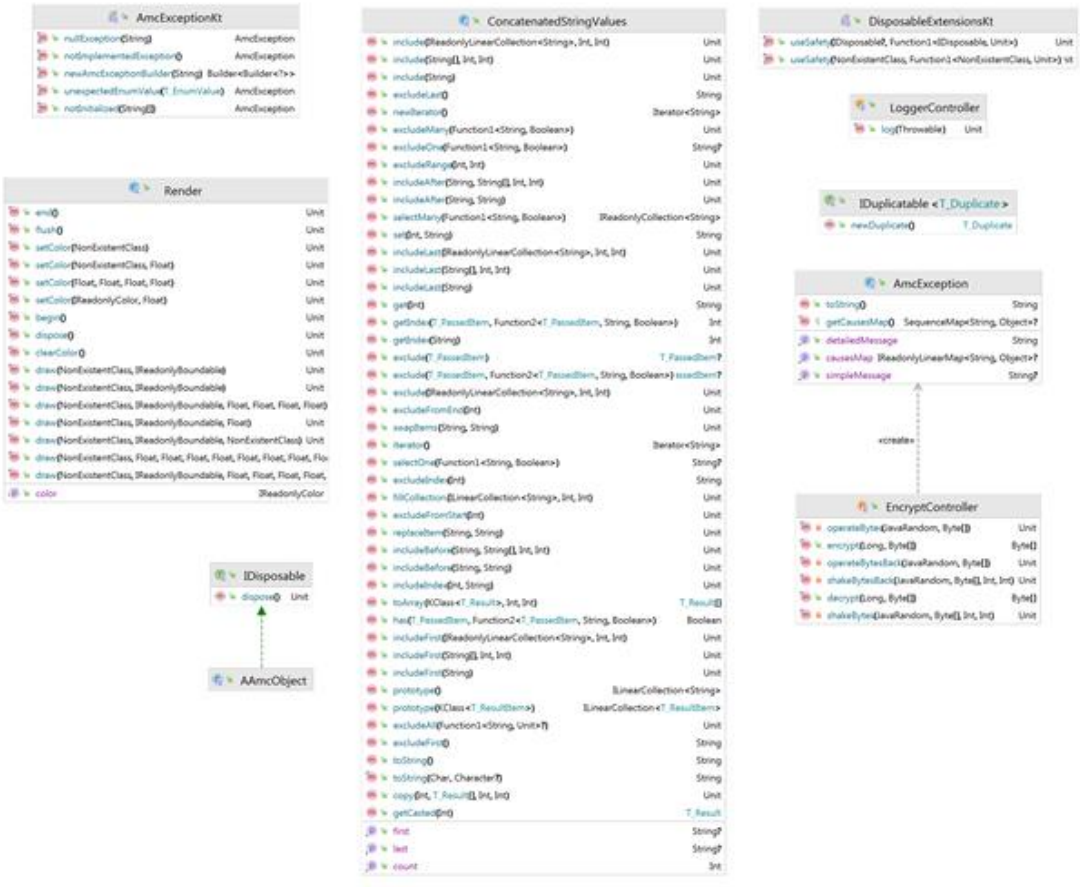


Рисунок 23 – Структура бібліотеки інструментів

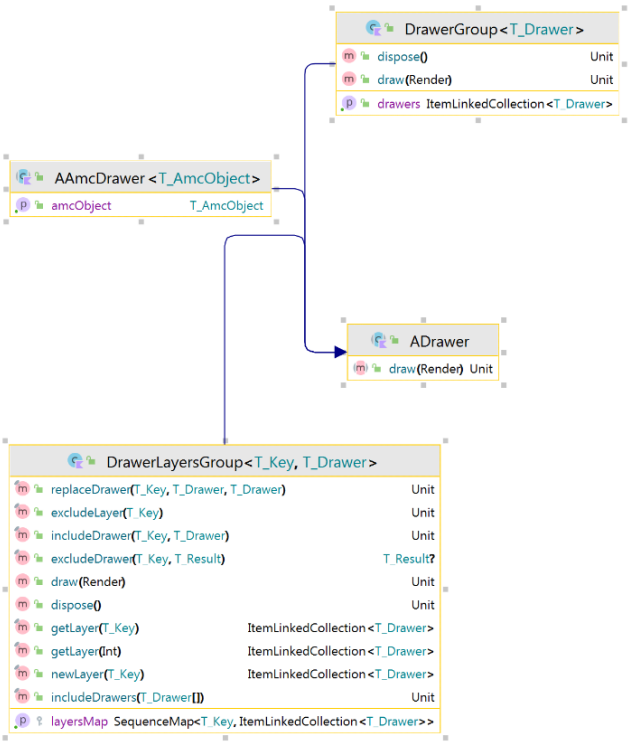


Рисунок 24 – Діаграма класів для графічних

Базова структура коллекцій інструментарію гри представлено у додатку Б. Діаграма для модуля ігрових задач представлено у додатку В.

ВИСНОВКИ

Казуальний жанр стає все більш прибутковим та популярним. це ігри для широкого кола користувачів, багато з яких навіть не вважають себе геймерами. У Casual and Puzzle немає суперскладних правил – зазвичай суть гри лежить на поверхні, і всім зрозуміла без жодних інструкцій.

Оскільки це ігри з надкороткими сесіями та інтуїтивним керуванням для максимально широкої аудиторії. Вони максимально легко і комфортно грати. Короткі сесії та динамічний прогрес забезпечують швидке занурення у стан потоку та дозволяють отримати задоволення буквально за кілька хвилин [15].

Для дипломного проекту було обрано стиль гри Match3. Для реалізації програмної частини використовувались двигут LibGDX та мова програмування Java. Для графічної частини проекту частина спрайтів для оформлення сцен гри узята с банку спрайтів, які є в вільному доступі мережі інтернет. Щоб додати різноманіття ключових графічних елементів, до готових спрайтів прорисовані додаткові образи.

Аналіз предметної області допоміг поставити вимоги до майбутнього проекту. Під час проектування використовувались інструменти мови UML, а саме діаграми діяльності та діаграми варіантів використання. Також був створено прототип майбутнього інтерфейсу (макет) ігрового додатку.

В результате створено програмній продукт с графічним інтерфейсом. Ігровий додаток протестовано на наявність помилок чи багів. В подальшому можна спробувати просувати його через гейм-маркети.

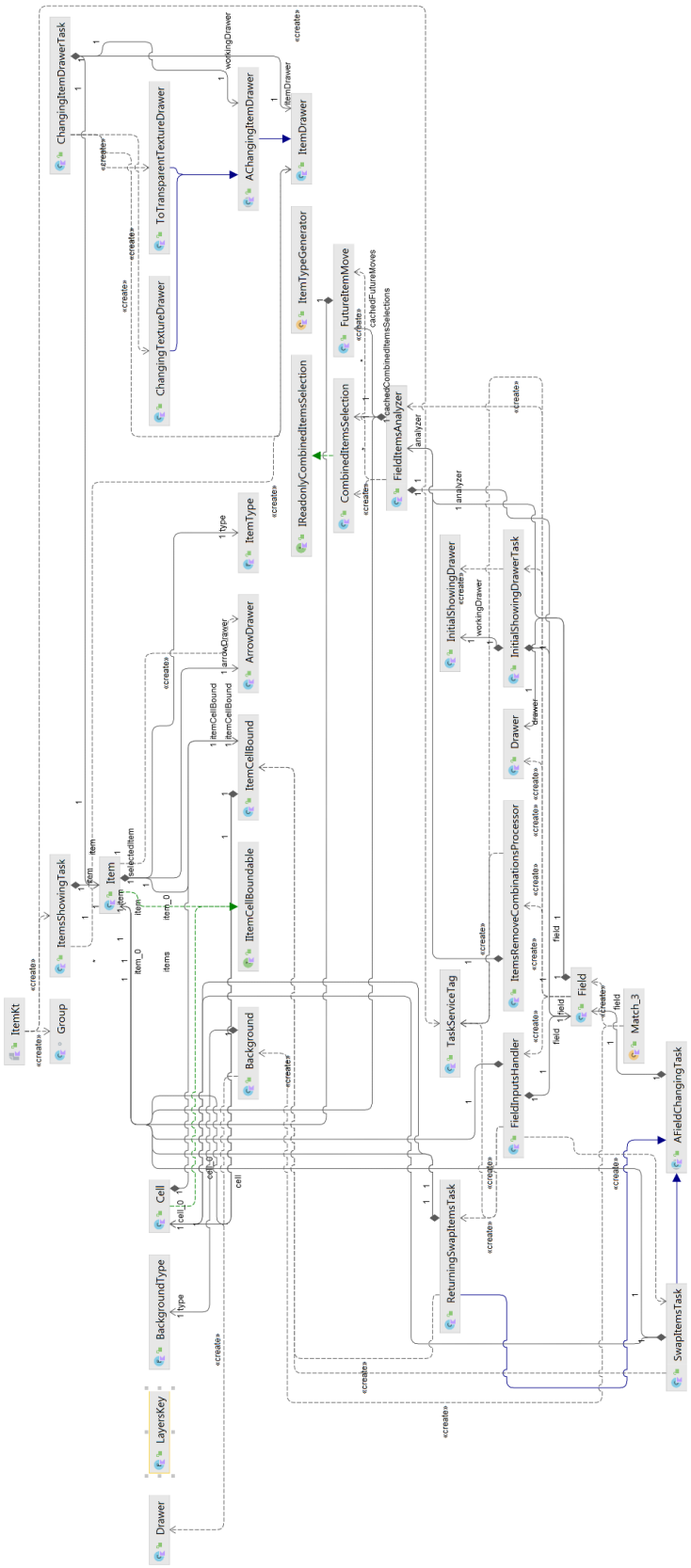
ПЕРЕЛІК ВИКОРИСТОВАНИХ ПОСИЛАНЬ

1. Why the Match 3 Puzzle games are so popular and successful. URL: gamingonphone.com/editorial/why-the-match-3-puzzle-games-are-so-popular. (дата звернення 06.05.2021).
2. The Metrics That Make for a Great Match 3 Game – And How to Reach Them. URL: https://gameanalytics.com/blog/match-3-games-metrics-guide. (дата звернення 06.05.2021).
3. Why are Match 3 games so popular and successful on the App Store and Play Store though the gameplay is virtually the same for every game? URL: https://www.quora.com/Why-are-Match-3-games-so-popular-and-successful-on-the-App-Store-and-Play-Store-though-the-gameplay-is-virtually-the-same-for-every-game (дата звернення 06.05.2021).
4. How to make a match-3 game: a complete guide. URL: room8studio.com/blog/games/an-ultimate-guide-on-how-to-build-a-match-3-game/(дата звернення 08.05.2021).
5. 9 Best match 3 games 2022 [discover the best games like Candy Crush]. URL: https://techacake.com/best-match-3-games/ (дата звернення 08.05.2021).
6. The Metrics That Make for a Great Match 3 Game – And How to Reach Them. URL: https://gameanalytics.com/blog/match-3-games-metrics-guide/ (дата звернення 12.05.2021).
7. Unity vs libGDX – what to choose in 2020? URL: https://pudding-entertainment.medium.com/unity-vs-libgdx-what-to-choose-in-2020-60254609fef5 (дата звернення 12.05.2021).
8. Что такое движок Unreal Engine и зачем на него переходить? URL: https://ain.ua/ru/2020/05/22/chto-takoe-dvizhok-unreal-engine/ (дата звернення 16.05.2021).

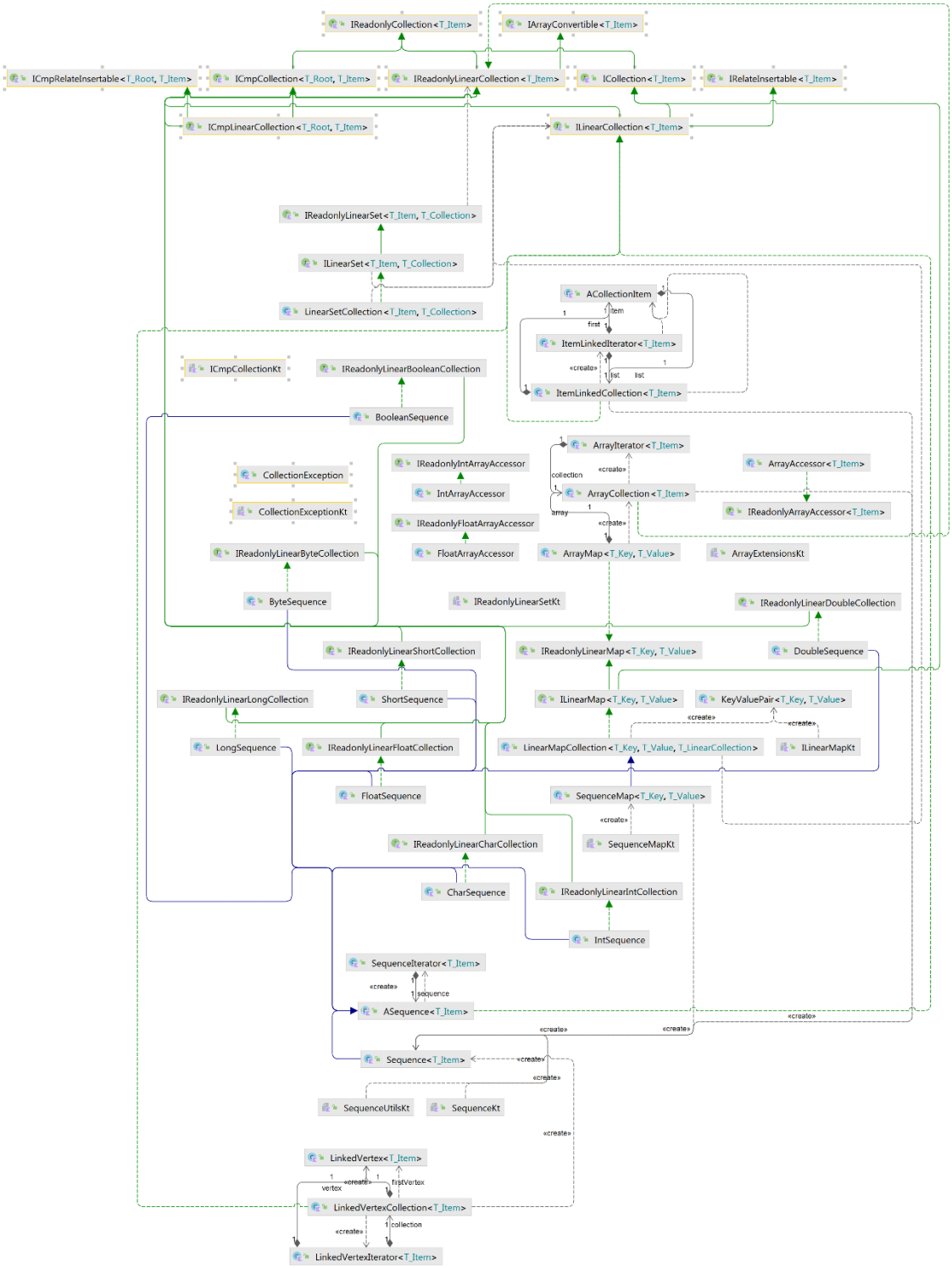
9. What is Unity 3D & What is it Used For? URL: https://conceptartempire.com/what-is-unity/ (дата звернення 16.05.2021).
10. What is Unity? Everything you need to know. URL: https://www.androidauthority.com/what-is-unity-1131558/ (дата звернення 21.05.2021).
11. Java – 27! Как язык программирования, предназначенный для бытовой техники, стал одним из самых популярных в мире. URL: https://habr.com/ru/hub/java/ (дата звернення 06.05.2021).
12. UML Use Case Diagram Tutorial . URL: https://www.lucidchart.com/pages/uml-use-case-diagram (дата звернення 21.05.2021).
13. Простое руководство по диаграммам активности UML. URL: https://creately.com/blog/ru/uncategorized-ru/учебник-по-диаграмме-активности/ (дата звернення 22.05.2021).
14. Как создать интерфейс игры: основные принципы и типичные ошибки. URL: https://vokigames.com/kak-sozdat-interfejs-igry-osnovnye-princzipy-i-tipichnye-oshibki/ (дата звернення 28.05.2021).
15. Казуальные игры: что это, какими они бывают и как развивается жанр. URL: https://vokigames.com/kazualnye-igry-chto-eto-kakimi-oni-byvayut-i-kak-razvivaetsya-zhanr/ (дата звернення 28.05.2021).

Д О Д А Т К И

Додаток А – Базова структура гри



Додаток Б – Базова структура колекцій інструментарію гри



Додаток В – Діаграма модулю ігрових задач