МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІГРОВОГО ЗАСТОСУНКУ ЗАСОБАМИ PYGAME ТА POSTGRESQL»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
К.О. Кабанов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, к.ф.-м.н. Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.

(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кабанову Костянтину Олексійовичу

(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка ігрового застосунку засобами Pygame та Postgresql

керівник роботи Кривохата Анастасія Григорівна, к.ф.-м.н.

(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023

3. Вихідні дані до роботи 1. Постановка задачі.

2. Перелік питань до розробки.

3. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд проблеми.

2. Розробка ігрового застосунку.

3. Результат.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Гребенюк С.М., завідувач кафедри фундаментальної та прикладної математики	08.02.2023	05.04.2023
2	Гребенюк С.М., завідувач кафедри фундаментальної та прикладної математики	06.04.2023	26.04.2023
3	Гребенюк С.М., завідувач кафедри фундаментальної та прикладної математики	27.04.2023	24.05.2023

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	13.02.2023	
3.	Обробка методичних та теоретичних	15.03.2023	
	джерел.		
4.	Розробка першого розділу.	05.04.2023	
5.	Розробка другого розділу.	26.04.2023	
6.	Розробка другого розділу.	24.05.2023	
7.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	22.06.2023	

Студент _____ К.О. Кабанов
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка ігрового застосунку засобами Pygame та Postgresql»: 54 с., 8 рис., 6 джерел, 6 додатків.

БАЗА ДАНИХ, ВІДЕОІГРА, ГЕЙМДИЗАЙН, ГРА, ГРАФІКА, ІГРОВИЙ ЗАСТОСУНОК, ІНТЕРАКТИВНІСТЬ, ПРОГРАМУВАННЯ, PYGAME, PYTHON, POSTGRESQL, SQL.

Об’єкт дослідження – процедура розробки ігрового застосунку засобами Pygame та PostgreSQL.

Мета роботи: розробка ігрового застосунку «Alien Invasion» з інтерактивними графічними ефектами та базою даних для збереження рекордів гравців.

Метод дослідження – аналітичний, практичний, порівняльний.

Гра є об’єктом, який ніколи не вичерпується, завдяки своїй нескінечній творчій потужності. В сучасному світі ігрова розробка може набувати все більшого розвитку завдяки застосуванню усієї глибини і різноманітності можливостей, які надають pygame та PostgreSQL.

Для розширення розуміння ігрової розробки, у процесі роботи над проєктом досліджуються різноманітні аспекти, включаючи графіку, фізику, анімацію та взаємодію з базою даних. У розробці ігор вивчаються різні елементи гри та їх взаємодія.

Під час розробки ігрового застосунку засобами pygame та PostgreSQL досліджуються можливості створення геймплею, впровадження системи керування гравцем, обробка різних подій та збереження даних гри в базі даних. Знання технічних можливостей цих інструментів дозволяє створювати складні та захоплюючі ігри, прискорюючи процес розробки та розв’язання різноманітних завдань.

SUMMARY

Bachelor’s Qualifying Paper «Development of a Game Application using Pygame and Postgresql Tools»: 54 pages, 8 figures, 6 references, 6 supplements.

DATABASE, GAME, GAME APPLICATION, GAME DESIGN, GRAPHICS, INTERACTIVITY, PROGRAMMING, PYGAME, PYTHON, SQL, VIDEO GAME.

The object of research is the development of a game application.

The aim of the work is to investigate all aspects of game application development.

The research method is analytical, practical, and comparative.

Developing a game application using Pygame and PostgreSQL can be exciting and meaningful. A game is an object that is never exhausted, thanks to its infinite creative potential. In today's world, game development is gaining increasing importance as we delve into the depth and variety of possibilities offered by pygame and PostgreSQL.

To expand our understanding of game development, we explore various aspects, including graphics, physics, animation, and interaction with a database. In game development, we study different game elements and their interactions.

During the development of a game application using pygame and PostgreSQL, we explore the possibilities of creating gameplay, implementing player control systems, handling various events, and storing game data in a database. Knowledge of the technical capabilities of these tools allows us to create complex and captivating games, speeding up the development process and solving various tasks.

ЗМІСТ

ВСТУП

Розробка ігор з використанням pygame та PostgreSQL надає розробникам можливість створювати високоякісні ігри зі зручним управлінням та збереженням даних. Цей процес включає аналіз вимог, проєктування, реалізацію та тестування застосунку. Використання цих технологій дозволяє створювати ігри з багатошаровою графікою та різноманітними рівнями. База даних PostgreSQL забезпечує стабільне збереження даних гравців, прогресу та досягнень. Ігри, створені з використанням цих технологій, надають задоволення користувачам будь-якого віку та ігрового досвіду. Розробники можуть також працювати над поліпшенням графіки, звукового супроводу та розширенням до інших платформ. Використання pygame та PostgreSQL відкриває широкі можливості для розробників та надає гравцям високоякісний ігровий досвід.

Отже, які шляхи вирішення даної проблеми є у наявності:

1) самостійно створити ігровий застосунок, який буде мати необхідний функціонал;

2) придбати необхідну гру самостійно.

Але великою проблемою є висока вартість гри та не готовність створення проєкту за ту чи іншу вартість. Також проблемою є можливість закриття студії розробки або зупинення його оновлення, у цьому разі користувачу необхідно шукати новий ігровий застосунок та знов його купувати. Виходячи з цього, було вирішено створити гру, котра б мала розширений функціонал і була доступна усім бажаючим.

Актуальність дослідження зумовлена потребою та прагненням людей пограти в культову та всім відому гру у цифровому форматі.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження.

Мета: розробити ігровий застосунок «Alien Invasion» з інтерактивними графічними ефектами та базою даних для збереження рекордів гравців.

Задачі:

1) сформувати вимоги до ігрового застосунку;

2) спроєктувати графічний інтерфейс та ігрову механіку;

3) реалізувати ігровий застосунок «Alien Invasion» з використанням бібліотеки Pygame;

4) інтегрувати базу даних PostgreSQL для збереження рекордів гравців;

5) протестувати роботу ігрового застосунку та бази даних.

Об’єкт дослідження: процедура розробки ігрового застосунку засобами Pygame та PostgreSQL.

Предмет дослідження: процес розробки ігрового застосунку «Alien Invasion» з використанням засобів Pygame та PostgreSQL із функціоналом необхідним користувачам.

Методи дослідження: аналітичний, практичний, порівняльний.

Перший розділ присвячено збору та аналізу вимог до ігрового застосунку «Alien Invasion», огляду його функціональності та графічного дизайну.

У другому розділі розглянуто етапи проєктування графічного інтерфейсу та ігрової механіки, наведено детальний опис елементів гри.

Третій розділ присвячено реалізації та тестуванню ігрового застосунку «Alien Invasion» з використанням бібліотеки Pygame, а також інтеграції бази даних PostgreSQL для збереження рекордів гравців.

1 ОСНОВНІ ВИМОГИ ТА ЗАСОБИ РЕАЛІЗАЦІЇ ПРИ РОЗРОБЦІ

1.1 Терміни та визначення

Загальні терміни.

Система – ігровий застосунок написаний на мові програмуванні Python та з використанням бібліотек pygame та psycopg2.

Python – це високорівнева, інтерпретована, загального призначення програмна мова. Вона має простий і зрозумілий синтаксис, що сприяє швидкому розробці програм і полегшує читання коду. Python є динамічною мовою, що означає, що типи змінних не обов’язково вказувати явно, вони визначаються автоматично під час виконання програми [1].

Pygame – це бібліотека (framework) для розробки ігор і візуальних додатків у мові програмування Python. Вона надає набір інструментів і функцій, що дозволяють легко створювати і взаємодіяти з графікою, звуком та іншими мультимедійними елементами. Pygame базується на платформонезалежній бібліотеці SDL (Simple DirectMedia Layer) і надає зручний інтерфейс для роботи з графікою, обробкою подій, анімацією, фізикою та управлінням звуком [2].

Psycopg2 – це бібліотека для з’єднання з базами даних PostgreSQL з використанням мови програмування Python [3].

PostgreSQL – це об’єктно-реляційна система управління базами даних (СУБД) з відкритим вихідним кодом [4].

Гравець – людина, котра пройшла авторизацію та має доступ до гри та системи обліку рекордів гравців.

Технічні терміни.

ІС – інформаційна система.

БД – база даних, місце збереження інформації ІС.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати гру, в яку буде приємно грати, в якій буде відбуватись збереження рекордів, що буде стимулювати вже зареєстрованих та заохочувати нових гравців.

Експлуатаційне призначення системи: гра може експлуатуватись та вдосконалюватись будь-ким, так як вона має відкритий код.

Мета створення системи – розробка гри, яка орієнтована на аудиторію будь-якого віку.

1.2.2 Загальні функціональні можливості системи

Система має надавати гравцям такі можливості:

- авторизація гравців;
- процес гри авторизованих гравців;
- перегляд рекордів.

У наступних оновленнях функціонал буде поповнюватись новими нововведеннями. Це мінімальний набір функцій для стабільної роботи додатку.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Гра повинна відображати інтерфейс користувача коректно на будь-якому пристрої. Також інтерфейс має бути інтуїтивно зрозумілим. Кольорова гамма повинна відповідати космічній тематиці. Прибульці та корабель повинні бути

яскравими та привертати увагу та спокушати нових гравців ознайомитись з грою. Завдяки цьому у гри зросте попит та база гравців буде постійно поповнюватись новими гравцями

1.3.2 Підтримка операційних систем

Система повинна працювати на наступних операційних системах:

- iOS;
- Ubuntu;
- Windows 10;
- Windows 11.

Гра перевірялась на усіх цих системах та не ніяких питань із запуском на будь-якій системі не було в’явлено. Це пов’язано з тим що Python оптимізований для запуску на усіх системах, на яких відбувались випробування.

1.3.3 Вимоги до продуктивності

Гра повинна відображати зміст гри не довше, ніж за 3 секунди. Також процес гри повинен бути стабільний та без “артефактів”.

Ігровий застосунок повинен відправляти запроси до бази даних не довше, ніж 5 секунд. Гра повинна бути оптимізована для слабких пристроїв. Для перевірки продуктивності гри використовувалось дві системи:

- персональний комп’ютер з процесором AMD 3600, AMD Radeon RX 5700 XT 8GB OC, 16 GB ОЗУ;
- ноутбук з процесором Intel I5 7200U, Intel HD Graphics, 8 GB ОЗУ.

1.3.4 Вимоги до безпеки

Для зберігання рекордів необхідно додати базу даних. Була обрана PostgreSQL. Після підключення цієї БД до проєкту необхідно забезпечити надійність збереження інформації о досягненнях гравців. Система не має надавати гравцям вільного доступу до БД. Тобто, гравець не може написати запит до БД з оновленням того або іншого запису у таблиці. Також система не повинна відкриватись не авторизованим гравцям.

1.4 Опис предметної області

Предметна область ігрового застосунку «Alien Invasion» – це науково-фантастичний світ, в якому земляни зіштовхуються з нападом прибульців з космосу. Гравець приймає роль пілота космічного корабля, якому доручено відбити атаку іноземних сил та захистити планету.

Після запуску ігрового застосунку відображається вікно, яке пропонує ввести ім’я, для подальшої взаємодії з грою. Після цього відкривається основне вікно з грою, та гравцю надається можливість почати гру або подивитись свої рекорди та досягнення.

У грі гравець буде взаємодіяти з різними видами ворожих космічних кораблів та інопланетними сутностями. Гра буде включати різноманітні рівні складності та завдання, включаючи захист планетарних об'єктів, боротьбу з босами та спеціальні місії. Гравець отримає можливість грати в одиночному режимі, просуваючись через захоплюючий сюжетний режим.

Предметна область «Alien Invasion» надає можливість гравцеві насолоджуватися швидкими темпами гри, екшеном та адреналіном, досліджувати космічну тематику та здійснювати епічні військові операції проти прибульців. Цей застосунок пропонує захоплюючий геймплей та можливість поринути у захоплюючий світ науково-фантастичної битви.

Ігровий застосунок має можливість записувати ігрові досягнення гравців у спеціальну таблицю. У майбутніх оновленнях буде реалізована можливість підбору найкращих гравців.

1.5 Опис системи

Система ігрового застосунку «Alien Invasion» – це комплексний набір компонентів та механізмів, що забезпечують функціонування гри. Вона включає в себе такі основні елементи:

1) графічний двигун: використовується графічний двигун, який забезпечує відтворення рухів корабля та ворожих сутностей, а також спеціальні ефекти лазерних променів;

2) фізичний двигун: вбудований фізичний двигун моделює рух космічних об'єктів;

3) штучний інтелект: використовуються для керування поведінкою ворожих сил (інопланетні вороги мають власну стратегію, тактику та реагують на дії гравця залежно від ситуації);

4) система керування: гравцю надається зручна система керування, що включає клавіатури (це дозволяє точно керувати космічним кораблем, маневрувати, стріляти та виконувати різні дії).

Система ігрового застосунку «Alien Invasion» створена з метою забезпечення екшенового та захоплюючого геймплейного досвіду, реалістичної графіки та фізики, а також можливості взаємодії з іншими гравцями. Вона дозволяє насолоджуватися незабутніми битвами в космосі та відчувати себе героєм у світі наукової фантастики.

1.6 Огляд інструментів розробки

1.6.1 Python

Python – це високорівнева інтерпретована мова програмування, яка широко використовується для розробки різних типів програмного забезпечення [1].

У контексті гри «Alien Invasion» Python – це мова програмування, яка може бути використана для створення цієї гри.

Python пропонує різні інструменти та бібліотеки, які можуть бути корисними під час розроблення ігор, включно з графічними бібліотеками, як-от Pygame, що широко використовується для створення 2D-ігор.

У грі «Alien Invasion» можна використовувати Python для таких завдань:

– керування ігровим процесом: Python дає змогу створити основну ігрову логіку, включно з керуванням екраном, обробкою введення користувача та оновленням ігрових об’єктів;

– створення ігрових об’єктів: можна використовувати Python для створення ігрових об’єктів, як-от гравець, прибульці та кулі, і визначення їхньої поведінки в грі;

– робота з графікою: за допомогою Python і відповідних бібліотек, як-от Pygame, можна створювати та відображати графічні елементи гри, включно із зображеннями, анімаціями та заднім фоном;

– реалізація ігрової механіки: використовуючи Python, можна визначити правила гри, як-от зіткнення об’єктів, підрахунок очок, рівнів складності та інші ігрові елементи. Це та більш інформації у послані у загальному переліку.

1.6.2 Pygame

Pygame – це бібліотека для розробки 2D ігор мовою програмування Python. Вона надає набір інструментів і функцій, які полегшують створення графічних

ігор, включно з обробленням подій, відображенням зображень, керуванням анімацією, відтворенням звуків і музики, а також обробленням зіткнень та інших ігрових завдань [2].

Pygame побудована на основі більш низькорівневої бібліотеки SDL (Simple DirectMedia Layer) і надає більш високорівневий інтерфейс, що спрощує створення ігор на Python. Вона широко використовується як початківцями-програмістами, так і досвідченими розробниками для створення 2D ігор різних жанрів і складності.

Використання Pygame зазвичай охоплює створення основного ігрового циклу, обробку введення користувача, відображення графіки, оновлення стану гри та взаємодію з ігровими об’єктами. Бібліотека також підтримує використання спрайтів, колізій, звукових ефектів, музики та інших елементів ігрового процесу.

Pygame є вільно розповсюджуваною і відкритою бібліотекою, доступною для різних платформ, включно з Windows, macOS і Linux. Вона має активне співтовариство розробників, які пропонують документацію, підручники, приклади коду і підтримку для допомоги у створенні ігор з використанням Pygame. Більш детально про цю бібліотеку можна дізнатись за посиланням у переліку посилань до дипломної роботи

1.6.3 Psycopg2

Psycopg2 – це бібліотека мовою Python, призначена для роботи з базами даних PostgreSQL. Вона забезпечує зручний доступ до функцій PostgreSQL, даючи змогу розробникам створювати з’єднання з базою даних, виконувати запити, отримувати результати та керувати транзакціями [3].

Деякі основні можливості бібліотеки psycopg2 включають таке.

Встановлення з’єднання з базою даних: psycopg2 надає функції для встановлення з’єднання з базою даних PostgreSQL, включно з вказівкою імені хоста, порту, імені користувача, пароля та імені бази даних.

Виконання SQL-запитів: за допомогою psycopg2 можна виконувати SQL-запити до бази даних PostgreSQL, включно з SELECT, INSERT, UPDATE і DELETE. Бібліотека надає зручні методи для передачі параметрів запиту та отримання результатів.

Робота з результатами запитів: psycopg2 дає змогу отримувати результати SQL-запитів, включно з вибіркою даних із таблиці. Можна витягувати дані, обробляти їх і використовувати у власному додатку Python.

Бібліотека psycopg2 є популярним інструментом для роботи з PostgreSQL у Python і широко використовується для розроблення додатків, пов’язаних із базами даних. Вона надає зручний та ефективний спосіб взаємодії з PostgreSQL з коду на Python. Про всі функції, які використовувались при написані дипломної роботи детально написано у документації до цієї бібліотеки. Посилання знаходиться у переліку

1.6.4 PostgreSQL

PostgreSQL – це потужна реляційна система управління базами даних (СУБД), яка пропонує широкий набір функцій і можливостей для зберігання, організації та обробки структурованих даних [4].

Ось деякі основні характеристики PostgreSQL:

– реляційна модель даних: PostgreSQL ґрунтується на реляційній моделі даних, де дані зберігаються в таблицях з певними схемами і зв’язками між ними, він підтримує різні типи даних, зокрема числа, рядки, дати, часи, бінарні дані та інші;

– потужна мова запитів: PostgreSQL надає повнофункціональну мову запитів SQL, яка дає змогу виконувати різноманітні операції з даними, включно з вибіркою, вставкою, оновленням, видаленням, об’єднанням таблиць, угрупуванням, сортуванням та іншими;

– підтримка транзакцій: PostgreSQL забезпечує механізми транзакцій

для обробки даних, він підтримує ACID-властивості (атомарність, узгодженість, ізольованість, стійкість) для забезпечення надійності та цілісності даних;

– розширюваність: PostgreSQL пропонує можливості для розширення та налаштування, даючи змогу розробникам створювати користувацькі типи даних, функції, оператори, агрегатні функції та інші об’єкти бази даних, це дає змогу адаптувати СУБД під конкретні потреби проєкту;

– підтримка JSON та інших розширень: PostgreSQL має вбудовану підтримку JSON-даних і надає можливості для роботи з ними, включно з індексуванням, запитами та маніпуляціями з JSON-документами, крім того, PostgreSQL підтримує різні розширення, такі як GIS (географічна інформаційна система), повнотекстовий пошук та інші;

– висока продуктивність і масштабованість: PostgreSQL відомий своєю продуктивністю і здатністю ефективно обробляти великі обсяги даних, він підтримує паралельне виконання запитів, багатопоточність, кешування даних та інші оптимізації для забезпечення високої продуктивності.

PostgreSQL є вільно розповсюджуваною і відкритою системою управління базами даних, доступною для різних операційних систем, включно з Windows, macOS і Linux. Для більшої інформації необхідно ознайомитись з посиланням наведеним у переліку посилань до кваліфікаційної роботи.

1.6.5 Docker

Docker – це платформа для віртуалізації на рівні операційної системи, що дає змогу упаковувати додатки та їхні залежності в легковагі та переносні контейнери. Контейнери Docker надають ізольоване оточення для виконання додатків, забезпечуючи узгодженість і переносимість між різними комп’ютерами та операційними системами [5].

Деякі основні поняття та компоненти Docker наведені нижче.

Контейнер: Контейнер – це стандартизоване та ізольоване оточення, у

якому упаковують застосунок і його залежності, включно з файлами системи, бібліотеками та іншими ресурсами. Контейнери Docker використовують одне спільне ядро операційної системи хоста, що робить їх більш легковажними та ефективними порівняно з традиційними віртуальними машинами.

Образ: Образ Docker – це статичний шаблон, на основі якого створюється контейнер. Образ містить усі необхідні компоненти для запуску програми, включно з операційною системою, залежностями, конфігурацією та кодом програми. Образи можуть бути створені вручну або автоматично з використанням файлу конфігурації Dockerfile.

Docker Compose: Docker Compose – це інструмент для визначення та запуску багатоконтейнерних додатків. За допомогою файлу конфігурації docker-compose.yml можна визначити безліч контейнерів, їхні залежності та налаштування мережі для запуску та управління комплексними додатками.

Переваги Docker включають легковажність, переносимість, масштабованість, ізоляцію додатків і полегшене розгортання. Docker став популярним інструментом у розробці програмного забезпечення, оскільки полегшує управління залежностями та забезпечує консистентність роботи додатків у різних оточеннях. В даному проєкті Dockrer використовується для поліпшення роботи з PostgreSQL та виключення встановлення БД на комп’ютер задля економії простору на дисках. Для більш детального розуміння Docker необхідно прочитати документацію наведену у посиланнях до роботи.

2 ПРОЄКТУВАННЯ ІГРОВОГО ДОДАНКУ

2.1 Використання UML під час розробки системи

UML-діаграма для гри Alien Invasion може бути корисним інструментом для проєктування та візуалізації структури та взаємодії компонентів гри. Нижче наведемо кілька причин, чому UML-діаграма може бути корисною.

Розуміння структури. UML-діаграма класів дає змогу візуалізувати класи, їхні атрибути та методи, а також зв'язки між ними. Це допомагає розробникам і дизайнерам гри краще зрозуміти, які компоненти гри існують і як вони пов'язані один з одним.

Організація коду. Використання UML-діаграми класів допомагає структурувати код гри. Вона допомагає визначити класи та їхні взаємозв’язки, що може суттєво спростити процес розроблення та підтримки кодової бази.

Комунікація та спільна робота. UML-діаграма є графічним засобом комунікації між розробниками, дизайнерами та іншими учасниками команди. Вона допомагає краще зрозуміти вимоги та концепцію гри, а також полегшує обговорення та взаємодію між різними учасниками команди.

Попередній аналіз і проєктування. UML-діаграма може бути використана для проведення попереднього аналізу гри та проєктування її архітектури. Вона допомагає виокремити основні компоненти та функціональність гри, а також визначити інтерфейси та взаємодії між ними.

Документація. UML-діаграма може слугувати документацією до гри. Вона дає змогу описати структуру і поведінку компонентів гри, що може бути корисно для майбутніх розробників, які підтримують і розширюють дану гру.

Загалом, UML-діаграма для гри Alien Invasion допомагає візуалізувати та структурувати компоненти гри, полегшує комунікацію в команді та сприяє кращому розумінню і проєктуванню ігрового процесу.

2.2 Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) для гри Alien Invasion дає змогу ідентифікувати основних дійових осіб (акторів) і функціональність гри. На рисунку 2.1 представлена діаграма прецедентів для Alien Invasion.

Рисунок 2.1 – Діаграма прецедентів для Alien Invasion

Пояснення до діаграми:

– «Alien» (Прибулець): актор, який представляє прибульця в грі, взаємодіє з грою шляхом нанесення ушкоджень гравцеві;

– «Player» (Гравець): актор, який представляє гравця в грі, взаємодіє з грою, стріляючи в прибульців;

– «Game Manager» (Менеджер гри): керує основною логікою гри, включно з керуванням об’єктами, обробкою взаємодії гравця і прибульців, а також відстеженням поточного стану гри;

– «Score Tracker» (Відстеження рахунку): відповідає за відстеженням та оновленням рахунку гравця.

Діаграма прецедентів допомагає візуалізувати основні взаємодії між

акторами та системою гри. Вона дає змогу краще зрозуміти функціональність гри та допомагає визначити ключові сценарії використання.

2.2.1 Опис варіантів використання

2.2.1.1 Прецедент «Вхід у систему»

Призначення: даний варіант використання надає можливість гравцю увійти до системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить ігрове ім’я, та натискає кнопку «Вхід». Система перевіряє, чи є користувач в БД, якщо так, то переадресовує до сеансу гри.

Альтернативний потік: якщо користувач з таким ім’ям не зареєстрований, то система автоматично зареєструє його у системі.

2.2.1.2 Прецедент «Сеанс гри»

Призначення: даний варіант використання надає можливість гравцю почати сеанс гри.

Основний потік подій: даний варіант використання починає виконуватися, коли гравець натискає на кнопку «Почати гру».

Передумова: перед початком виконання даного варіанта використання гравець повинен виконати вхід до системи.

2.2.1.3 Прецедент «Перегляд рекордів»

Призначення: даний варіант використання надає можливість гравцю

подивитися свої досягнення у грі.

Основний потік подій: даний варіант використання починає виконуватися, коли гравець натискає на кнопку «Переглянути рекорди» до сеансу гри.

Альтернативний потік подій: даний варіант використання починає виконуватись, коли гравець натискає кнопку «Переглянути рекорди» після сеансу гри

Передумова: перед початком виконання даного варіанта використання гравець повинен виконати вхід до системи.

2.3 Діаграма діяльності

Діаграма діяльності (Activity Diagram) для гри Alien Invasion допомагає уявити послідовність дій і потоки управління в грі. Така діаграма діяльності для Alien Invasion відображена на рисунку 2.2.

Пояснення діаграми:

– «Start Game» (Почати гру): початкова точка, з якої починається гра;

– «Initialize Game» (Ініціалізація гри): виконується налаштування початкового стану гри, ініціалізація ігрових об’єктів і параметрів;

– «Game Loop» (Ігровий цикл): представляє основний цикл гри, у якому відбувається оновлення та відображення ігрових елементів;

– «Process Player Input» (Обробка введення гравця): обробляє введення гравця, таке як рух і стрільба;

– «Update Game State» (Оновлення стану гри): оновлює стан гри на основі дій гравця і руху прибульців;

– «Process Alien Movement» (Обробка руху прибульців): визначає та обробляє рух прибульців по ігровому полю;

– «Process Collisions» (Обробка зіткнень): перевіряє зіткнення між гравцем, прибульцями та іншими об’єктами в грі й обробляє їх;

– «Check Game Over» (Перевірка на кінець гри): перевіряє умови

завершення гри, такі як втрата гравця або знищення всіх прибульців;

– «Display Game Over» (Відображення закінчення гри): відображає відповідний екран, що інформує гравця про закінчення гри;

– «End Game» (Завершення гри): завершує гру і переходить у вихідний стан або виходить із гри.

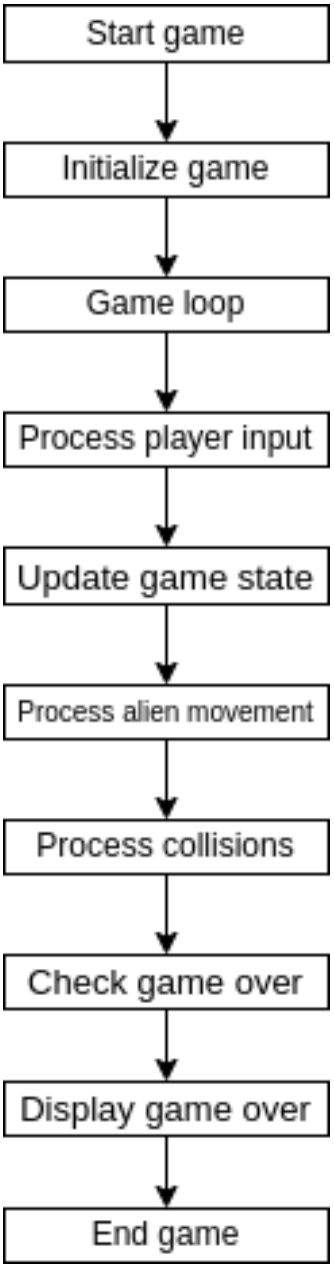

Рисунок 2.2 – Діаграма діяльності для Alien Invasion

Діаграма діяльності допомагає візуалізувати послідовність кроків і потік управління в грі, від початку до завершення. Вона допомагає зрозуміти, як гра взаємодіє з гравцем і керує своїм станом і логікою.

2.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) для гри Alien Invasion допомагає показати взаємодію між різними об’єктами та компонентами гри в часовій послідовності. На рисунку 2.3 така діаграма послідовності для Alien Invasion.

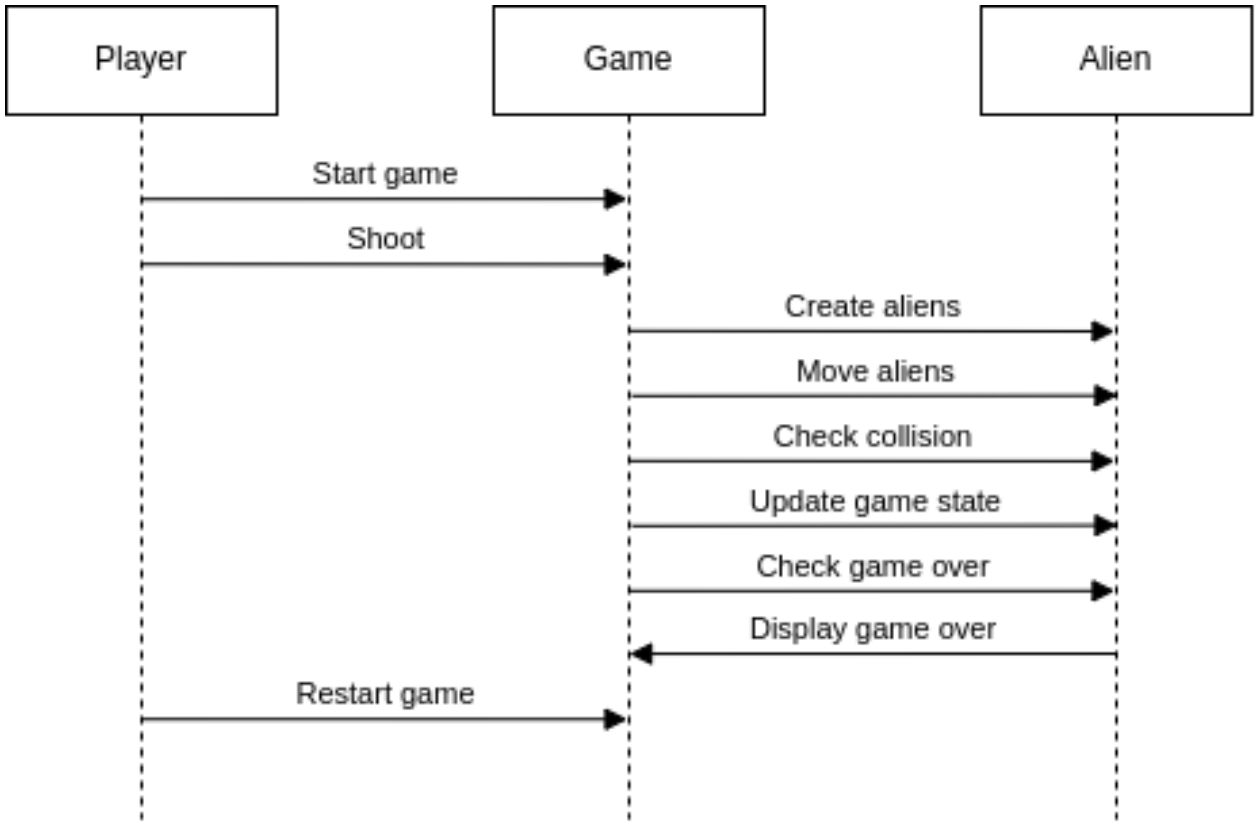

Рисунок 2.3 – Діаграма послідовності для Alien Invasion

Пояснення діаграми:

– «Player» (Гравець): представляє дії гравця, такі як початок гри, постріли та перезавантаження гри;

– «Game» (Гра): відповідає за управління грою, включно зі створенням прибульців, їхнім рухом, опрацюванням сутичок і оновленням стану гри;

– «Alien» (Прибулець): представляє прибульців у грі та їхні дії, такі як створення і рух.

Стрілки вказують на напрямок передачі повідомлень і викликів між

об’єктами.

Взаємодія починається з дій гравця, які спричиняють відповідні дії в грі, як-от створення прибульців, опрацювання зіткнень і оновлення стану гри.

Після кожного кроку гри відбувається взаємодія між грою і прибульцями для виконання відповідних дій, таких як рух і перевірка зіткнень.

Зрештою, ігровий доданок перевіряє умови завершення гри і відображає відповідний екран закінчення гри.

Після закінчення гри гравець може вирішити перезавантажити гру і процес почнеться знову.

Діаграма послідовності допомагає візуалізувати взаємодію між об’єктами в часовій послідовності і зрозуміти потік управління в грі Alien Invasion.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ ІГРОВОГО ДОДАНКУ

3.1 Опис інструментів розробки

Для реалізації були використані мова програмування Python, бібліотеки pygame та psycopg2. Як база даних була обрана PostgreSQL.

Python – це високорівнева, інтерпретована мова програмування, спочатку розроблена для простоти читання і написання коду. Вона має простий синтаксис і велику стандартну бібліотеку [1].

Pygame – це бібліотека для розроблення комп'ютерних ігор і мультимедійних додатків мовою програмування Python. Вона надає простий і зручний інтерфейс для роботи з графікою, звуком і введенням пристроїв, таких як клавіатура та миша [2].

Psycopg2 – це популярний пакет для роботи з базами даних PostgreSQL мовою програмування Python. Він надає зручні та ефективні засоби для взаємодії з PostgreSQL і виконання запитів до бази даних [3].

3.2 Ієрархія даних проєкту

3.2.1 Генерування флоту прибульців

Гра побудована на нищенні ворожого флоту. Для цього необхідно написати код, який з заданого набору прибульців буде обирати рандомних юнітів та створювати флот. Код, який відповідає за генерування флоту відбувається у файлі alien.py. Більш детальну інформацію про генерацію флоту та всіляки перевірки можна подивитися у додатку Г. Поведінка руху та перевірка на досягнення краю екрану відбувається у файлі settings.py.

3.2.2 Відображення корабля гравця

Призначення клавіш відповідальних за рух корабля ліворуч та праворуч відбувається у файлі ship.py.

Код цього файлу наведено у додатку В.

Основні налаштування корабля відбувається також у цьому файлі (відцентрування його по середині екрану на початку гри, оновлення його позиції).

У подальших оновленнях буде доданий функціонал щодо змінення клавіш та підключенню геймпаду для більш зручного керування.

3.2.3 Кулі

У корабля гравця є можливість стріляти з кулеметів кулями, які вражають прибульців.

Поведінка куль, перевірки на зіткнення з прибульцями та контроль швидкості та кількості відбувається у файлі bullet.py.

Код надано у додатку Б.

3.2.4 Файл налаштувань

У цьому файлі прописані основні налаштування екрану, фізика кулі, прибульців.

Пришвидшення прибульців та змінення їх вартості також знаходяться у цьому файлі.

У подальших оновленнях буде додана можливість змінювати всі налаштування через спеціальне меню, в якому всі налаштування будуть зберігатись.

3.2.5 Статистика гравців

Ігрова статистика прописана у файлах ai_stats.py, record.py та statistic_settings.py.

У файлі statistic_settings.py прописані налаштування для створення вікна, в якому відображаються досягнення гравця.

У файлі record.py відбувається запит до БД та вивід отриманої інформації в новостворене вікно для поточного гравця.

У файлі ai_stats.py відбувається ініціалізація вище описаних даних та налаштувань.

3.2.6 Файли, необхідні для проєкту

Для створення елементів інтерфейсу, намалювання кораблів гравців, іконок програм необхідні картинки у растровому форматі .png або .jpg. У папці src/image розташовані зображення прибульців, корабля та іконок для вікон гри.

У подальших оновленнях буде більше варіацій кораблів та прибульців, які можна буде обирати у налаштуваннях.

3.2.7 Робота з базою даних

У файлі db.py створюється зв’язок з базою, перевірки на валідність користувача та вивід рекордів на мові запитів SQL. Код наведено у додатку Д.

Цей файл інтегровано у подальшу програму, для форматування та виводу рекордів та досягнень гравців.

У подальших оновленнях, з міркувань безпеки, всі SQL-запити будуть захищені від сторонніх SQL-ін’єкцій.

3.3 Керівництво користувача

3.3.1 Рівень підготовки користувача

Для роботи з розробленим додатком пересічний користувач неповинен володіти якімось навичками в галузі програмування. Якщо ж деякий користувач захоче додати щось нове до представленого проєкту тоді – так, зробити це без елементарних навичок розуміння програмування буде досить складно.

3.3.2 Підготовка до роботи

Перед запуском системи необхідно пересвідчитись, що python інстальований на машину. Це можна зробити за допомогою команди python3 –version (рис 3.1).

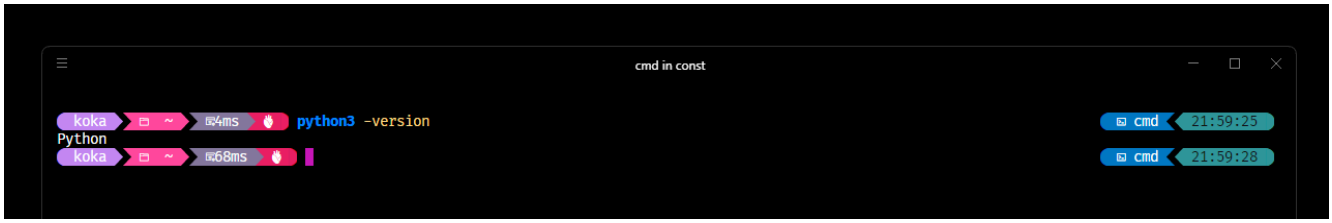

Рисунок 3.1 – Приклад перевірки, чи інстальований Python

Після перевірки наявності Python необхідно перевірити чи є встановлені бібліотеки для подальшої роботи застосунку. Це можна зробити за допомоги команди pip list (рис 3.2).

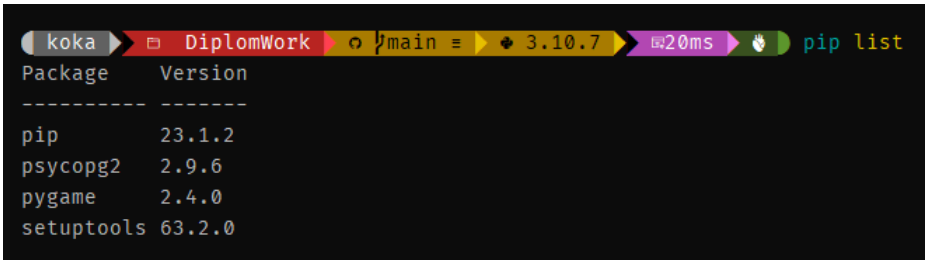

Рисунок 3.2 – Список необхідних бібліотек для роботи додатку

Якщо, необхідних бібліотек не інстальовано, це можна зробити за допомоги команди pip install <назва необхідної бібліотеки>.

Для запуску гри в консолі необхідно написати запит python input_name.py або python3 input_name.py.

3.3.3 Вхід

Після запуску файлу, з’явиться вікно в якому необхідно ввести ігрове ім’я (рис. 3.3).

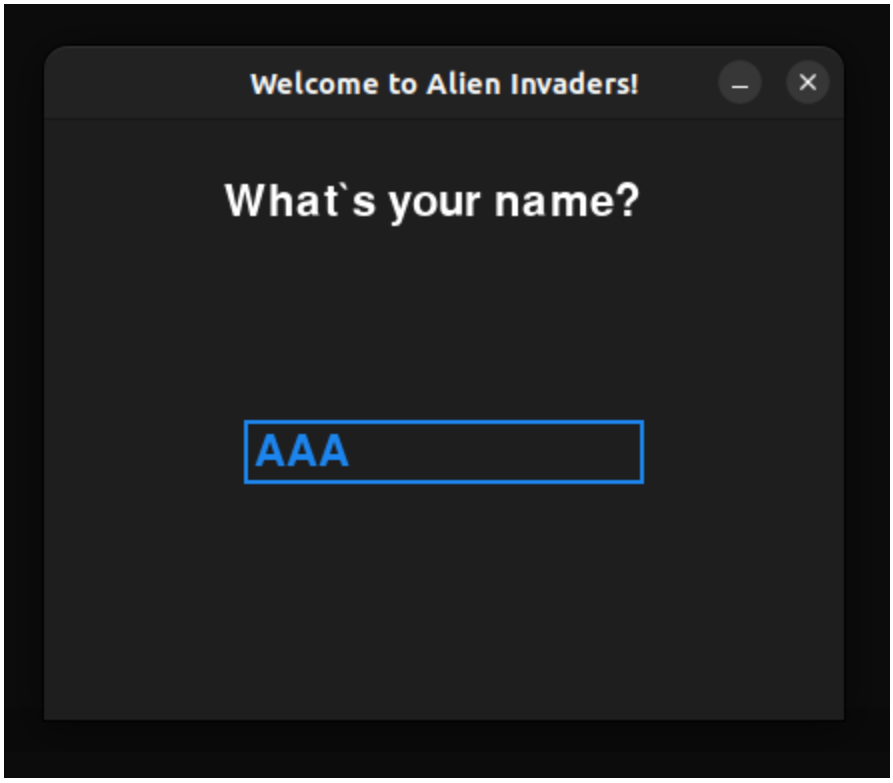

Рисунок 3.3 – Діалогове вікно початку гри Alien Invasion

Під час заповнення форми система буде автоматично перевіряти введені дані на валідність, якщо введені дані невалідні, то система автоматично реєструє користувача у системі та надає доступ до гри.

3.3.4 Сеанс гри

Після перевірки наявності користувача у базі даних та успішній авторизації у системі гравцю надається доступ до основного сеансу гри, в якому він може поринути у космічні пригоди та змагатися разом із друзями (рис. 3.4). Більш детально ознайомитись з програмою можна за посиланням на GitHub[6], яке розташовано у додатку Е.

Рисунок 3.4 – Вікно ігрового доданку Alien Invasion

3.3.5 Рекорди гравця

Після закінчення сеансу у гравця є можливість подивитися свої досягнення. Знизу зліва є кнопка, яка відкриває вікно із досягненнями (рис. 3.5).

Рисунок 3.5 – Результати гри користувача у Alien Invasion

ВИСНОВКИ

В даній кваліфікаційній роботі було розглянуто процес розробки ігрового застосунку з використанням Pygame та PostgreSQL. Дослідження включало ретельний аналіз можливостей та функціональності обох інструментів, а також розробку самого застосунку з використанням цих технологій.

У процесі роботи було проведено детальний аналіз Pygame, що дозволило отримати глибше розуміння його можливостей та принципів роботи. Були досліджені різні аспекти гри, такі як графічне відображення, обробка подій, фізичне моделювання та звукові ефекти. Застосування Pygame дозволило розробити зручну та ефективну ігрову платформу.

Також було вивчено PostgreSQL як систему керування базами даних і розроблено схему бази даних для збереження інформації про гравців, рекорди гри та інші важливі дані. Базуючись на цій схемі, було реалізовано функціонал збереження, оновлення та видалення даних з бази даних.

Результати дослідження та розробки були впроваджені у реальному ігровому застосунку. Застосунок виявився успішним, забезпечуючи зручну геймплейну механіку та ефективне управління даними. Ігровий застосунок заснований на Pygame та PostgreSQL здатний задовольнити потреби користувачів та забезпечити стабільну та надійну роботу.

Отже, розробка ігрового застосунку з використанням Pygame та PostgreSQL є актуальною та перспективною галуззю розвитку програмного забезпечення. Ця робота детально описує процес розробки, аналізуючи інструменти та технології, що дозволяють досягти бажаних результатів. Отримані знання та досвід розробки гри засобами Pygame та PostgreSQL можуть бути корисними для подальшої роботи у галузі геймдеву та розробки відповідного програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційна документація Python. URL: https://docs.python.org/3/ (дата звернення: 12.03.2023).
2. Офіційна документація Pygame. URL: https://www.pygame.org/docs/ (дата звернення: 18.03.2023).
3. Офіційна документація Psycopg2. URL: https://www.psycopg.org/docs/ (дата звернення: 07.04.2023).
4. Офіційна документація PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 07.04.2023).
5. Офіційна документація GitHub. URL: https://docs.github.com/en (дата звернення: 15.03.2023).
6. Офіційна документація Docker. URL: https://docs.docker.com/ (дата звернення: 26.04.2023).

ДОДАТОК А

Головний файл гри, який перевіряє кулі, флот, прибульців і т.д.

```
import sys
import pygame
from time import sleep
from settings import Settings
from ship import Ship
from bullet import Bullet
from alien import Alien
from game_stats import GameStats
from button import Button
from scoreboard import Scoreboard
from button_stats import ButtonStatistic
from statistic.ai_stats import AIStatistic


class AlienInvasion:
    ««««Загальній клас, який керує поведінкою та ресурсами гри»»««

    def __init__(self, user_name):
        ««««Ініціалізує гру та створює ресурси гри»»««

        pygame.init()
        self.settings = Settings()
        # Запуск гри
        img = pygame.image.load('src/image/ufo.png')
        pygame.display.set_icon(img)
        self.screen = pygame.display.set_mode((1800, 900))
```

```
        self.settings.screen_width = self.screen.get_rect().width
        self.settings.screen_height = self.screen.get_rect().height
        # # self.screen = pygame.display.set_mode((self.settings.screen_width,
self.settings.screen_height))
        pygame.display.set_caption(«Alien Invasion»)
        # Створити екземпляр для збереження ігрової статистики та табло на екрані

        self.user_name = user_name
        self.stats = GameStats(self)
        self.sb = Scoreboard(self, user_name)
        self.ship = Ship(self)

        self.bullets = pygame.sprite.Group()
        self.aliens = pygame.sprite.Group()
        # Створити флот прибульців
        self._create_fleet()
        self.statistic = AIStatistic(self)
        # Створити кнопку «Почати гру»
        self.play_button = Button(self, «Почати гру»)
        self.stat_button = ButtonStatistic(self, «Показати статистику»)

    def _check_events(self):
        ««««Реагує на натискання клавіш та миші»««

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                self._check_keydown_events(event)
            elif event.type == pygame.KEYUP:
```

```
                self._check_keyup_events(event)
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mouse_pos = pygame.mouse.get_pos()
                self._check_play_button(mouse_pos)
                self._check_stats_button(mouse_pos)

        pygame.display.flip()

    def _check_stats_button(self, mouse_pos):
        if self.stat_button.rect.collidepoint(mouse_pos):
            button_clicked = self.stat_button.rect.collidepoint(mouse_pos)
            if button_clicked and not self.stats.game_active:
                self.statistic.stat_window(self.user_name, self.stats.high_score,
self.stats.level)

    def _check_play_button(self, mouse_pos):
        ««««Розпочати нову гру, коли користувач натисне кнопку 'Почати гру'««««

        if self.play_button.rect.collidepoint(mouse_pos):
            button_clicked = self.play_button.rect.collidepoint(mouse_pos)
            if button_clicked and not self.stats.game_active:
                # Аналювати ігрову статистику
                self.settings.initialize_dynamic_settings()
                self.stats.reset_stats()
                self.stats.game_active = True
                self.sb.prep_score()
                self.sb.prep_level()
                self.sb.prep_ships()
                # Позбавитися надлишку прибульців та куль
                self.aliens.empty()
```

```
            self.bullets.empty()
            # Створити новий флот та відцентрувати корабель
            self._create_fleet()
            self.ship.center_ship()
            # Приховати курсор миші
            pygame.mouse.set_visible(False)

    def _check_keydown_events(self, event):
        ««««Реагує на натискання клавіші»»««

        if event.key == pygame.K_END:
            sys.exit()
        elif event.key == pygame.K_SPACE:
            self._fire_bullet()
        elif event.key == pygame.K_RIGHT:
            self.ship.moving_right = True
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = True

    def _check_keyup_events(self, event):
        ««««Реагує, коли клавіша не натиснута»»««

        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = False

    def _ship_hit(self):
        ««««Реагувати на зіткнення прибульця з кораблем»»««
```

```
    if self.stats.ships_left > 0:
        # Зменшити ships_left та оновити табло
        self.stats.ships_left -= 1
        self.sb.prep_ships()
        # Позбавитись надлишку прибульців та куль
        self.aliens.empty()
        self.bullets.empty()
        # Створити новий флот та відцентрувати корабель
        self._create_fleet()
        self.ship.center_ship()
        # Пауза
        sleep(0.5)
    else:
        self.stats.game_active = False
        pygame.mouse.set_visible(True)

def _fire_bullet(self):
    ««««Створити кулю та додати її до групи куль»»»»
    if len(self.bullets) < self.settings.bullets_allowed:
        new_bullet = Bullet(self)
        self.bullets.add(new_bullet)

def _update_bullets(self):
    ««««Оновити позицію куль та позбавитись старих куль»»»»

    # Оновити позиції куль
    self.bullets.update()
    # Позбавитись куль, що зникли
    for bullet in self.bullets.copy():
        if bullet.rect.bottom <= 0:
```

```
            self.bullets.remove(bullet)
    self._check_bullet_alien_collisions()

def _check_bullet_alien_collisions(self):
    ««««Реакція на зіткнення куль з прибульцями»»««

    # Виділити всі наявні кулі, що зіткнулися
    colisions = pygame.sprite.groupcollide(self.bullets, self.aliens, True, True)
    if colisions:
        for aliens in colisions.values():
            self.stats.score += self.settings.alien_points * len(aliens)
        self.sb.prep_score()
        self.sb.check_high_score()
    if not self.aliens:
        self.bullets.empty()
        self._create_fleet()
        self.settings.increase_speed()

        # Збільшити рівень
        self.stats.level += 1
        self.sb.prep_level()

def _create_fleet(self):
    ««««Створити флот прибульців»»««

    # Створити прибульців та визначити їх кількість у ряду
    alien = Alien(self)
    alien_width, alien_height = alien.rect.size
    available_space_x = self.settings.screen_width - (2 * alien_width)
    number_aliens_x = available_space_x // (2 * alien_width)
```

```
    ship_height = self.ship.rect.height
    available_space_y = self.settings.screen_height - (3 * alien_height) - ship_height
    number_rows = available_space_y // (2 * alien_height)

    # Створити перший ряд прибульців
    for row_number in range(number_rows):
        for alien_number in range(number_aliens_x):
            self._create_alien(alien_number, row_number)

def _create_alien(self, alien_number, row_number):
    ««««Створити прибульця та поставити його до ряду»»»»

    alien = Alien(self)
    alien_width, alien_height = alien.rect.size
    alien.x = alien_width + 2 * alien_width * alien_number
    alien.rect.x = alien.x
    alien.rect.y = alien.rect.height + 2 * alien.rect.height * row_number
    self.aliens.add(alien)

def _update_screen(self):
    ««««Оновлюється зображення на екрані та перемикається на новий екран»»»»

    # Наново перемальовує екран на кожній ітерації циклу
    self.screen.fill(self.settings.bg_color)
    self.ship.blitme()
    for bullet in self.bullets.sprites():
        bullet.draw_bullet()
    self.aliens.draw(self.screen)
    # Намалювати інформацію про рахунок
```

```
        self.sb.show_score()
        # Намалювати кнопку, якщо гра не активна
        if not self.stats.game_active:
            self.play_button.draw_button()
            self.stat_button.draw_button()
        pygame.display.flip()

    def _update_aliens(self):
        ««««Перевірити, чи флот знаходиться на краю, тоді оновити позиції всіх прибульців у флоті»»««

        self._check_fleet_edges()
        self.aliens.update()
        # Шукати зіткнення куль із прибульцями
        if pygame.sprite.spritecollideany(self.ship, self.aliens):
            self._ship_hit()
            # Шукати, чи котрийсь із прибульців досяг нижнього краю екрана
        self._check_aliens_bottom()

    def _check_aliens_bottom(self):
        ««««Перевірити, чи котрийсь із прибульців досяг нижнього краю екрана»»««

        screen_rect = self.screen.get_rect()
        for alien in self.aliens.sprites():
            if alien.rect.bottom == screen_rect.bottom:
                # Зреагувату, ніби корабель бідбито
                self._ship_hit()
                break

    def _check_fleet_edges(self):
```

```
    ««««Реагує відповідно чи досяг краю екрану один із прибульців»»««

    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    ««««Спуск всього флоту та зміна його напрямку»»««

    for alien in self.aliens.sprites():
        alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1

def run_game(self):
    ««««Розпочинае головний цикл гри»»««

    while True:
        # Слідкуе за поведінкою миші та клавіатури та миші
        self._check_events()
        if self.stats.game_active:
            self.ship.update()
            self._update_bullets()
            self._update_aliens()
        self._update_screen()
```

ДОДАТОК Б

Файл, який відповідає за поведінку кулі та її створення

```
import pygame
from pygame.sprite import Sprite


class Bullet(Sprite):
    ««««Клас для керування кулями, випущених з корабля»»««

    def __init__(self, ai_game):
        ««««Створюється об'єкт bullet у поточній позиції корабля»»««

        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # Створити rect у (0, 0) та задати правильну позицію
        self.rect = pygame.Rect(0, 0, self.settings.bullet_width, self.settings.bullet_height)
        self.rect.midtop = ai_game.ship.rect.midtop

        # Зберігати позицію як десяткове значення
        self.y = float(self.rect.y)

    def update(self):
        ««««Посунути кулю нагору екраном»»««

        # Оновити десяткову позицію кулі
```

```
        self.y -= self.settings.bullet_speed
        # Оновити позицію rect
        self.rect.y = self.y

    def draw_bullet(self):
        ««««Намалювати кулю на екрані»»««
        pygame.draw.rect(self.screen, self.color, self.rect)
```

ДОДАТОК В

Файл, який відповідає за поведінку та створення корабля та перевірку зіткнення корабля з границями екрану та флотом прибульців

```
import pygame
from pygame.sprite import Sprite


class Ship(Sprite):
    ««««Клас для керування кораблем»»««

    def __init__(self, ai_game):
        ««««Ініціалізувати корабель та задати його початкову позицію»»««

        super().__init__()
        self.screen = ai_game.screen
        self.screen_rect = ai_game.screen.get_rect()
        self.settings = ai_game.settings
        # Завантажити зображення корабля та отримати його rect
        self.image = pygame.image.load('src/image/space_ship.png')
        self.rect = self.image.get_rect()
        # Створюється кожний новий корабель по центру екрану
        self.rect.midbottom = self.screen_rect.midbottom
        # Зберегти десяткове значення для позиції корабля по горизонталі
        self.x = float(self.rect.x)
        # Індикатор руху
        self.moving_right = False
        self.moving_left = False
```

```
def update(self):
    ««««Оновити поточну позицію корабля на основі індикатору руху»»««

    # Рух корабля та запобігання виходу його за межі екрану
    # З правого боку
    if self.moving_right and self.rect.right < self.screen_rect.right:
        self.x += self.settings.ship_speed
    # З лівого боку
    if self.moving_left and self.rect.left > 0:
        self.x -= self.settings.ship_speed
    # Оновити об`єкт rect з self.x
    self.rect.x = self.x

def blitme(self):
    ««««Намалювати корабель у його поточному розташуванні»»««

    self.screen.blit(self.image, self.rect)

def center_ship(self):
    ««««Відцентрувати корабель на екрані»»««

    self.rect.midbottom = self.screen_rect.midbottom
    self.x = float(self.rect.x)
```

ДОДАТОК Г

Файл, в якому створюється флот прибульців та перевіряє зіткнення із стінками вікна та кораблем гравця

```
import random

import pygame
from pygame.sprite import Sprite


def random_alien_in_fleet():
    c = random.randint(0, 2)
    if c == 0:
        return 'src/image/alien_first.png'
    elif c == 1:
        return 'src/image/alien_second.png'
    elif c == 2:
        return 'src/image/alien_third.png'


class Alien(Sprite):
    ««««Клас, що представляє одного прибульця з флоту»»««

    def __init__(self, ai_game):
        ««««Ініціалізувати прибульця та задати його початкове розташування»»««

        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
```

```
    # Завантажує зображення прибульця та отримати його rect
    self.image = pygame.image.load(random_alien_in_fleet())
    self.rect = self.image.get_rect()

    # Створюється кожного нового прибульця з краю екрану
    self.rect.x = self.rect.width
    self.rect.y = self.rect.height

    # Зберегти десяткове значення для позиції прибульця
    self.x = float(self.rect.x)

def check_edges(self):
    ««««Повертає істину, якщо прибулець знаходиться на краю екрану»»««

    screen_rect = self.screen.get_rect()
    if self.rect.right >= screen_rect.right or self.rect.left <= 0:
        return True

def update(self):
    ««««Змістити прибульця праворуч»»««

    self.x += (self.settings.alien_speed * self.settings.fleet_direction)
    self.rect.x = self.x
```

ДОДАТОК Д

Файл, в якому ініціалізується підключення до бази даних

```
from psycopg2 import connect

from psycopg2.extensions import connection as ConnectionType
from psycopg2.extensions import cursor as CursorType



class DataBase:
    def __init__(self):
        self.host = «localhost»
        self.port = 5432
        self.database = «postgres»
        self.user = «postgres»
        self.password = «postgres»
        self.init_connection()
        self.init_cursor()
        self.init_database()

    @property
    def connection(self) -> ConnectionType:
        return self.__connection

    @connection.setter
    def connection(self, value: ConnectionType):
        self.__connection = value

    @property
```

```
def cursor(self) -> CursorType:
    return self.__cursor

@cursor.setter
def cursor(self, value: CursorType):
    self.__cursor = value

def init_connection(self):
    self.connection: ConnectionType = connect(
        host=self.host,
        port=self.port,
        database=self.database,
        user=self.user,
        password=self.password,
    )

def init_cursor(self):
    self.cursor: CursorType = self.connection.cursor()

def init_database(self):
    create_table_query = ««««
        CREATE TABLE IF NOT EXISTS high_scores (
            name varchar(50),
            score integer
        );
    »»»»
    self.cursor.execute(create_table_query)
    self.connection.commit()

def check_score(self, table: str, username: str):
```

```
        query_to_print = f»SELECT name, score FROM {table} WHERE name =
'{username}';»
        self.cursor.execute(query_to_print)
        return self.cursor.fetchone()

    def count_query(self, table: str, username: str):
        count_query = f»SELECT COUNT(*) FROM {table} WHERE name =
'{username}';»

        self.cursor.execute(count_query)
        return self.cursor.fetchone()[0]

    def add_query(self, table: str, username: str, score: int):
        query_to_add = f»INSERT INTO {table} (name, score) VALUES ('{username}',
{score});»

        self.cursor.execute(query_to_add)
        print(«Done!»)
        return self.connection.commit()

    def update_query(self, table: str, username: str, score: int):
        check_query = f»SELECT score FROM {table};»
        self.cursor.execute(check_query)
        score_check = self.cursor.fetchone()[0]
        if score > score_check:
            query_to_update = f»UPDATE {table} SET score = {score} WHERE name =
'{username}';»
            self.cursor.execute(query_to_update)
        else:
            query_to_update = f»UPDATE {table} SET score = {score_check} WHERE
```

```
name = '{username}';»
            self.cursor.execute(query_to_update)
        return self.connection.commit()

    def select_function(self, table: str, user_name: str, score: int):
        if self.count_query(table, user_name) > 0:
            self.update_query(table, user_name, score)
        else:
            self.add_query(table, user_name, score)

    def __del__(self):
        if self.cursor:
            self.cursor.close()
        if self.connection:
            self.connection.close()
```

ДОДАТОК Е

Посилання на GitHub

https://github.com/kconstantijne/DiplomWork