МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп’ютерних наук та кібербезпеки

На правах рукопису

МАЛАЩУК ВЛАДИСЛАВ АНДРІЙОВИЧ

ДОСЛІДЖЕННЯ ТА РОЗРОБКА МЕРЕЖЕВИХ ДОДАТКІВ У СФЕРІ АЗАРТНИХ ІГОР

Спеціальність: 122 Комп’ютерні науки
Освітньо-професійна програма: Комп’ютерні науки та інформаційні технологіїКваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ПАСТЕРНАК ЯРОСЛАВ МИХАЙЛОВИЧ,
доктор фізико-математичних наук, професор
кафедри комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри
комп’ютерних наукта
кібербезпеки
від_____2023 р.
Завідувач кафедри

(_____) Гришанович Т. О.

ЛУЦЬК – 2024

ЗМІСТ

ВСТУП

У сучасному швидкозмінному світі, інформаційні технології відіграють ключову роль, впливаючи на всі аспекти нашого існування. Наша присутність в онлайн-середовищі зростає щодня, а вплив цифрових технологій на наше життя стає ще більш помітним і важливим. У цьому контексті, розваги, що є популярними серед людей, також еволюціонують. Зокрема, азартні ігри займають особливе місце серед онлайн розваг.

Популярність азартних ігор в сучасному світі пояснюється декількома факторами. По-перше, розвиток технологій, зокрема Інтернету та мобільних пристроїв, зробив азартні ігри більш доступними для широкої аудиторії. Гравці можуть легко отримати доступ до азартних ігор з будь-якого місця та в будь-який час, що значно розширило їх популярність.

По-друге, азартні ігри викликають емоційні реакції та забезпечують адреналін, що привертає увагу гравців і робить гру захоплюючою та цікавою. Крім того, можливість виграшу привертає багато людей до участі в азартних іграх.

По-третє, зростання конкуренції серед розробників програмного забезпечення для азартних ігор стимулює інновації та технологічний прогрес. Нові технології, такі як використання віртуальної реальності, розширеної реальності, штучного інтелекту та блокчейн-технологій, спрямовані на поліпшення якості геймплею та забезпечення комфорту гравців. Ці інновації можуть створювати нові можливості для взаємодії гравців та покращення їх грального досвіду.

Проте, важливо зазначити, що безвідповідальний підхід до азартних ігор може призвести до серйозних наслідків. Недооцінка ризиків пов'язаних з азартними іграми може спричинити фінансові проблеми, руйнування сімейних відносин, а також призвести до розвитку залежності від гри.

Актуальність теми полягає в тому, що розуміння принципів та механізмів роботи азартних ігор є критично важливим для того, щоб недопустити

негативних наслідків від їх участі. Дослідження та розробка мережевих додатків у сфері азартних ігор має потенціал зробити гральний процес більш прозорим та контрольованим, що в свою чергу сприятиме зменшенню ризику виникнення негативних наслідків та залучить увагу до проблеми безпеки гравців.

Метою даної дипломної роботи є дослідження роботи популярних додатків азартних сайтів з метою розуміння їхнього функціоналу та особливостей. На основі отриманих даних буде проведена розробка аналогів цих популярних онлайн додатків, з використанням мови програмування Python та унікальних бібліотек.

Основні завдання дослідження включають в себе аналіз інтерфейсу та функціоналу популярних азартних сайтів, вивчення механік гри, а також аналіз популярних стратегій та алгоритмів, що використовуються в цих додатках.

Основними **завданнями даної дипломної роботи** є дослідження технології Python та особливих бібліотек мови програмування Python, які використовуються для створення ігор, зокрема "pygame", "pymunk" та інших.

Окрім цього, передбачається розуміння принципів та механік роботи однієї з найбільш популярних азартних ігор. Аналіз функціоналу, інтерфейсу та алгоритмів, що використовуються в цій грі для досягнення успішного геймплею.

В рамках дослідження передбачається розробка аналогу вибраної популярної азартної гри з використанням мови програмування Python та відповідних бібліотек. Створення цього аналогу має на меті глибше розуміння всіх аспектів роботи гри та виявлення її механік.

Головною метою цих завдань є не лише створення аналогу популярної азартної гри, а й привернення уваги до серйозності відповідального ставлення до азартних ігор. Шляхом розуміння і розгляду механік інших азартних ігор можна підвищити обізнаність про їхні ризики та зробити гру більш контрольованою та безпечною для гравців.

Об'єктом дослідження є мова програмування Python та ряд спеціалізованих бібліотек, таких як "pygame" та "pymunk", призначених для розробки ігрових додатків. Python обрано як основну мову програмування через

його простоту використання, багатий вибір бібліотек та широку спільноту розробників.

Крім того, досліджуватимуться можливості роботи з аудіофайлами в ігрових додатках з використанням Python та відповідних бібліотек. Це включає в себе відтворення звуків, музики та інших аудіо ефектів для створення іммерсивного геймплею та поліпшення вражень від гри.

Предметом дослідження є процес розробки однієї з найбільш популярних азартних ігор, а також створення її аналогу на основі подібних механік. Основна увага буде зосереджена на аналізі функціоналу, інтерфейсу та геймплею обраної азартної гри.

Основна мета дослідження полягає у глибокому розумінні принципів роботи азартних ігор, виявленні їхніх ключових елементів та розробці власного аналогу, який відобразить основні механіки та геймплей відомої гри. Дослідження покликане показати якісну роботу з азартними іграми, а також звернути увагу на важливість відповідального та обачного ставлення до азартних розваг.

Проаналізувавши результати написання дипломної роботи, були написані та опубліковані тези на тему «Дослідження та розробка мережевих додатків у сфері азартних ігор». Дані тези були представлені на І міжнародній науково-практичній конференції 2024 «Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем».

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ФУНКЦІОНАЛУ ТА МЕХАНІК РОБОТИ ПОПУЛЯРНИХ ІГРОВИХ СИСТЕМ

1.1 Вступ до ігрових систем.

Ігрові системи є невід'ємною частиною сучасного цифрового світу, але їх вплив і значення виходять за межі простої розваги. Зараз ігрові системи охоплюють широкий спектр платформ і форматів, включаючи спеціальні веб-сайти, на яких зберігаються азартні ігри. Ці сайти створені для того, щоб надати користувачам можливість грати в улюблені ігри з будь-якого місця та в будь-який зручний для них час.

Основною особливістю таких ігрових сайтів є можливість доступу до ігор через Інтернет без необхідності завантаження додаткового програмного забезпечення. Після реєстрації на такому сайті користувач може скористатися спеціальними фільтрами для вибору категорії ігор, що відповідає їхнім вподобанням та інтересам. Приклад такого сайту можна побачити на рис. 1.1.

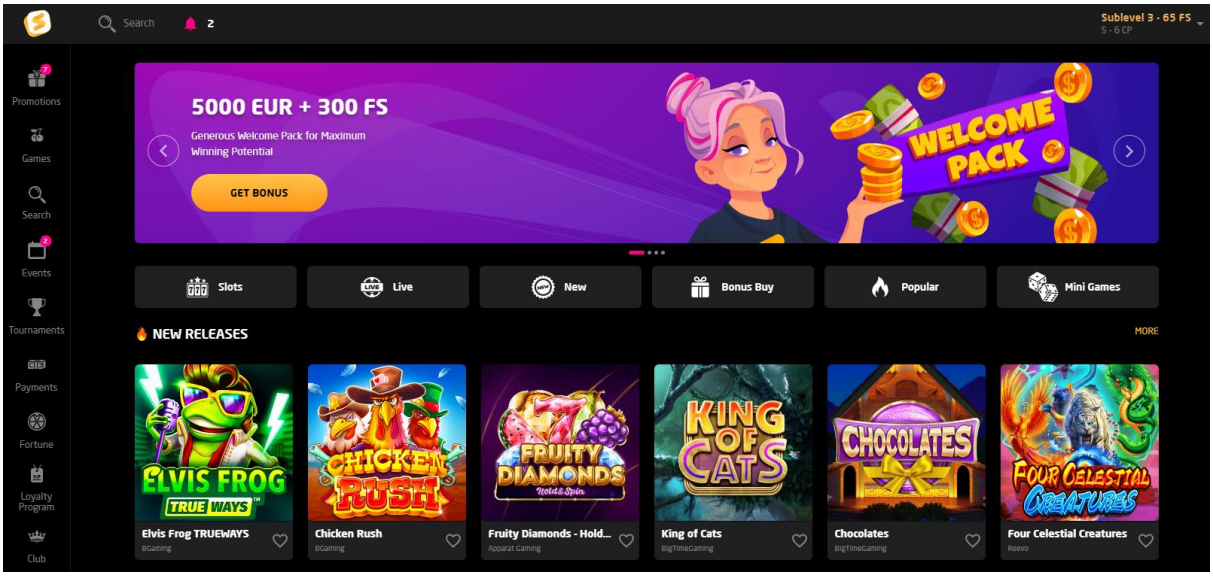

Рис. 1.1 – Сайт, на якому зберігаються ігрові системи

Основними категоріями азартних ігор на таких сайтах є:

1. Слот-машини - це категорія ігор, яка включає в себе широкий вибір

віртуальних ігрових автоматів, які розраховані на випадковий результат. Гравець просто натискає кнопку, і гра випадково призупиняється на одному з символів. Це одна з найпопулярніших форм азартних ігор у віртуальному середовищі.

2. Живі ігри передбачають участь реальних дилерів або крупьє, які проводять гру в реальному часі перед камерою. Гравці можуть ставити на свої улюблені ігри, спостерігаючи за діями дилера в реальному часі. Це дозволяє гравцям відчути атмосферу звичайного казино, граючи зручно з власного пристрою.

3. Аркадні ігри – це категорія ігор, яка включає в себе різноманітні азартні ігри, які базуються на вміннях гравця. Вони можуть включати в себе такі ігри, як покер, блекджек, рулетку та інші, де важливою є стратегія та знання правил гри.

У табл. 1.1 представлено порівняння популярних видів азартних ігрових систем.

Таблиця 1.1

Порівняння популярних видів ігрових систем

Параметр	Слот-машини	Живі ігри	Аркадні ігри
Елемент випадку	Виграш залежить від випадкових символів на барабанах	Виграш залежить від випадкового руху кульки або картки	Виграш залежить від вмінь та стратегії гравця
Рівень стратегії	Низький	Низький	Високий
Взаємодія з гравцем	Мінімальна	Обмежена	Активна
Шанс на виграш	Високий	Середній	Залежить від вмінь
Емоційний вплив	Може викликати хвилювання	Низький, більш фокусується на виграші та програші	Високий, може викликати хвилювання

Соціальний аспект	Часто ізольований гравець, але можуть бути і групові ігри	Обмежений, але може бути соціальним заходом, особливо якщо гра відбувається в реальному часі з іншими гравцями	Часто ізольований гравець, але можуть бути і групові ігри
Потрібна стратегія	Не потрібна	Може залежати від виду гри	Так, особливо в аркадних іграх, де стратегія та вміння гравця визначають результат

Порівнявши різні види азартних ігор - слот-машини, живі ігри та аркадні ігри - можна зробити декілька важливих висновків. Відповідно до нашого аналізу, кожен вид ігор має свої унікальні характеристики і особливості.

Слот-машини відомі своїм високим рівнем випадковості та простотою гри, що робить їх дуже популярними серед гравців, які шукають відпочинок від складних стратегічних рішень. Живі ігри, навпаки, пропонують більш активну взаємодію з гравцем та соціальний аспект, зокрема в режимі реального часу з живими дилерами. Аркадні ігри відображають середній рівень випадковості, але вимагають більше стратегії та вмінь від гравців, тому вони можуть бути цікавими для тих, хто шукає викликів та відчуття досягнення.

Крім того, емоційний вплив і соціальний аспект відрізняються в залежності від типу гри, що важливо враховувати при виборі певного виду розваги. Враховуючи ці різноманітні фактори, гравці можуть обирати ігри, які відповідають їхнім уподобанням, навичкам та очікуванням.

1.2 Огляд загальних принципів функціонування ігрових систем

Огляд загальних принципів функціонування ігрових систем розкриває важливі аспекти, що лежать в основі структури та функціонування різних видів ігор. Це важливо для гравців, розробників і операторів гральних платформ, оскільки глибоке розуміння функціонування ігрових систем допомагає створювати більш привабливі, чесні та безпечні ігрові середовища, що в свою чергу забезпечує задоволення гравців, підвищує довіру до платформи та сприяє розвитку індустрії гральних розваг.

1.2.1 Розбір загальних принципів роботи віртуальних слот-машин

Розглянемо віртуальну слот-машину як один із популярних варіантів азартних ігор. Ці ігрові автомати відображаються на екранах комп'ютерів або мобільних пристроях і пропонують користувачам можливість спробувати удачу та отримати виграш.

Основний принцип дії віртуальної слот-машини полягає в генерації випадкової комбінації символів на віртуальних барабанах. Гравець робить ставку та запускає гру, після чого барабани починають обертатися. Коли вони зупиняються, виграшна комбінація символів, яка може бути заздалегідь визначена відповідно до правил гри, призводить до виплати виграшу. Приклад символів які найчастіше використовуються у віртуальних слот-машинах можна побачити на рис. 1.2.

Важливою частиною віртуальних слот-машин є їхнє програмне забезпечення, яке відповідає за генерацію випадкових результатів та відображення графіки. Гравці можуть відчути азарт та емоції, спостерігаючи за обертанням барабанів та очікуванням на виграш. Мати якісне програмне забезпечення для віртуальних слот-машин є критично важливим для забезпечення стабільності та надійності гри, а також для забезпечення чесності та справедливості результатів.

Ось кілька важливих причин, чому якість програмного забезпечення є

настільки важливою:

1. Якісне програмне забезпечення гарантує випадковість результатів гри. Це означає, що кожна гра має рівні шанси на виграш і що ніхто, навіть розробник, не може передбачити чи маніпулювати результатами.

2. Якісне програмне забезпечення гарантує стабільність гри, уникнення збоїв та недоліків, а також захист від можливих кібератак чи витоків даних, що може загрожувати конфіденційності користувачів.

3. Від якості програмного забезпечення залежить зручність та естетика ігрового процесу. Плавність анімацій, якість графіки та інтерфейсу впливають на задоволення гравців від гри.

4. Якщо гравці не довіряють програмному забезпеченню, вони можуть утримуватися від гри або вибирати інші платформи.

Таким чином, якісне програмне забезпечення є ключовим чинником у побудові довіри гравців до ігрової платформи.

Використання неякісного програмного забезпечення може призвести до негативних наслідків, таких як:

1. Маніпуляція результатами гри на користь розробника.

2. Збої та перебої в роботі гри, що призводять до невдалих гральних сесій та розчарування гравців.

3. Загроза безпеці даних користувачів, включаючи можливість крадіжки особистої інформації та фінансових даних.

Отже, рівень програмного забезпечення для віртуальних слот-машин визначає якість гри, впливає на довіру гравців та забезпечує їхню безпеку та задоволення від гри.

Рис. 1.2. Символи в ігрових системах азартних ігор.

Проведемо аналіз шансів виграти або програти у віртуальних слот-машинах.

Спочатку важливо розрахувати кількість можливих комбінацій символів на барабанах. якщо взяти до уваги віртуальну слот-машину з трьома барабанами, кожен з яких містить десять різних символів, і врахувати, що є двадцять виграшних комбінацій, можна провести аналіз ймовірності виграшу. Загальна кількість можливих комбінацій буде дорівнювати добутку кількості символів на кожному барабані, тобто 10 × 10 × 10 = 1000 комбінацій.

Оскільки слот-машини зазвичай виплачують призи за певні комбінації символів (наприклад, три однакових символи на активній лінії), кількість виграшних комбінацій буде залежати від конкретних правил гри.

Ймовірність виграшу буде визначатися як відношення кількості виграшних комбінацій до загальної кількості можливих комбінацій (рис. 1.3).

$$\text{Ймовірність виграшу} = \frac{\text{Кількість виграшних комбінацій}}{\text{Загальна кількість комбінацій}}$$

Рис. 1.3 – Формула визначення ймовірності виграшу у віртуальній слот машині

Таким чином, ймовірність виграшу обчислюється за такою математичною формулою: 20 / 1000 = 0,02. Це означає, що в кожній грі існує ймовірність 2% на виграш.

1.2.2 Огляд функціоналу живих ігор

Розглянемо функціонал роботи живих ігор. В якості прикладу, опишемо рулетку або колесо фортуни.

Рулетка - це азартна гра, яка базується на обертанні колеса з номерами та розташованими навколо нього кишеньками, в які кулька може впасти після зупинки колеса. Основний принцип роботи рулетки полягає в тому, що гравець робить ставку на конкретний номер або групу номерів, після чого дилер обертає колесо у одному напрямку, а кулька - у протилежному. Після того, як

колесо сповільнюється і кулька впадає в одну з кишеньок, визначається результат гри - чи виграв гравець, чи ні.

Основними характеристиками функціоналу рулетки є:

1. Можливість робити різні види ставок, включаючи ставки на конкретні номери, на кольори (червоний або чорний), на парні або непарні числа та інші.

2. Кожен тип ставки має свої виплати, які зазвичай визначаються ймовірністю випадіння відповідного результату.

3. Людина, яка відповідає за проведення гри, обертаючи колесо та запускаючи кульку.

4. Кожна гра може мати свої правила гри, які визначають деякі деталі, такі як максимальні та мінімальні ставки, додаткові типи ставок, тощо.

Основною принциповою характеристикою роботи рулетки є випадковість результатів, яка забезпечується обертанням колеса та випадковим падінням кульки в одну з кишеньок. Це робить гру чесною для всіх гравців. На рис. 1.4 зображено приклад стандартної рулетки.

Рис. 1.4 – Приклад стандартної ігрової рулетки

Щоб порахувати ймовірність виграшу в рулетці, якщо робити ставку саме на випадіння конкретного числа, спочатку потрібно з'ясувати, скільки всього є номерів на колесі рулетки. Зазвичай у класичній рулетці є 37 номерів: числа від 0 до 36. Однак у деяких варіаціях рулетки може бути 38 номерів, коли є ще й додатковий "нуль".

Тепер, якщо гравець робить ставку на конкретне число, то існує тільки одна виграшна комбінація (відповідне число) серед усіх можливих комбінацій на

колесі. Тому ймовірність виграшу можна обчислити як відношення кількості виграшних комбінацій до загальної кількості комбінацій.

Загальна кількість комбінацій залежить від кількості номерів на колесі. Якщо у нас є 37 номерів, тоді загальна кількість комбінацій буде 37.

Отже, ймовірність виграшу, якщо робити ставку на конкретне число в рулетці, може бути обчислена за формулою: 1 / 37 = 0.027. Шанс виграти, роблячи ставку на конкретне число в рулетці, складає близько 2.7%.

Інший популярний вид живих ігор - колесо фортуни. Це ігрова механіка, яка використовується для випадкового визначення результату. У цій грі є велике колесо з розділами, на яких розташовані різні винагороди або призи. Гравець обирає свій варіант і, після обертання колеса, результат визначається тим розділом, на якому зупиняється стрілка. Колесо фортуни зазвичай використовується для надання різноманітних нагород або випадкових виграшів у різних ситуаціях, включаючи телевізійні програми, розважальні заходи та інші події.

1.2.3 Розбір функціоналу аркадних азартних ігор

Аркадні азартні ігри, такі як гра "Плінко", представляють собою віртуальні версії класичних аркадних ігор, які можна знайти у розважальних закладах та ігрових залах. "Плінко" - це гра, в якій гравець керує рухом кульки через різні перешкоди, з метою досягнення максимальної кількості виграшів.

Головний елемент "Плінко" - це графічна сцена, на якій зображено вертикальну дошку з вузькими зазорами. Внизу дошки розташовані відділи з різними значеннями виграшу. Гравець починає гру, запускаючи кульку з верхньої частини дошки. Кулька пролітає через перешкоди, відбиваючись від стінок, і в кінці завершує свій шлях в одному з відділів з виграшем.

Механіка гри полягає у тому, що кулька рухається згори вниз під впливом гравітації, змінюючи напрямок свого руху при зіткненні з перешкодами. Гравець може впливати на рух кульки, вибираючи кут запуску та контролюючи силу

удару. Це дозволяє гравцеві спрямовувати кульку в бажаному напрямку та спробувати досягти більш вигідних відділів з виграшем.

Основними елементами гри "Плінко" є:

1. Зображення вертикальної дошки з відділами для виграшу.

2. Об'єкт, який рухається по дошці під впливом гравітації та взаємодіє з перешкодами.

3. Елементи, які змушують кульку змінювати свій напрямок руху, наприклад, кільця або шипи.

4. Різні області на дні дошки, кожна з яких має своє значення виграшу.

Гра "Плінко" зараз набирає популярності і вважається однією з найбільш популярних азартних ігор на всіх платформах. Її захопливий та простий для розуміння функціонал привертає увагу гравців різного віку та досвіду, що робить її привабливою як для початківців, так і для досвідчених гравців. У зв'язку з цим, функціонал гри "Плінко" буде детально досліджено та реалізовано в рамках цієї дипломної роботи з метою розуміння його механіки та використання в подальшому розробленні схожих азартних ігор. Приклад гри "Плінко" зображено на рисунку 1.5.

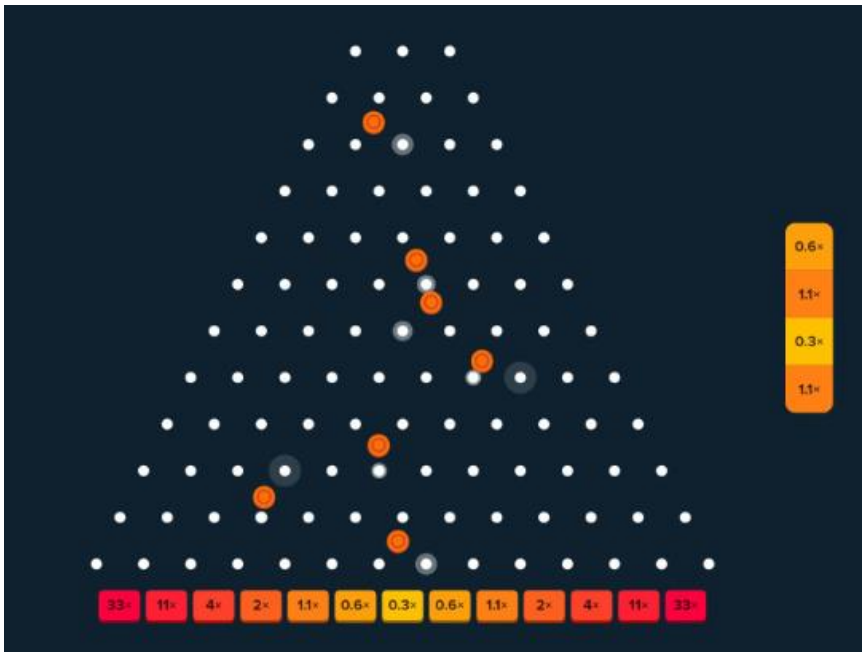

Рис. 1.5 – Приклад популярної азартної гри "Плінко"

Щоб розрахувати шанси виграшу та програшу в стандартній грі "Плінко", потрібно спочатку з'ясувати кількість виграшних комбінацій та загальну кількість можливих варіантів, а потім застосувати ймовірнісний підхід. Розрахуємо шанс виграшу в грі "Плінко" з урахуванням таких умов: є 13 відділів, 3 з яких є програшними, тобто розташовані в центрі, і всі інші є виграшними. Також врахуємо те, що всі кульки кидаються по центру, і для того, щоб потрапити на виграшний відділ, кульці спочатку потрібно відбитися від перешкод у сторону виграшних відділів.

У цьому випадку, кулька може виграти, якщо успішно пройде через перешкоди і приземлиться в одному з 10 виграшних відділів. Щоб розрахувати це, ми спочатку визначимо загальну кількість можливих шляхів, якими кулька може пройти через перешкоди, а потім порівняємо це з кількістю виграшних відділів. Загальна кількість можливих шляхів = 102. Кількість виграшних відділів = 10. Ймовірність виграшу = Кількість виграшних відділів / Загальна кількість можливих шляхів = 10 / 102. Ймовірність виграшу ≈ 0.098 (або 9.8%). Отже, шанс виграшу в грі "Плінко" у цьому випадку становить приблизно 9.8%.

Шанс програти в цій грі відповідає ймовірності того, що кулька не пройде через перешкоди і приземлиться в одному з трьох програшних відділів. Ймовірність програшу = 1 - Ймовірність виграшу = 1 - 0.098. Ймовірність програшу ≈ 0.902 (або 90.2%). Отже, шанс програти в грі "Плінко" у цьому випадку становить приблизно 90.2%.

Ці розрахунки вказують на те, що шанс виграшу в грі "Плінко" в цьому випадку становить лише 9.8%, оскільки кульці потрібно успішно пройти через перешкоди, щоб потрапити на виграшний відділ.

1.3 Мова програмування Python як чудовий інструмент для розробки аркадних азартних ігор.

Мова програмування Python є потужним інструментом, який використовується для розробки різноманітних програм, включаючи аркадні

азартні ігри. Вона вирізняється своєю простотою вивчення та використанням, синтаксисом, який нагадує звичайну англійську мову, що робить її доступною навіть для початківців.

Однією з особливостей Python є його високий рівень абстракції, що дозволяє програмістам швидко писати і тестувати код. Він має велику кількість вбудованих бібліотек і модулів, які полегшують роботу з різними аспектами розробки, такими як графіка, звук, фізика тощо. Також важливою особливістю Python є його спрощена система управління пам'яттю, що робить його більш надійним та стабільним у порівнянні з іншими мовами програмування.

Ці особливості роблять Python ідеальним вибором для розробки аркадних азартних ігор. Він дозволяє розробникам швидко створювати складні ігри з відмінною графікою та фізикою, простим інтерфейсом користувача та великою кількістю можливостей для розширення функціональності гри. Крім того, Python має активну спільноту розробників, яка постійно вдосконалює та розширює інструменти для розробки ігор, що робить його ще більш привабливим для створення азартних ігор.

Аркадну азартну гру можна написати на різних мовах програмування, таких як C++ та JavaScript, які також є потужними інструментами для розробки ігор. C++ є однією з найпоширеніших мов програмування в індустрії ігор завдяки своїй високій продуктивності, потужності та здатності працювати з ресурсами на низькому рівні. Ця мова забезпечує розробникам можливість створювати дуже ефективні та складні ігри, використовуючи потужні графічні двигуни, такі як Unreal Engine та Unity.

JavaScript, у свою чергу, широко використовується для створення веб-ігор. Завдяки своїй популярності в веб-розробці, JavaScript має велику спільноту розробників і потужні інструменти, що дозволяють створювати інтерактивні та крос-платформенні ігри, які можуть працювати безпосередньо у веб-браузерах.

Попри те, що C++ та JavaScript є ефективними мовами програмування для створення ігор, Python все одно має свої переваги, особливо для розробки аркадних азартних ігор. Python вирізняється простотою вивчення та

використання, швидкістю розробки завдяки великій кількості вбудованих бібліотек, а також активною спільнотою розробників, що постійно вдосконалює та розширює інструменти для розробки ігор. Це робить його відмінним вибором для широкого спектру ігрових проектів, зокрема для аркадних азартних ігор.

Нижче наведено порівняльну таблицю мов програмування Python, C++ та JavaScript, що підкреслює їхні основні особливості, переваги а також недоліки у контексті розробки ігор (табл. 1.2).

Таблиця 1.2

Порівняння мов програмування Python, C++ та JavaScript у контексті розробки аркадних ігор

Особливості	Python	C++	JavaScript
Простота	Легкий синтаксис, простий для вивчення та використання.	Складний синтаксис, вимагає більшої уваги до деталей.	Простий для вивчення, але менш потужний в порівнянні з Python.
Оптимізація	Зазвичай менш ефективний у порівнянні з C++, але прийнятний для більшості ігор.	Дуже ефективний, особливо для важких обчислень та графічних застосувань.	Зазвичай менш ефективний для важких обчислень у порівнянні з C++ та Python.
Швидкість розробки	Швидка розробка завдяки високому рівню абстракції та великій кількості бібліотек.	Потужний та швидкий, але вимагає більшої кількості коду та часу для розробки.	Відносно повільна розробка через меншу кількість вбудованих функцій.

Графічний двигун	Підтримка графічних двигунів, таких як Pygame, Pyglet тощо.	Можливість використання потужних графічних двигунів, таких як Unreal Engine.	Відсутність потужних графічних двигунів у порівнянні з C++ та Python.
Крос-платформеність	Широка підтримка для різних операційних систем.	Потужна крос-платформеність, але вимагає компіляції для кожної платформи.	Крос-платформеність за рахунок вбудованих браузерних двигунів.
Спільнота розробників	Велика та активна спільнота розробників, що постійно розширюється та вдосконалюється.	Активна спільнота, але менша у порівнянні з Python.	Велика спільнота, особливо веб-розробників.

1.4 Опис потреб та вимог користувачів аркадних азартних ігор

Аналіз потреб і вимог користувачів дозволяє виявити, які аспекти гри є найбільш важливими для гравців, які функції вони очікують і які елементи гри можуть підвищити їхню залученість. Цей аналіз допомагає зрозуміти, які фактори впливають на рішення гравців продовжувати гру, повертатися до неї знову і знову, та рекомендувати її іншим. Розуміння потреб і очікувань користувачів дозволяє розробникам створювати продукти, які не лише приваблюють гравців, але й забезпечують їх задоволення та утримання. У світі азартних ігор, де конкуренція постійно зростає, задоволення користувачів є ключовим фактором для успішного функціонування та комерційного успіху

ігрових платформ. Крім того, врахування потреб користувачів дозволяє уникнути поширених проблем, які можуть призвести до негативного досвіду гри, таких як незрозумілий інтерфейс, технічні збої, або недостатня різноманітність ігрових можливостей. Це також сприяє створенню більш безпечного та відповідального ігрового середовища, що є важливим для зменшення ризиків, пов'язаних із залежністю від азартних ігор.

1.4.1 Потреби користувачів

Під час розробки аркадної азартної гри необхідно враховувати різноманітні потреби користувачів, щоб забезпечити їм захопливий, привабливий та зручний ігровий досвід.

1. Користувачі очікують, що інтерфейс буде інтуїтивно зрозумілим і простим у використанні. Важливо, щоб усі елементи управління були логічно розташовані, а процес гри був легко зрозумілим без необхідності звертатися до інструкцій.

2. Візуальна привабливість грає ключову роль у залученні та утриманні користувачів. Якісна графіка та плавні анімації роблять гру більш захопливою та цікавою.

3. Звукові ефекти та музика створюють атмосферу гри та впливають на загальний досвід користувача. Важливо, щоб звукове оформлення було якісним і гармонійно доповнювало візуальні елементи гри.

4. Користувачі бажають мати можливість грати в гру, яка пропонує баланс між викликом та розвагою. Гра повинна бути достатньо простою для новачків, але також містити елементи, які стимулюють повернення до гри досвідчених гравців.

5. Стабільність роботи гри є важливою для позитивного досвіду користувача. Технічні збої, довге завантаження або інші проблеми можуть негативно вплинути на задоволення від гри.

6. Враховуючи сучасні тенденції, важливо забезпечити можливість грати у

гру на різних пристроях, включаючи мобільні телефони та планшети. Це забезпечить більшу гнучкість та доступність для користувачів.

1.4.2 Вимоги користувачів

Розробка аркадної азартної гри повинна враховувати конкретні вимоги користувачів, щоб забезпечити відповідність їхнім очікуванням та потребам. Нижче наведені ключові вимоги, які необхідно врахувати.

1. Користувачі потребують чітких та доступних правил гри, які легко зрозуміти з першого погляду. Інструкції повинні бути короткими, але вичерпними, щоб уникнути будь-яких непорозумінь щодо механіки гри.

2. Гра повинна працювати без затримок і технічних збоїв. Це включає швидке завантаження гри, плавну анімацію та стабільну роботу без вильотів чи зависань.

3. Користувачі очікують, що шанси на виграш будуть реалістичними і чесними. Результати гри повинні базуватися на справедливих алгоритмах, що гарантують випадковість і непередбачуваність результатів.

4. Гра повинна бути доступна на різних платформах, включаючи мобільні пристрої, щоб забезпечити максимальну зручність для користувачів. Гра повинна бути оптимізована для різних екранів та операційних систем.

5. Гравці повинні мати можливість легко зв’язатися з підтримкою у випадку виникнення проблем або питань. Наявність ефективної системи зворотного зв'язку підвищить довіру та лояльність користувачів.

Забезпечення цих вимог користувачів є ключовим для створення успішної аркадної азартної гри, яка буде відповідати високим стандартам якості та задовольняти очікування гравців.

1.4.3 Сумарний опис потреб та вимог користувачів

У процесі розробки аркадної азартної гри важливо враховувати потреби та вимоги користувачів для забезпечення приємного і захопливого досвіду.

Користувачі очікують інтуїтивно зрозумілий інтерфейс, що дозволяє швидко та легко розпочати гру, з чіткими і доступними правилами. Важливо, щоб усі елементи управління були логічно розташовані, а процес гри був зрозумілим без потреби в додаткових інструкціях.

Якісна графіка та плавні анімації створюють захопливу атмосферу, а звукові ефекти гармонійно доповнюють візуальні елементи. Користувачі цінують чесність гри, що базується на прозорих алгоритмах, які гарантують випадковість результатів.

Технічна стабільність гри, що працює без затримок та збоїв, є критичною. Гра повинна бути оптимізована для різних пристроїв, включаючи мобільні, що дозволяє грати в будь-який час і місці.

Таким чином, сумарний опис потреб та вимог користувачів включає інтуїтивний інтерфейс, високу якість графіки та звуку, реалістичність та чесність гри, технічну стабільність та ефективну підтримку. Забезпечення цих аспектів є ключовим для створення успішної аркадної азартної гри, яка задовольнятиме очікування користувачів та забезпечуватиме їм приємний ігровий досвід.

1.5 Огляд та аналіз аналогічних програмних розробок

Аналіз аналогічних програмних розробок є ключовим етапом у процесі створення нових ігрових продуктів. Він дозволяє виявити найкращі практики, зрозуміти поточні тенденції та уникнути поширених помилок. У контексті розробки аркадних азартних ігор, такий аналіз допомагає розробникам побачити, які аспекти ігор є найбільш привабливими для користувачів, які функціональні можливості необхідно включити, а також які технічні рішення можуть бути оптимальними.

При аналізі аналогічних програмних розробок важливо звертати увагу на такі аспекти:

1. Якість візуальних елементів та анімаційних ефектів є критично важливою для залучення та утримання гравців.

2. Інтуїтивно зрозумілий інтерфейс і приємний досвід користувача значно підвищують привабливість гри.

3. Принципи та правила гри, зокрема, як відбувається взаємодія гравця з грою, які елементи впливають на результат.

4. Продуктивність гри на різних платформах, відсутність збоїв і помилок.

5. Алгоритми, що забезпечують випадковість результатів, та механізми, що гарантують чесну гру.

6. Можливість гри працювати на різних пристроях і операційних системах.

Процес аналізу передбачає ретельне вивчення існуючих ігор, зокрема:

1. Практичне тестування ігор для оцінки їхньої функціональності та привабливості.

2. Аналіз коментарів та рейтингів на різних платформах для виявлення сильних та слабких сторін.

3. Розгляд використовуваних технологій, архітектурних рішень та методів реалізації ключових функцій.

Загалом, огляд та аналіз аналогічних програмних розробок допомагає створити конкурентоспроможний продукт, який задовольнятиме потреби та вимоги користувачів, забезпечуючи при цьому стабільну та приємну гру.

"Stake Plinko" є популярною аркадною азартною грою на платформі Stake, яка поєднує в собі просту механіку та захоплюючий геймплей. Гра заснована на класичному принципі телевізійної гри "Plinko", де гравці кидають кульку з вершини дошки, що складається з рядів шпильок. Веб-версія гри написана на JavaScript, а серверна частина використовує Python. Кулька випадково відбивається від шпильок і потрапляє в один з відсіків внизу, кожен з яких має різний коефіцієнт виграшу або програшу. Приклади гри можна побачити на рис. 1.6.

Важливим елементом гри є її механіка. Гравець обирає рівень ризику (низький, середній або високий) і кількість рядів (від кількох до декількох десятків). Більша кількість рядів збільшує випадковість результату. Кулька, відбиваючись від шпильок, непередбачувано змінює свій напрямок і зрештою

падає в один з відсіків внизу дошки.

Гра має яскравий і барвистий дизайн, який робить її привабливою для гравців. Висока якість графіки та анімації забезпечує приємний візуальний досвід, підкреслюючи динамічність гри.

Перед запуском кульки гравець може обрати суму ставки. Ця сума визначає потенційний виграш або програш, залежно від того, в який відсік впаде кулька.

Кожен відсік внизу дошки має свій коефіцієнт виплат. Коефіцієнти варіюються залежно від рівня ризику та конкретного відсіку, в який потрапила кулька. Високоризиковані варіанти мають більші потенційні виграші, але ймовірність виграшу в них нижча.

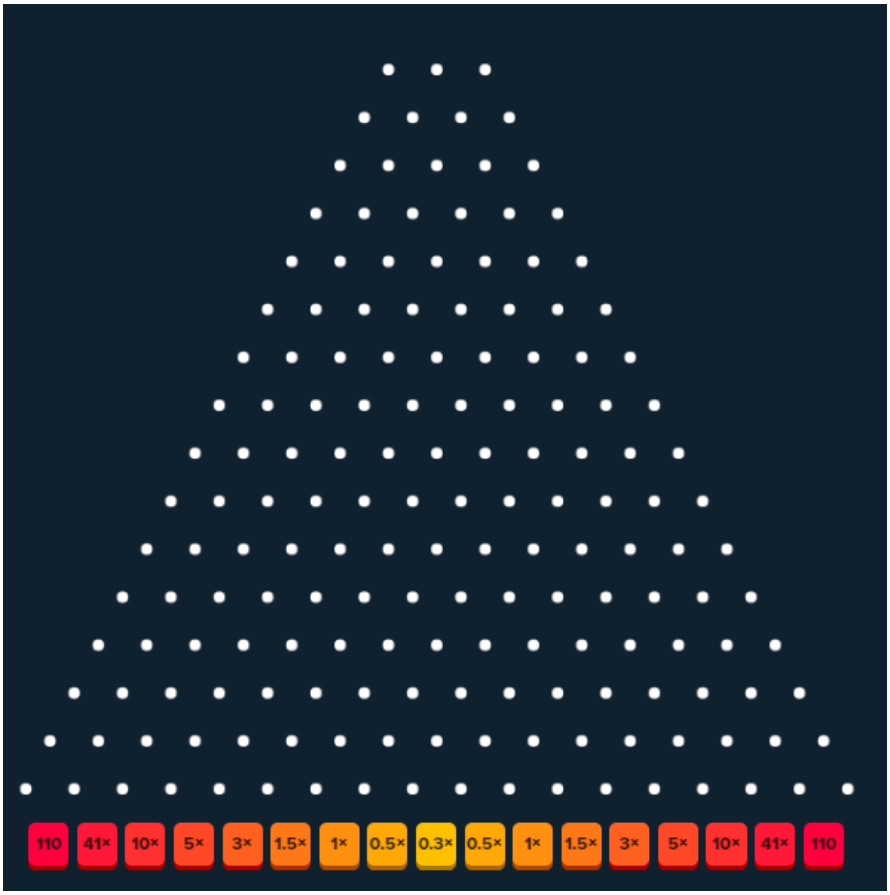

Рис. 1.6 – Аркадна азартна гра «Stake Plinko»

Переваги:

1. Простий і зрозумілий геймплей, який приваблює широке коло гравців.
2. Яскрава та приваблива графіка, що забезпечує приємний візуальний

досвід.

3. Різноманітність рівнів ризику та налаштувань дозволяє гравцям обрати оптимальні умови для гри.

4. Швидкий доступ через веб-платформу, що забезпечує зручність використання.

5. Високий рівень інтерактивності та залученості гравців.

Недоліки:

1. Високий рівень випадковості може відлякувати деяких гравців.

2. Обмежений контроль гравців над результатом, що може знизити відчуття стратегічної гри.

3. Залежність від інтернет-з'єднання для гри у веб-версії.

На рис. 1.7 зображено вікно налаштувань гри, в якому можна обрати рівень складності, який впливає на можливі коефіцієнти, кількість перешкод, та суму ставки.

Рис. 1.7 – Меню налаштувань гри «Stake Plinko»

"Stake Plinko" демонструє, як проста механіка гри може бути адаптована для створення захоплюючого користувацького досвіду. Ця гра є відмінним прикладом успішного поєднання випадковості та стратегії, що робить її популярною серед широкої аудиторії. Аналіз цієї гри дозволяє краще зрозуміти, які елементи роблять аркадні азартні ігри привабливими та успішними, і які аспекти варто врахувати при розробці власного аналогічного продукту.

"Plinko Spribe" - це аркадна азартна гра, розроблена компанією Spribe. Гравець робить ставку та запускає кульку, спостерігаючи, як вона відбивається від перешкод і врешті-решт потрапляє в один із відсіків, що визначає виграш або програш. Веб-версія гри написана на мові програмування JavaScript, а серверна частина використовує Python. На рис. 1.8 зображено приклад популярної аркадної азартної гри "Plinko Spribe".

Особливості гри "Plinko Spribe" включають:

1. Вибір рівня ризику (низький, середній, високий), який визначає потенційні виграші та шанси.

2. Налаштування кількості рядків перешкод на дошці, що впливає на складність гри.

3. Високоякісна графіка та анімація, що створюють привабливий візуальний досвід.

4. Можливість вибору суми ставки перед запуском кульки.

5. Різні коефіцієнти виплат залежно від результату.

Гра приваблює своєю простотою та захоплюючим геймплеєм, надаючи можливість гравцям випробувати удачу та відчути азарт. "Plinko Spribe" популярна серед користувачів різних платформ завдяки своїй доступності та інтерактивності.

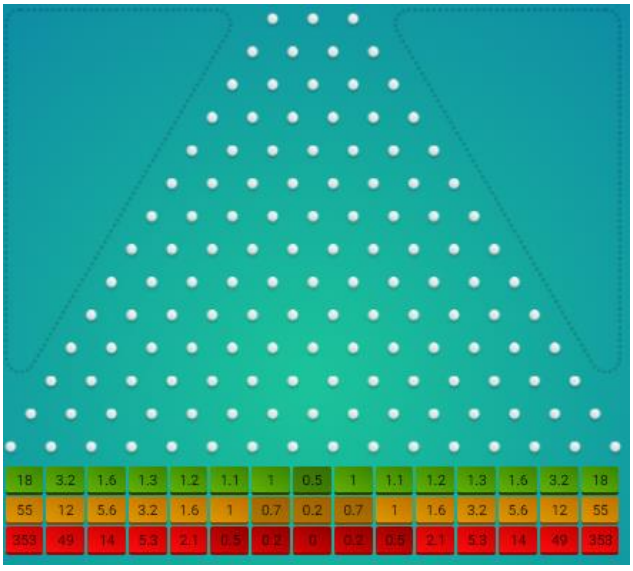

Рис. 1.8 – Аркадна азартна гра "Plinko Spribe"

Гра "Plinko Spribe" демонструє, як можна успішно поєднувати простоту геймплею та привабливий дизайн, створюючи захоплюючий користувацький досвід. Аналіз цієї гри дозволяє зрозуміти, які елементи роблять аркадні азартні ігри привабливими та успішними.

Гра "BC.Game Plinko" від компанії BC.Game є веб-додатком, розробленим з використанням мов програмування JavaScript для фронтенду та Node.js для бекенду. Приклад аркадної азартної гри "BC.Game Plinko" можна побачити на рис 1.9. Гра має доступний та легкий інтерфейс, який зробить гру зрозумілою для будь-якого гравця. Є можливість вибрати рівень ризику, налаштувати кількість рядів дошки та вибрати суму ставки. Що стосується переваг та недоліків, то BC.Game Plinko має свої сильні та слабкі сторони. Серед переваг можна відзначити простий та інтуїтивно зрозумілий інтерфейс, який приваблює новачків і досвідчених гравців однаково. Крім того, швидкий доступ до гри через веб-платформу додає зручності для гравців. Проте, серед недоліків можна відзначити обмежену кількість налаштувань порівняно з іншими версіями гри, а також залежність від стабільності інтернет-з'єднання.

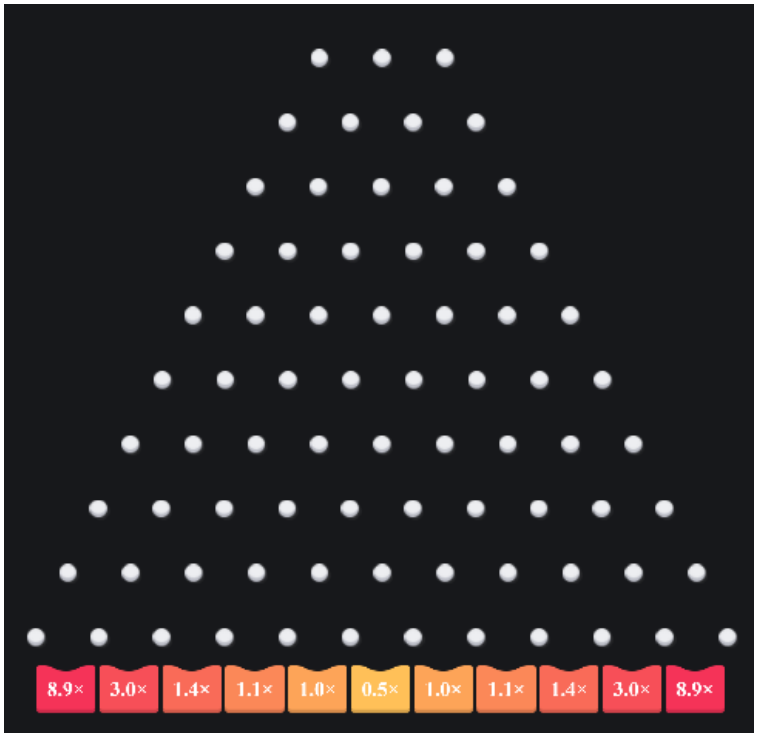

Рис. 1.9 - Приклад аркадної азартної гри "BC.Game Plinko"

Аналізуючи ігри "BC.Game Plinko", "Plinko Spribe" та "Stake Plinko", можна зробити висновок, що кожна з них має свої унікальні особливості та переваги. Вони всі забезпечують користувачам можливість насолоджуватися азартними іграми у форматі плінко, пропонуючи різноманітність функцій та налаштувань для гравців.

Гра "BC.Game Plinko" відзначається своїм зрозумілим інтерфейсом та яскравою графікою, "Plinko Spribe" вражає широким спектром рівнів ризику та налаштувань, а "Stake Plinko" вирізняється найбільшою зручністю використання через веб-платформу.

Однак кожна гра також має свої недоліки. Наприклад, високий рівень випадковості та обмежений контроль над результатом можуть відлякати деяких гравців. Також, залежність від інтернет-з'єднання для гри у веб-версіях може бути несприятливою для користувачів із поганим зв'язком.

У підсумку, кожна з цих ігор пропонує захоплюючий інтерфейс та різноманітність функцій, але кінцевий вибір буде залежати від власних уподобань та потреб гравця.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ТА РОЗРОБКА АНАЛОГУ МЕРЕЖЕВОГО ДОДАТКУ «ПЛІНКО» ЗА ДОПОМОГОЮ МОВИ ПРОГРАМУВАННЯ PYTHON

2.1 Постановка задачі, призначення і вимоги до мережевого додатку «Плінко»

Метою цього проєкту є розробка мережевого додатку, що є аналогом популярної азартної аркадної гри "Плінко". Основним завданням є створення функціонального та зручного інтерфейсу для гри, який буде доступний як на десктопних, так і на мобільних платформах. Програмний продукт повинен забезпечувати гравцям захоплюючий досвід гри завдяки якісній механіці та зручному показу результатів.

Додаток призначений для надання користувачам інтерактивного досвіду гри в "Плінко" в онлайн-середовищі. Він буде використовуватися гравцями, які шукають розвагу та можливість випробувати удачу. Програмний продукт повинен бути доступним через веб-браузери, що забезпечить зручність використання незалежно від типу пристрою.

Основна функціональність гри передбачає реалізацію механіки "Плінко", де кулька падає по дошці з перешкодами і потрапляє у виграшні або програшні відділи. Гравці повинні мати можливість налаштовувати кількість рядів на дошці, що додає варіативності та індивідуалізації гри. Реалістична анімація руху кульки та відбиття від перешкод, а також високоякісні звукові ефекти, забезпечують інтерактивність та підвищують загальне задоволення від гри. Проста і зрозуміла навігація дозволяє гравцям швидко розібратися в інтерфейсі та зосередитися на процесі гри.

Гра повинна надавати зручний показ результатів у реальному часі, що дозволяє гравцям відразу бачити свої виграші або програші. Інтерфейс додатку має бути мінімалістичним та зрозумілим, з адаптацією до різних розмірів екранів. Використання високоякісної графіки та анімації покращує візуальний досвід та робить гру привабливою для гравців.

Інтерфейс повинен бути інтуїтивно зрозумілим, з простотою у навігації та

швидким доступом до налаштувань і історії ігор. Елементи керування мають бути інтуїтивними, що полегшує взаємодію користувача з грою.

Розробка цього додатку вимагає ретельного планування та виконання всіх вимог для забезпечення високої якості продукту, що задовольнить потреби користувачів та надасть їм незабутній ігровий досвід.

2.2 Вибір моделі розробки програмного засобу «Плінко»

Для розробки програмного засобу «Плінко» була обрана мова програмування Python завдяки її універсальності, зручності використання та широкому набору бібліотек, що значно спрощують процес розробки. Python дозволяє швидко створювати прототипи, що є важливою перевагою в процесі розробки інтерактивних ігор.

Для реалізації даного проєкту була обрана модель розробки Agile (рис. 2.1), яка передбачає ітеративний підхід до розробки програмного забезпечення. Agile дозволяє розбивати проєкт на невеликі етапи (ітерації), кожен з яких включає планування, розробку, тестування та оцінку результатів. Цей підхід забезпечує гнучкість і можливість швидко реагувати на зміни вимог або виявлені проблеми.

Основні переваги обраної моделі розробки включають:

1. Agile дозволяє адаптуватися до змін у вимогах та швидко реагувати на зворотний зв'язок користувачів. Це особливо важливо в процесі розробки інтерактивних ігор, де вимоги можуть змінюватися в залежності від реакцій гравців та ринкових тенденцій.

2. Регулярні огляди ітерацій забезпечують прозорість процесу розробки. Можна легко відстежувати прогрес та вносити необхідні корективи.

3. Постійне тестування на кожному етапі розробки дозволяє виявляти та виправляти помилки на ранніх стадіях, що значно підвищує якість кінцевого продукту.

4. Модель Agile передбачає активну участь у процесі розробки, що сприяє кращій координації та комунікації з майбутніми користувачами.

Водночас, існують певні недоліки, пов'язані з обраною методологією розробки:

1. Agile вимагає частих оглядів ітерацій та постійної комунікації з користувачами, що може бути складно організувати за умов обмежених ресурсів.

2. У процесі швидкої розробки та постійних змін вимог може виникнути проблема з недостатньою документацією, що може ускладнити подальшу підтримку та розвиток проєкту.

3. Часті зміни вимог та необхідність швидкого реагування можуть призвести до перевантаження та зниження продуктивності.

Загалом, вибір мови програмування Python та методології Agile для розробки програмного засобу «Плінко» забезпечує оптимальне поєднання гнучкості, швидкості та якості розробки. Цей підхід дозволяє створити продукт, який відповідає потребам користувачів та забезпечує їм високий рівень задоволення від гри.

Рис. 2.1 – Методологія розробки програмного забезпечення «Agile»

2.3 Загальний опис проєкту

Проєкт передбачає створення стандартної аркадної гри «Плінко» з використанням мови програмування Python. Гра являє собою інтерактивну платформу, де гравці можуть кидати кульки на ігрове поле, заповнене перешкодами, з метою потрапити у відсіки виграшів або програшів.

Механіка гри полягає в тому, що гравець натискає на кнопку для запуску гри, після чого з верхньої частини ігрового поля випускається кулька. Ігрове поле складається з кількох рядів перешкод, розташованих у вигляді піраміди. Кулька, взаємодіючи з перешкодами, змінює напрямок руху, що додає елемент випадковості та непередбачуваності до гри.

У нижній частині поля розташовані відсіки виграшів та програшів. Відсіки виграшів приносять гравцеві різні винагороди, залежно від того, куди потрапила кулька. Відсіки програшів, навпаки, не приносять жодних винагород і завершують гру для гравця. Розташування перешкод і відсіків створює унікальний ігровий досвід кожного разу, коли гравець запускає кульку.

Основні компоненти гри включають кнопку для гри, яка запускає процес гри та випуск кульки на ігрове поле. Перешкоди, розташовані у вигляді піраміди, змінюють напрямок руху кульки, створюючи випадковість у грі. Кульки є основним ігровим елементом, який гравець запускає на поле. Вони взаємодіють з перешкодами та в кінцевому підсумку потрапляють у відсік виграшу або програшу. Табло з результатами відображає результати гри, включаючи кількість виграшів і програшів, що дозволяє гравцеві стежити за своїми успіхами.

Проєкт має на меті створити інтерактивну ігрову платформу, яка забезпечить користувачам цікавий і захоплюючий ігровий досвід. Особливу увагу буде приділено якості графіки та анімації, а також інтеграції звукових ефектів, щоб забезпечити максимальне занурення гравця у процес гри. Крім того, зручний інтерфейс та чітке табло з результатами зроблять гру зрозумілою і доступною для широкого кола користувачів.

2.4 Обґрунтування вибору інструментальних засобів розробки

Вибір мови програмування Python для розробки гри «Плінко» є обґрунтованим з кількох причин, серед яких основними є простота у використанні, широкі можливості для розробки ігор та наявність потужних бібліотек.

Python є однією з найпопулярніших мов програмування завдяки своїй простоті та легкості вивчення. Його синтаксис є зрозумілим та інтуїтивним, що дозволяє швидко розпочати розробку та зосередитися на реалізації функціональності гри без необхідності витрачати багато часу на вирішення технічних деталей. Це особливо важливо для проектів, які мають обмежені часові рамки або ресурсні обмеження.

Однією з ключових переваг Python є наявність потужних бібліотек, спеціалізованих для розробки ігор. Зокрема, бібліотеки Pygame та Pymunk.

Важливою перевагою Python є також його кросплатформенність. Розроблена гра може бути легко портована на різні операційні системи, такі як Windows, macOS та Linux, без необхідності значних змін у коді. Це забезпечує ширшу аудиторію користувачів та полегшує тестування і розповсюдження гри.

Python також має добре розвинену екосистему інструментів для розробки, тестування та відлагодження коду. Інтегровані середовища розробки (IDE), такі як PyCharm, надають зручні інструменти для написання, відлагодження та профілювання коду, що значно прискорює процес розробки та підвищує якість кінцевого продукту.

Враховуючи всі ці фактори, вибір Python для розробки гри «Плінко» є оптимальним рішенням, що дозволяє швидко та ефективно реалізувати задуманий функціонал гри, забезпечити високу якість графіки та анімації, а також створити реалістичну фізичну модель руху кульки. Таким чином, Python надає всі необхідні інструменти для створення захоплюючої та інтерактивної аркадної гри, яка задовольнить потреби користувачів та забезпечить приємний ігровий досвід.

2.4.1 Бібліотека Pygame мови програмування Python

Pygame – це бібліотека для мови програмування Python, яка забезпечує набір модулів для розробки повноцінних ігрових додатків. Вона була створена для надання розробникам інструментів, необхідних для розробки інтерактивних ігор, включаючи графіку, звуки та анімацію. Однією з ключових переваг Pygame є її простота у використанні, що дозволяє навіть початківцям розробникам швидко освоїтися та розпочати створення своїх ігор.

Основні можливості та особливості Pygame:

1. Pygame надає модулі для створення та обробки зображень, що дозволяє легко малювати спрайти, створювати фони та реалізовувати складні графічні ефекти. Бібліотека підтримує різні формати зображень, включаючи JPEG, PNG та BMP, що забезпечує гнучкість у виборі графічних ресурсів.

2. Pygame включає модулі для роботи з аудіофайлами, що дозволяє додавати звукові ефекти та фонову музику до ігрового процесу. Підтримка різних форматів аудіо, таких як WAV, MP3 та OGG, дозволяє використовувати будь-які звукові файли без додаткових перетворень.

3. Pygame забезпечує зручну систему обробки подій, яка дозволяє реагувати на дії користувача, такі як натискання клавіш, рухи миші та кліки. Це робить можливим створення інтерактивних елементів управління та динамічних ігрових механік.

4. Завдяки Pygame можна легко створювати анімації, включаючи рух об'єктів, зміну кадрів спрайтів та інші ефекти. Це дозволяє робити ігровий процес більш динамічним та візуально привабливим.

5. Pygame інтегрується з бібліотеками для фізичних симуляцій, такими як Pymunk, що дозволяє створювати реалістичні моделі руху об'єктів та їх взаємодії з оточенням. Це особливо важливо для ігор, де фізика грає ключову роль.

6. Pygame підтримує розширення за допомогою додаткових модулів та бібліотек, що дозволяє додавати нові функціональні можливості до гри. Крім того, Pygame є кросплатформенною бібліотекою, що дозволяє розробникам

створювати ігри для різних операційних систем.

Pygame широко використовується в навчальних закладах та серед незалежних розробників для створення ігор та інтерактивних додатків.

Приклади додатків, створених з використанням Pygame:

«Frets on Fire» - музична гра, в якій гравці використовують клавіатуру як гітару для виконання пісень. Гра була популярною серед фанатів музичних ігор та має активну спільноту моддерів.

«SolarWolf» - аркадна гра, яка є рімейком класичної гри Solar Fox. Гравець керує космічним кораблем, збираючи спеціальні предмети та уникаючи перешкод.

Pygame залишається популярним вибором серед розробників завдяки своїй простоті, гнучкості та потужним можливостям, які вона надає. Вона є чудовим інструментом як для новачків, так і для досвідчених розробників, які бажають створювати високоякісні ігрові проекти.

2.4.2 Бібліотека Pymunk мови програмування Python

Іншою корисною бібліотекою мови програмування Python для створення аркадних азартних ігор є Pymunk – це бібліотека, яка забезпечує простий і зручний інтерфейс для фізичних симуляцій. Вона побудована на основі відомої C-бібліотеки «Chipmunk» і дозволяє створювати реалістичні фізичні моделі для ігрових додатків. Pymunk спеціально розроблена для інтеграції з такими бібліотеками, як Pygame, що робить її ідеальним вибором для розробників, які хочуть додати складні фізичні ефекти у свої ігри.

Основні можливості та особливості Pymunk:

1. Pymunk дозволяє моделювати поведінку твердих тіл у двовимірному просторі. Бібліотека підтримує симуляцію руху, зіткнень, тертя, пружності та інших фізичних властивостей, що дозволяє створювати реалістичні фізичні ефекти.

2. Pymunk підтримує різні геометричні форми для моделювання об'єктів,

включаючи кола, багатокутники та сегменти. Це дозволяє розробникам створювати об'єкти будь-якої складності та форми.

3. Pymunk має простий і інтуїтивно зрозумілий інтерфейс, що полегшує інтеграцію фізичних симуляцій у ігрові додатки. Навіть розробники без досвіду роботи з фізичними симуляціями можуть швидко освоїти бібліотеку та почати використовувати її можливості.

4. Pymunk легко інтегрується з іншими бібліотеками для розробки ігор на Python, такими як Pygame, що дозволяє створювати повноцінні ігрові додатки з багатими фізичними ефектами. Це забезпечує високу гнучкість та модульність при розробці ігор.

5. Pymunk дозволяє моделювати складні фізичні системи з багатьма взаємодіючими об'єктами. Це включає симуляцію механізмів, транспортних засобів, ланцюгів, важелів та інших складних механічних систем.

6. Бібліотека підтримує різні типи обмежень, включаючи пружини, шарніри та шестерні, що дозволяє створювати реалістичні механічні з'єднання між об'єктами. Це особливо важливо для симуляції рухомих частин та складних механізмів.

Pymunk широко використовується для створення ігор та симуляцій, які вимагають реалістичної фізичної поведінки об'єктів.

Приклади проєктів, де використовується Pymunk:

Фізичні головоломки – тобто ігри, в яких гравцям потрібно вирішувати завдання, використовуючи фізичні закони. Наприклад, симуляція руху кульок через лабіринт або створення мостів, які повинні витримувати навантаження.

Ігри, де гравець керує персонажем, що стрибає та взаємодіє з різними об'єктами в ігровому світі. Pymunk дозволяє реалізувати реалістичні рухи та зіткнення персонажа з навколишнім середовищем.

Pymunk є потужним інструментом для розробки ігрових додатків, що потребують реалістичної фізики. Її простий інтерфейс та можливості інтеграції з іншими бібліотеками роблять її незамінним інструментом для розробників ігор будь-якого рівня складності.

2.5 Особливості програмної реалізації та основні режими роботи програмного засобу «Плінко»

Розробка програмного засобу «Плінко» передбачає використання мови програмування Python, яка забезпечує потужний та зручний інструментарій для створення інтерактивних ігрових додатків. У цьому розділі буде детально розглянуто особливості програмної реалізації гри «Плінко», зокрема використання бібліотек Pygame та Pymunk, які відповідають за графічний інтерфейс і фізичну симуляцію відповідно.

Буде наведено структуроване пояснення ключових частин коду, з яких складається програма, включаючи ініціалізацію гри, обробку подій, рендеринг графіки та симуляцію фізичних процесів. Окрім того, буде описано основні режими роботи програмного засобу, такі як запуск гри, налаштування параметрів та відображення результатів.

При розробці гри «Плінко» важливо забезпечити модульність і організованість коду, що сприятиме легкому його підтриманню та розширенню. Одним із найкращих способів досягнення цієї мети є використання об'єктно-орієнтованого підходу, де кожен елемент гри буде представлений окремим класом. У цьому підрозділі ми розглянемо створення окремих класів для ключових компонентів гри: дошки (board.py), кульки (ball.py), перешкод (obstacles.py), секції з результатом (multis.py) та віконця з попередніми результатами.

Для зручного та ефективного редагування часто використовуваних налаштувань програми доцільно створити окремий файл settings.py. Цей файл міститиме всі основні параметри, такі як розміри дошки, швидкість кульок, кольори елементів та інші налаштування, які можна легко змінювати без необхідності редагування основного коду програми.

Основний файл main.py слугуватиме точкою входу в програму, поєднуючи всі елементи гри та забезпечуючи їх взаємодію. У цьому файлі відбуватиметься ініціалізація дошки, кульок, перешкод та інших елементів, а також запуск головного циклу гри, що відповідатиме за оновлення стану гри та її рендеринг.

У файлі main.py для створення та налаштування основного екрану гри використовується кілька важливих функцій. Одна з них: self.screen = pygame.display.set_mode((WIDTH, HEIGHT)).

Ця функція встановлює параметри екрану гри, визначаючи його ширину та висоту. Значення WIDTH та HEIGHT є змінними, що задають розміри вікна гри, що дозволяє визначити розмір екрану, на якому буде відображатися гра. За допомогою цієї функції ми створюємо вікно з вказаними розмірами, яке стане основним полем для відображення всіх графічних елементів гри.

Ще одна важлива функція — self.screen.fill(BG_COLOR). Вона відповідає за заповнення всього екрану певним кольором, що зберігається у змінній BG_COLOR. Ця функція використовується для встановлення фону гри, що забезпечує однорідний колірний фон для всіх інших елементів гри. Це дозволяє уникнути візуальних артефактів та забезпечує естетичну цілісність відображення гри.

Для створення фізичних моделей і симуляцій в грі "Плінко" ми будемо використовувати бібліотеку pymunk. Після імпорту цієї бібліотеки в коді, ми встановлюємо кілька основних налаштувань, що визначають поведінку фізичних об'єктів у грі (рис. 2.2).

```
self.space = pymunk.Space()
self.space.gravity = (0, 1800)
```

Рис. 2.2 – Основні налаштування бібліотеки pymunk

Перший рядок коду створює новий об'єкт простору (Space) у бібліотеці pymunk. Простір (Space) є основним контейнером для всіх фізичних об'єктів і сил у симуляції. В ньому розміщуються всі елементи фізичної моделі, включаючи кульки, перешкоди та інші об'єкти. Простір відповідає за управління їхньою взаємодією та обчисленням фізичних властивостей.

Другий рядок коду встановлює гравітаційне поле для нашого простору. Змінна gravity визначає силу і напрям гравітації, що впливає на всі об'єкти в

просторі. В даному випадку, значення (0, 1800) означає, що гравітація спрямована вниз з силою 1800 одиниць. Це симулює вплив земної гравітації на кульки, змушуючи їх падати вниз по дошці. Значення осі x встановлено в 0, що означає відсутність горизонтальної гравітації, тому кульки падають виключно вниз.

Для налаштування гри "Плінко" використовуються кілька ключових компонентів, які забезпечують управління та відображення об'єктів гри, за допомогою бібліотеки pygame. У цьому розділі коду ми створюємо групу для кульок та ініціалізуємо дошку (рис. 2.3).

```
self.ball_group = pygame.sprite.Group()
self.board = Board(self.space)
```

Рис. 2.3 – Налаштування гри з використанням бібліотеки pygame

Перший рядок створює нову групу спрайтів за допомогою бібліотеки pygame. Група спрайтів (pygame.sprite.Group()) — це спеціальний контейнер для зберігання та управління кількома спрайтами (графічними об'єктами) одночасно. У нашій грі "Плінко", ця група використовується для зберігання всіх кульок, які падають по дошці. Використання групи дозволяє легко керувати всіма кульками як одним об'єктом, наприклад, для їхнього оновлення або відображення на екрані.

Другий рядок ініціалізує об'єкт дошки (Board) та передає йому простір (self.space), створений раніше за допомогою бібліотеки pymunk. Об'єкт дошки відповідає за відображення ігрового поля та розміщення всіх необхідних елементів, таких як перешкоди та відсіки виграшів і програшів. Простір (self.space) використовується для інтеграції фізичної моделі pymunk з відображенням pygame, що дозволяє дошці взаємодіяти з фізичними об'єктами, такими як кульки.

Перейдемо до створення класу Board у грі "Плінко", адже це є ключовим етапом, який забезпечує налаштування ігрової дошки та інтеграцію фізичної моделі. Базові загальні налаштування класу зображені на рис. 2.4.

```
class Board():
    def __init__(self, space):
        self.space = space
        self.display_surface = pygame.display.get_surface()
```

Рис. 2.4 - Базові загальні налаштування класу Board

Метод «__init__» є конструктором класу Board і викликається при створенні нового об'єкта цього класу. Він ініціалізує початкові налаштування дошки.

Перший рядок основного коду встановлює атрибут space для об'єкта Board. space представляє простір pymunk, в якому будуть взаємодіяти всі фізичні об'єкти гри. Простір визначає фізичні властивості, такі як гравітація, і дозволяє моделювати рух і зіткнення об'єктів.

Другий рядок основного коду отримує поточну поверхню відображення pygame і зберігає її як атрибут display_surface. Поверхня відображення (display_surface) представляє основне вікно гри, на якому будуть відображатися всі графічні елементи. Це забезпечує можливість малювати на екрані всі об'єкти, що належать до дошки.

Ці базові налаштування створюють основу для класу Board, забезпечуючи його взаємодію з фізичною моделлю pymunk та графічним інтерфейсом pygame. У подальшому в цьому класі будуть описані додаткові частини коду, які дозволять налаштувати і відобразити всі елементи гри "Плінко".

Для налаштування перешкод у грі, ми визначаємо основні параметри та змінні, які дозволять ефективно розмістити перешкоди на дошці. Ці параметри допоможуть нам використовувати цикл для додавання перешкод на кожному ряді з поступовим збільшенням їх кількості. Відповідний код налаштувань перешкод зображений на рис. 2.5.

```
self.curr_row_count = 3
self.final_row_count = 18
self.obstacles_list = []
self.obstacle_sprites = pygame.sprite.Group()
self.updated_coords = OBSTACLE_START
```

Рис. 2.5 – Налаштування перешкод у грі

Перша змінна визначає початкову кількість перешкод у першому ряді. У нашому випадку, ми починаємо з трьох перешкод, які розміщені в першому ряді. Це забезпечує основу для подальшого збільшення кількості перешкод у наступних рядах.

Друга змінна визначає максимальну кількість перешкод у останньому ряді. У даному випадку, ми встановлюємо, що в останньому ряді буде 18 перешкод. Це значення визначає кінцеву кількість перешкод у нижньому ряді дошки.

Далі створимо порожній список, в якому будуть зберігатися всі об'єкти перешкод. Кожен об'єкт перешкоди буде додаватися до цього списку після його створення та розміщення на дошці.

Створимо групу спрайтів для всіх перешкод. Це зручний спосіб керування всіма перешкодами одночасно. Група спрайтів дозволяє виконувати операції над усіма спрайтами в групі одночасно, наприклад, малювати їх на екрані або оновлювати їх стан.

Останній рядок визначає початкові координати для розміщення першої перешкоди. OBSTACLE_START - це константа, яка містить координати, з яких починається розміщення перешкод на дошці. Ці координати будуть оновлюватися для кожної нової перешкоди, щоб забезпечити правильне розміщення всіх перешкод у рядах.

Функція spawn_obstacle у класі Board відповідає за створення та розміщення перешкод на дошці гри. Ця функція забезпечує налаштування фізичних параметрів перешкоди та додає її до відповідних груп об'єктів у програмі. Код функції зображений на рис. 2.6.

```
def spawn_obstacle(self, pos, space):
    body = pymunk.Body(body_type = pymunk.Body.STATIC)
    body.position = pos
    body.friction = 0.6
    shape = pymunk.Circle(body, OBSTACLE_RAD)
    shape.elasticity = 0.4
    shape.filter = pymunk.ShapeFilter(categories=OBSTACLE_CATEGORY, mask=OBSTACLE_MASK)
    self.space.add(body, shape)
    obstacle = Obstacle(body.position.x, body.position.y)
    self.obstacle_sprites.add(obstacle)
    return shape
```

Рис. 2.6 – Функція «spawn_obstacle»

Першим кроком у функції є створення тіла перешкоди, яке визначає її фізичні властивості. Використовується функція pymunk.Body з параметром body_type=pymunk.Body.STATIC, що означає, що перешкода буде статичною і не буде рухатися під впливом фізичних сил. Позиція тіла встановлюється за допомогою body.position = pos, де pos - це координати розташування перешкоди. Властивість тертя тіла встановлюється за допомогою body.friction = 0.6, що визначає, наскільки поверхня перешкоди буде спротивляти ковзанню кульок при контакті. Форма перешкоди створюється за допомогою pymunk.Circle, яка визначає перешкоду як коло з радіусом OBSTACLE_RAD. Властивість пружності форми встановлюється за допомогою shape.elasticity = 0.4, що визначає, наскільки перешкода буде відштовхувати кульки при зіткненні. Рядок shape.filter = pymunk.ShapeFilter (categories = OBSTACLE_CATEGORY, mask = OBSTACLE_MASK) встановлює фільтр зіткнень для форми. Це потрібно для того, щоб кульки відбивались від перешкод, але не відбивались одна від одної, тобто від інших куль. Перешкода додається до фізичного простору pymunk за допомогою self.space.add(body, shape), що дозволяє програмі враховувати перешкоду під час симуляції фізики. Візуальне представлення перешкоди створюється за допомогою класу Obstacle. Класу передаються координати перешкоди body.position.x та body.position.y. Створений спрайт додається до групи спрайтів перешкод self.obstacle_sprites.add(obstacle), що дозволяє програмі відображати перешкоду на екрані. Наприкінці функція повертає створену форму shape, що може бути використано для подальших налаштувань або обробки.

Таким чином, функція spawn_obstacle забезпечує створення перешкоди з усіма необхідними фізичними та візуальними параметрами, додає її до фізичного простору та групи спрайтів, що дозволяє коректно відображати і обробляти взаємодію з кульками у грі "Плінко".

Функція draw_obstacles (рис. 2.7) відповідає за відображення перешкод на екрані гри.

```
def draw_obstacles(self, obstacles):
    for obstacle in obstacles:
        pos_x, pos_y = int(obstacle.body.position.x), int(obstacle.body.position.y)
        pygame.draw.circle(self.display_surface, (255, 255, 255), (pos_x, pos_y), OBSTACLE_RAD)
```

Рис. 2.7 – Функція «draw_obstacles»

Ця функція приймає список перешкод obstacles і проходить по кожній перешкоді у цьому списку. Для кожної перешкоди визначаються її координати pos_x та pos_y, які відповідають позиції тіла перешкоди в просторі. За допомогою функції pygame.draw.circle кожна перешкода відображається на екрані як коло. Параметри функції включають поверхню для відображення self.display_surface, колір (255, 255, 255) (білий), центр кола (pos_x, pos_y) та радіус OBSTACLE_RAD перешкоди.

Перейдемо до створення класу для кульки. Клас Ball (рис. 2.8) відповідає за моделювання та відображення кульки у грі.

При створенні об'єкта класу Ball, спочатку виконується метод __init__(). Він приймає параметри pos (позиція кульки), space (простір фізики pymunk), board (дошка, на якій кулька рухається), delta_time (часовий крок). Створюється тіло кульки self.body з типом pymunk.Body.DYNAMIC (динамічне тіло). Встановлюється позиція тіла кульки self.body.position відповідно до переданої позиції. Створюється форма кульки self.shape з типом pymunk.Circle, яка пов'язана з тілом self.body та має радіус BALL_RAD. Задаються параметри еластичності, щільності, маси та фільтрації для форми кульки. Фільтр потрібен саме для того, щоб реалізувати функціонал відбиття кульки лише від перешкод,

але не від інших кульок. Тіло кульки та її форма додаються в простір self.space за допомогою методу self.space.add(). Створюється зображення кульки self.image за допомогою pygame.Surface. Встановлюється розмір зображення, який дорівнює BALL_RAD * 2 у ширину і висоту. Встановлюється позиція та розмір прямокутника self.rect, що відповідає зображенню кульки, за допомогою методу self.image.get_rect(). Клас Ball відображає кульку на екрані гри, моделює її фізичні властивості та взаємодіє з іншими елементами гри.

```
class Ball(pygame.sprite.Sprite):
    def __init__(self, pos, space, board, delta_time):
        super().__init__()
        self.display_surface = pygame.display.get_surface()
        self.space = space
        self.board = board
        self.delta_time = delta_time
        self.body = pymunk.Body(body_type = pymunk.Body.DYNAMIC)
        self.body.position = pos
        self.shape = pymunk.Circle(self.body, BALL_RAD)
        self.shape.elasticity = 0.9
        self.shape.density = 10000
        self.shape.mass = 1000
        self.shape.filter = pymunk.ShapeFilter(categories=BALL_CATEGORY, mask=BALL_MASK)
        self.space.add(self.body, self.shape)
        self.image = pygame.Surface((BALL_RAD * 2, BALL_RAD * 2), pygame.SRCALPHA)
        self.rect = self.image.get_rect(topleft=(self.body.position.x, self.body.position.y))
```

Рис. 2.8 – Клас кульки у грі

Для налаштування перешкод у грі, необхідно динамічно створювати їх на дошці у кількох рядах, де кожен наступний ряд буде містити більшу кількість перешкод. Це досягається за допомогою циклу while у класі Board, який забезпечує послідовне розміщення перешкод на дошці та вирішує проблеми з падінням кульок поза межі гри (рис. 2.9).

Спочатку встановлюється початкова координата для перешкод. Далі, цей цикл виконується, поки кількість поточного ряду self.curr_row_count не перевищить кінцеву кількість рядів self.final_row_count. Внутрішній цикл for розміщує перешкоди у поточному ряді. Він виконується self.curr_row_count разів, тобто стільки разів, скільки перешкод має бути в поточному ряді.

Під час виконання циклу for виконуються умови, які визначають позицію перешкод:

1. Зберігання координат для сегмента B;

2. Зберігання першої координати для сегмента A;

3. Зберігання другої координати для сегмента B;

4. Створення перешкоди та додавання її до списку self.obstacles_list за допомогою функції spawn_obstacle.

5. Оновлення координат для наступної перешкоди у поточному ряді.

Після завершення внутрішнього циклу, координати оновлюються для початку наступного ряду та збільшується кількість рядів. Після виходу з циклу while, зберігаються кінцеві координати для мультиплікаторів.

```
self.segmentA_2 = OBSTACLE_START
while self.curr_row_count <= self.final_row_count:
    for i in range(self.curr_row_count):
        # Get first point for segmentB
        if self.curr_row_count == 3 and self.updated_coords[0] > OBSTACLE_START[0] + OBSTACLE_PAD:
            self.segmentB_1 = self.updated_coords
        # Get first point for segmentA
        elif self.curr_row_count == self.final_row_count and i == 0:
            self.segmentA_1 = self.updated_coords
        # Get second point for segmentB
        elif self.curr_row_count == self.final_row_count and i == self.curr_row_count - 1:
            self.segmentB_2 = self.updated_coords
        self.obstacles_list.append(self.spawn_obstacle(self.updated_coords, self.space))
        self.updated_coords = (int(self.updated_coords[0] + OBSTACLE_PAD), self.updated_coords[1])
    self.updated_coords = (int(WIDTH - self.updated_coords[0] + (.5 (parameter) self: Self@Board ted_coords[1] + OBSTACLE_PAD))
    self.curr_row_count += 1
self.multi_x, self.multi_y = self.updated_coords[0] + OBSTACLE_PAD, self.updated_coords[1]
```

Рис. 2.9 – Цикл, який відповідає за динамічне створення перешкод у грі

Під час розробки було помічено, що деякі кульки падали поза межі дошки, потрапляючи на край першої або третьої перешкоди. Це призводило до того, що кульки випадали за межі гри. Щоб вирішити цю проблему, було додано кілька рядків коду (рис. 2.10), які коригують початкові координати перешкод, забезпечуючи правильне розміщення кульок на дошці та запобігаючи їх падінню поза межі ігрового поля.

Таким чином, цей цикл відповідає за правильне та послідовне розміщення перешкод на дошці, забезпечуючи стабільну та передбачувану поведінку кульок у грі.

```
if self.curr_row_count == 3 and self.updated_coords[0] > OBSTACLE_START[0] + OBSTACLE_PAD:
    self.segmentB_1 = self.updated_coords
# Get first point for segmentA
elif self.curr_row_count == self.final_row_count and i == 0:
    self.segmentA_1 = self.updated_coords
# Get second point for segmentB
elif self.curr_row_count == self.final_row_count and i == self.curr_row_count - 1:
    self.segmentB_2 = self.updated_coords
```

Рис. 2.10 – Частина коду, яка відповідає за вирішення проблеми з випадінням кульок за межі дошки

Для налаштування структури гри необхідно створювати додаткові елементи, такі як сегменти, які будуть служити межами для перешкод або іншими важливими елементами, що впливають на поведінку кульок. Функція spawn_segments у класі Board відповідає саме за створення таких сегментів (рис. 2.11). Вона додає статичні сегменти до ігрового простору, визначаючи їхні позиції та фізичні властивості.

Функція spawn_segments відповідає за створення статичних сегментів у грі. Вона приймає три параметри: початкову точку pointA, кінцеву точку pointB, та простір space, у якому буде створений сегмент.

Спершу, створюється статичне тіло segment_body за допомогою pymunk.Body, з вказанням типу тіла як STATIC, що означає, що це тіло не буде рухатися під впливом сил. Далі, створюється форма сегмента segment_shape з використанням pymunk.Segment, де вказуються початкова і кінцева точки та радіус сегмента (у даному випадку 5). Після цього, створене тіло і форма додаються до простору space, що дозволяє їм взаємодіяти з іншими елементами гри. Ця функція корисна для створення меж і додаткових структур на ігровій дошці, які можуть впливати на траєкторії руху кульок, забезпечуючи більш складну і динамічну гру.

```
def spawn_segments(self, pointA, pointB, space):
    segment_body = pymunk.Body(body_type = pymunk.Body.STATIC)
    segment_shape = pymunk.Segment(segment_body, pointA, pointB, 5)
    self.space.add(segment_body, segment_shape)
```

Рис. 2.11 – Функція, яка відповідає за створення сегментів у грі

Для завершення налаштувань структури ігрового поля ми додамо кілька рядків коду, які використовують функцію spawn_segments, щоб створити сегменти, що забезпечують межі для перешкод та додаткові елементи, необхідні для коректного функціонування гри (рис. 2.12).

```
# Segments (boundaries on side of obstacles)
self.spawn_segments(self.segmentA_1, self.segmentA_2, self.space)
self.spawn_segments(self.segmentB_1, self.segmentB_2, self.space)
# Segments at top of obstacles
self.spawn_segments((self.segmentA_2[0], 0), self.segmentA_2, self.space)
self.spawn_segments(self.segmentB_1, (self.segmentB_1[0], 0), self.space)
```

Рис. 2.12 – Створення сегментів на дошці

Перший рядок відповідає за створення сегмента між точками segmentA_1 та segmentA_2. Це встановлює нижню межу перешкод з одного боку дошки. Другий рядок створює сегмент між точками segmentB_1 та segmentB_2, встановлюючи нижню межу перешкод з іншого боку дошки. Третій рядок додає сегмент від верхньої частини ігрового поля (висота 0) до точки segmentA_2, забезпечуючи вертикальну межу для перешкод з одного боку. Четвертий рядок додає сегмент від точки segmentB_1 до верхньої частини ігрового поля (висота 0), встановлюючи вертикальну межу для перешкод з іншого боку.

Ці сегменти забезпечують необхідні межі для гри, запобігаючи випадковому виходу кульок за межі ігрової дошки та забезпечуючи правильну взаємодію з перешкодами. Завдяки цим сегментам, гра стає більш стабільною і передбачуваною, оскільки всі елементи розташовані відповідно до задуму ігрової механіки.

Для завершення налаштувань ігрового поля та додавання елементів, що визначають результат гри, ми перейдемо до створення мультиплікаторів. Мультиплікатори — це відсіки, на які будуть потрапляти кульки, визначаючи виграш або програш гравця. Правильне розташування та налаштування цих мультиплікаторів є важливим для коректної роботи ігрової механіки та забезпечення цікавого ігрового досвіду. Функція spawn_multis відповідає за

створення мультиплікаторів на ігровій дошці (рис. 2.13). Вона визначає, де розташовані мультиплікатори та які значення вони мають.

Функція починається з ініціалізації списків self.multi_amounts та self.rgb_vals. У першому рядку ми створюємо список значень мультиплікаторів, витягуючи їх із ключів multi_rgb. Ці значення визначають, на скільки разів буде збільшено або зменшено ставку гравця при попаданні кульки в конкретний мультиплікатор. Другий рядок створює список RGB-значень кольорів для кожного мультиплікатора, які також витягуються зі списку multi_rgb. Ці кольори використовуються для візуального відображення мультиплікаторів на дошці. Далі, цикл for створює об'єкти мультиплікаторів. Цикл проходить по кількості мультиплікаторів, визначеної змінною NUM_MULTIS. В кожній ітерації створюється новий мультиплікатор. Клас Multi ініціалізується координатами self.multi_x та self.multi_y, кольором self.rgb_vals[i] та значенням мультиплікатора self.multi_amounts[i]. Після створення, мультиплікатор додається до групи multi_group для подальшої обробки та відображення. Координата self.multi_x збільшується на значення OBSTACLE_PAD, щоб забезпечити правильне розташування наступного мультиплікатора. Таким чином, функція spawn_multis створює всі необхідні мультиплікатори на ігровій дошці, забезпечуючи їх правильне розташування та налаштування. Ці мультиплікатори визначають кінцевий результат гри, додаючи їй елемент азарту та стратегічного планування.

```
def spawn_multis(self):
    self.multi_amounts = [val[1] for val in multi_rgb.keys()]
    self.rgb_vals = [val for val in multi_rgb.values()]
    for i in range(NUM_MULTIS):
        multi = Multi((self.multi_x, self.multi_y), self.rgb_vals[i], self.multi_amounts[i])
        multi_group.add(multi)
        self.multi_x += OBSTACLE_PAD
```

Рис. 2.13 – Функція створення мультиплікаторів

Для завершення налаштувань мультиплікаторів, які визначають кінцевий результат гри, створимо клас Multi. Цей клас відповідає за відображення та функціональність мультиплікаторів на ігровій дошці. Кожен мультиплікатор має

свою позицію, колір і значення, яке впливає на результат гри при попаданні кульки.

Клас Multi реалізований як підклас pygame.sprite.Sprite, що дозволяє легко інтегрувати його в ігровий цикл і обробляти всі мультиплікатори як спрайти (рис. 2.14). Ініціалізація класу Multi починається з виклику конструктора батьківського класу pygame.sprite.Sprite. Далі визначаються ключові параметри, такі як позиція (position), колір (color) і значення мультиплікатора (multi_amt). Позиція вказує, де саме на дошці розташований мультиплікатор. Колір використовується для візуальної відмінності між мультиплікаторами, а значення мультиплікатора визначає, який вплив він матиме на результат гри. Наступний крок полягає в налаштуванні відображення мультиплікатора. Задані параметри кольору та радіусу заокруглення кутів створюють естетично привабливий вигляд мультиплікатора. Ширина та висота прямокутника, що представляє мультиплікатор, визначаються відстанню між перешкодами (OBSTACLE_PAD) та висотою мультиплікатора (MULTI_HEIGHT). Після цього створюється поверхня для мультиплікатора з прозорим фоном, на якій малюється прямокутник з заданим кольором і радіусом заокруглення кутів. Параметр multi_amt зберігає значення мультиплікатора, яке буде використовуватися для розрахунку виграшу або програшу, забезпечуючи функціональність гри. Величина prev_multi використовується для правильного розташування наступного мультиплікатора.

```
class Multi(pygame.sprite.Sprite):
    def __init__(self, position, color, multi_amt):
        super().__init__()
        self.display_surface = pygame.display.get_surface()
        self.font = pygame.font.SysFont(None, 26)
        self.color = color
        self.border_radius = 10
        self.position = position
        self.rect_width, self.rect_height = OBSTACLE_PAD - (OBSTACLE_PAD / 14), MULTI_HEIGHT
        self.image = pygame.Surface((self.rect_width, self.rect_height), pygame.SRCALPHA)
        pygame.draw.rect(self.image, self.color, self.image.get_rect(), border_radius=self.border_radius)
        self.rect = self.image.get_rect(center=position)
        self.multi_amt = multi_amt
        self.prev_multi = int(WIDTH / 21.3)
```

Рис. 2.14 – Ініціалізація класу мультиплікаторів

Клас Multi відповідає за створення та налаштування мультиплікаторів на ігровій дошці. Він інтегрує кожен мультиплікатор в ігровий цикл, забезпечуючи інтерактивність ігрового процесу. Завдяки цьому класу, мультиплікатори мають різні кольори, розміри, позиції та значення, що додає варіативності та функціональності грі, роблячи її більш захоплюючою та цікавою для гравців.

Для забезпечення функціональності та візуальної привабливості мультиплікаторів у грі, клас Multi включає функцію render_multi (рис. 2.15). Ця функція відповідає за відображення тексту, що показує значення мультиплікатора на відповідному прямокутнику.

Функція render_multi починається з створення текстової поверхні, яка буде відображати значення мультиплікатора. Це досягається за допомогою методу render об'єкта шрифту (self.font). Значення мультиплікатора (self.multi_amt) форматуються як текст із суфіксом "x" і рендеряться чорним кольором на прозорій поверхні (text_surface). Цей текст символізує, наскільки буде збільшено виграш або програш гравця. Далі визначається прямокутник для текстової поверхні (text_rect), який буде розташований в центрі зображення мультиплікатора. Це гарантує, що текст буде вирівняний і правильно розташований відносно графічного елемента мультиплікатора. Нарешті, текстова поверхня (text_surface) накладається на зображення мультиплікатора (self.image) за допомогою методу blit, що інтегрує текст безпосередньо в графічний елемент.

Таким чином, функція render_multi забезпечує візуальне відображення значення мультиплікатора на відповідному елементі гри, роблячи гру більш зрозумілою та інформативною для гравців. Завдяки цій функції, гравці можуть легко бачити, яке значення мультиплікатора вони отримали.

```
def render_multi(self):
    text_surface = self.font.render(f"{self.multi_amt}x", True, (0, 0, 0))
    text_rect = text_surface.get_rect(center=self.image.get_rect().center)
    self.image.blit(text_surface, text_rect)
```

Рис. 2.15 – Функція, яка відповідає за створення текстової поверхні мультиплікатора

Для підвищення динамічності гри та залученості гравців, мультиплікатори повинні мати анімаційний ефект. Це додає візуальний інтерес і робить гру більш захоплюючою. Анімація включає переміщення мультиплікаторів вниз і назад, створюючи ефект спуску і повернення до початкового положення. Функція animate реалізує цю анімацію, створюючи ефект руху мультиплікатора (рис. 2.16).

Ця функція починається з перевірки, чи кількість анімованих кадрів (self.animated_frames) менша за половину загальної кількості кадрів анімації (self.animation_frames // 2). Якщо ця умова виконується, мультиплікатор рухається вниз на два пікселі за допомогою збільшення нижньої межі прямокутника (self.rect.bottom) на два пікселі. Після досягнення половини кадрів, мультиплікатор починає рухатися вгору, повертаючись до початкового положення. Це досягається зменшенням нижньої межі прямокутника на два пікселі. Таким чином, мультиплікатор рухається вниз і потім назад вгору, створюючи анімаційний ефект. З кожним викликом функції значення self.animated_frames збільшується на один. Коли кількість анімованих кадрів досягає подвоєної половини загальної кількості кадрів анімації ((self.animation_frames // 2) * 2), анімація завершується. Встановлюється прапорець self.is_animating в False, що сигналізує про завершення анімації, і кількість анімованих кадрів скидається до нуля (self.animated_frames = 0). Ця функція забезпечує плавний і природний рух мультиплікатора, що робить гру більш живою та привабливою. Завдяки анімації, гравці отримують візуальний зворотний зв'язок, який підсилює їхнє відчуття залученості в ігровий процес.

```
def animate(self, color, amount):
    if self.animated_frames < self.animation_frames // 2:
        self.rect.bottom += 2
    else:
        self.rect.bottom -= 2
    self.animated_frames += 1
    if self.animated_frames == (self.animation_frames // 2) * 2:
        self.is_animating = False
        self.animated_frames = 0
```

Рис. 2.16 – Функція анімації мультиплікатора

Секція для зберігання результатів останніх кидків є важливим елементом гри "Плінко", оскільки вона надає гравцям можливість швидко переглядати свої попередні результати. Це підвищує загальну зручність гри та надає гравцям корисну інформацію. Клас PrevMulti відповідає за створення та відображення таких результатів на екрані (рис. 2.17).

Клас PrevMulti успадковується від pygame.sprite.Sprite, що дозволяє використовувати його як спрайт у Pygame. У конструкторі (__init__) задаються початкові параметри та налаштовується відображення результату. На початку конструктору викликається ініціалізація базового класу pygame.sprite.Sprite за допомогою super().__init__(). Далі зберігається поверхня відображення (self.display_surface) за допомогою pygame.display.get_surface(). Основні параметри прямокутника (результату) задаються через self.multi_amt, який відповідає за кількість множників, і self.font, який визначає шрифт для відображення тексту. Встановлюються розміри прямокутника (self.rect_width і self.rect_height) та створюється поверхня для відображення результату (self.prev_surf) з використанням pygame.Surface. Кольори множника задаються через параметр rgb_tuple, який зберігається в self.rgb. Для відображення прямокутника використовується pygame.draw.rect, що малює прямокутник на поверхні self.prev_surf з заданим кольором. Прямокутник розташовується за допомогою self.prev_rect, який визначає його положення на екрані. Параметр midbottom задає середню нижню точку прямокутника відносно заданої позиції. Для анімації задаються початкові параметри self.y_traverse та self.traveled, які відповідають за переміщення результату по вертикалі. Ці параметри будуть використовуватись у функції, що відповідає за анімацію результатів. На завершення конструктору викликається функція render_multi(), яка відповідає за відображення тексту результату на поверхні. Вона використовує pygame.font.SysFont для створення текстової поверхні та відображає текст у центрі прямокутника.

```
class PrevMulti(pygame.sprite.Sprite):
    def __init__(self, multi_amt, rgb_tuple):
        super().__init__()
        self.display_surface = pygame.display.get_surface()

        # Rectangle stuff
        self.multi_amt = multi_amt
        self.font = pygame.font.SysFont(None, 36)
        self.rect_width = SCORE_RECT
        self.rect_height = SCORE_RECT
        self.prev_surf = pygame.Surface((self.rect_width, self.rect_height), pygame.SRCALPHA)
        self.rgb = rgb_tuple
        pygame.draw.rect(self.prev_surf, self.rgb, (0, 0, self.rect_width, self.rect_height))
        self.prev_rect = self.prev_surf.get_rect(midbottom=(int(WIDTH * .85), (HEIGHT / 2) - (SCORE_RECT * 2)))

        # Animation
        self.y_traverse = 0
        self.traveled = 0

        self.render_multi()
```

Рис. 2.17 – Класс, який відповідає за секцію з відображенням результатів останніх кидків куль

Функція render_multi в класі PrevMulti відповідає за відображення тексту, що представляє множник на поверхні прямокутника результату (рис. 2.18). Це дозволяє гравцям легко побачити значення кожного результату в історії кидків. Функція реалізує простий процес рендерингу тексту та його розміщення на відповідній поверхні. Функція render_multi починається з виклику методу self.font.render, який створює текстову поверхню з заданим текстом. У даному випадку текст є множником результату (self.multi_amt) з доданим суфіксом "x", що вказує на множник. Після створення текстової поверхні, її положення визначається за допомогою методу get_rect. Цей метод створює прямокутник (text_rect), який визначає розміри та позицію текстової поверхні. Прямокутник центрується відносно прямокутника поверхні результату (self.prev_surf.get_rect().center). Останнім кроком є виклик методу blit, який копіює текстову поверхню на поверхню результату (self.prev_surf). Це забезпечує відображення тексту в центрі прямокутника, що відповідає за результат останнього кидка.

```
def render_multi(self):
    text_surface = self.font.render(f"{self.multi_amt}x", True, (0, 0, 0))
    text_rect = text_surface.get_rect(center=self.prev_surf.get_rect().center)
    self.prev_surf.blit(text_surface, text_rect)
```

Рис. 2.18 – Фукнція, яка відповідає за відображення тексту на секції

Функція update в класі PrevMulti відповідає за анімацію руху секції з результатами та її видалення, коли вона перевищує певну межу екрану (рис. 2.19). Це дозволяє підтримувати актуальність відображення результатів гри, зберігаючи лише останні результати.

Функція починається з перевірки, чи нижній край прямокутника результату (self.prev_rect.bottom) перевищує задану межу екрану (висоту екрану мінус подвійна висота прямокутника результату). Якщо це так, об'єкт видаляється з групи спрайтів за допомогою методу self.kill(). Це забезпечує видалення застарілих результатів і збереження лише останніх. Якщо межа не перевищена, функція продовжує анімацію руху. Анімація руху об'єкта реалізується через поступове збільшення позиції нижнього краю прямокутника результату (self.prev_rect.bottom). Для цього визначається загальна відстань, яку потрібно пройти (total_distance), і швидкість переміщення за секунду (distance_per_second). Щоб уникнути різких стрибків, швидкість переміщення за один кадр обчислюється як добуток швидкості переміщення за секунду та часу одного кадру (distance_per_frame = distance_per_second * delta_time). Однак, щоб забезпечити плавність анімації, швидкість переміщення за кадр коригується за допомогою подільника (divisor). Наприкінці функції результат блокується на екран за допомогою методу blit, який копіює поверхню результату (self.prev_surf) на основний екран відображення (self.display_surface).

```
def update(self):
    if self.prev_rect.bottom > (HEIGHT - (SCORE_RECT * 2)): # 864 at 1080
        self.kill()

    else:
        if self.traveled < self.y_traverse:
            total_distance = SCORE_RECT
            distance_per_second = 1800
            distance_per_frame = distance_per_second * delta_time # 28 at dt = .016
            divisor = int(SCORE_RECT / distance_per_frame)
            distance_per_frame = SCORE_RECT / divisor
            self.prev_rect.bottom += int(distance_per_frame)
            self.traveled += int(distance_per_frame)
        self.display_surface.blit(self.prev_surf, self.prev_rect)
```

Рис. 2.19 – Функція, яка відповідає за анімацію секції з результатами гри

Клас PrevMultiGroup, наслідуваний від pygame.sprite.Group, створений для управління групою спрайтів, що відображують останні результати гри (рис. 2.20). Цей клас відповідає за додавання, оновлення та анімацію цих спрайтів, забезпечуючи збереження останніх результатів та їх коректне відображення на екрані.

Метод update цього класу спершу викликає метод update базового класу для оновлення всіх спрайтів у групі. Потім додаються додаткові логічні блоки для управління кількістю спрайтів у групі та їх анімацією. Перший логічний блок забезпечує, щоб у групі залишалося максимум п'ять спрайтів. Якщо кількість спрайтів перевищує п'ять, найстаріший спрайт видаляється за допомогою методу self.remove(self.sprites().pop(0)). Наступні логічні блоки відповідають за анімацію спрайтів. Залежно від кількості спрайтів у групі, встановлюється різна відстань (y_traverse), на яку кожен спрайт повинен переміститися вниз. Це забезпечує, що спрайти розміщуються один під одним з рівномірним відступом. Наприклад, якщо в групі один спрайт, він переміщується на відстань SCORE_RECT. Якщо два спрайти, перший переміщується на відстань SCORE_RECT * 2, а другий на відстань SCORE_RECT. Аналогічним чином, для трьох, чотирьох і п'яти спрайтів встановлюються відповідні відстані для кожного спрайта.

```
class PrevMultiGroup(pygame.sprite.Group):
    def __init__(self):
        super().__init__()
        pass

    def update(self):
        super().update()

        # Maintain four previous multis at a maximum; animate
        if len(self) > 5:
            self.remove(self.sprites().pop(0))
        if len(self) == 1:
            self.sprites()[0].y_traverse = SCORE_RECT
        elif len(self) == 2:
            self.sprites()[0].y_traverse, self.sprites()[1].y_traverse = SCORE_RECT * 2, SCORE_RECT
        elif len(self) == 3:
            self.sprites()[0].y_traverse, self.sprites()[1].y_traverse, self.sprites()[2].y_traverse = SCORE_RECT * 3, SCORE_RECT * 2, SCORE_RECT
        elif len(self) == 4:
            self.sprites()[0].y_traverse, self.sprites()[1].y_traverse, self.sprites()[2].y_traverse, self.sprites()[3].y_traverse = SCORE_RECT * 4,
            SCORE_RECT * 3, SCORE_RECT * 2, SCORE_RECT
        elif len(self) == 5:
            self.sprites()[0].y_traverse, self.sprites()[1].y_traverse, self.sprites()[2].y_traverse, self.sprites()[3].y_traverse,
            self.sprites()[4].y_traverse = SCORE_RECT * 5, SCORE_RECT * 4, SCORE_RECT * 3, SCORE_RECT * 2, SCORE_RECT
```

Рис. 2.20 – Клас, створений для управління групою спрайтів, що відображують останні результати гри

Для покращення візуального вигляду секції, що відображає результати останніх ігор у грі, ми додамо нову функцію draw_prev_multi_mask у класі Board (рис. 2.21). Ця функція створить напівпрозору маску, яка виділяє зону, де відображаються останні результати, надаючи грі більш привабливий та професійний вигляд.

Функція draw_prev_multi_mask розпочинається зі створення поверхні multi_mask_surface з розмірами чверті ширини екрану по горизонталі і повної висоти екрану по вертикалі. Поверхня створюється з використанням формату pygame.SRCALPHA, що дозволяє підтримувати прозорість. Наступним кроком є заповнення поверхні кольором фону BG_COLOR. Це створює основу для нашої маски. Потім визначаються координати для розташування маски. Змінна right_side_of_board відповідає за позицію маски по горизонталі, яка розміщується праворуч від ігрового поля. Змінна right_side_pad використовується для невеликого відступу від правого краю. Далі, використовуючи функцію pygame.draw.rect, ми малюємо прямокутник на поверхні multi_mask_surface. Прямокутник має напівпрозорий фон (RGBA: 0, 0, 0, 0) і закруглені кути з радіусом 30 пікселів. Нарешті, функція blit використовується для відображення поверхні multi_mask_surface на основній поверхні дисплея self.display_surface у визначеній позиції. Додавання функції draw_prev_multi_mask значно покращує візуальний вигляд секції результатів останніх ігор у грі "Плінко". Напівпрозора маска створює більш чіткий та привабливий інтерфейс, виділяючи зону результатів та надаючи грі професійний вигляд. Це сприяє поліпшенню загального користувацького досвіду та робить гру більш естетично привабливою.

```
def draw_prev_multi_mask(self):
    multi_mask_surface = pygame.Surface((WIDTH / 4, HEIGHT), pygame.SRCALPHA)
    multi_mask_surface.fill(BG_COLOR)
    right_side_of_board = (WIDTH / 16) * 13
    right_side_pad = right_side_of_board / 130
    mask_y = (HEIGHT / 4) + ((HEIGHT / 4) / 9)
    pygame.draw.rect(multi_mask_surface, (0, 0, 0, 0), (right_side_pad, mask_y, SCORE_RECT, SCORE_RECT * 4), border_radius=30)
    self.display_surface.blit(multi_mask_surface, (right_side_of_board, 0))
```

Рис. 2.21 – Функція, яка покращує візуальний вигляд секції з результатами гри

Для додавання функціоналу, що дозволяє запускати кульку лише після натискання спеціальної кнопки, ми створимо кнопку "Play" (рис. 2.22). Вона буде зображена двома різними зображеннями для натиснутого та ненатиснутого стану. Ми використаємо pygame для завантаження, масштабування та відображення цих зображень.

Спочатку ми завантажуємо зображення для кнопки "Play" у двох станах: ненатиснутому (play_up) та натиснутому (play_down). Обидва зображення завантажуються з файлів play01.png та play02.png, відповідно. Для відстеження натискання кнопки ми створюємо логічну змінну self.pressing_play, яка буде зберігати стан кнопки. Далі, ми визначаємо оригінальні ширину та висоту зображення кнопки "Play". Масштабуємо обидва зображення кнопки до 50% їх початкового розміру для зручності розташування та більш привабливого вигляду. Для цього використовуємо функцію pygame.transform.scale. Після масштабування, ми визначаємо прямокутник (play_rect), який буде використовуватися для відстеження положення кнопки на екрані. Центр кнопки буде розташований на одній шостій ширини екрану по горизонталі та посередині висоти екрану.

```
# Play button
self.play_up = pygame.image.load("graphics/play01.png").convert_alpha()
self.play_down = pygame.image.load("graphics/play02.png").convert_alpha()
self.pressing_play = False
self.play_orig_width = self.play_up.get_width()
self.play_orig_height = self.play_up.get_height()

# Scale the play image by 50%
self.play_scaled_width = self.play_orig_width // 2
self.play_scaled_height = self.play_orig_height // 2
self.scaled_play_up = pygame.transform.scale(self.play_up, (self.play_scaled_width, self.play_scaled_height))
self.scaled_play_down = pygame.transform.scale(self.play_down, (self.play_scaled_width, self.play_scaled_height))
self.play_rect = self.scaled_play_up.get_rect(center=(WIDTH / 6, HEIGHT / 2))
```

Рис. 2.22 – Додавання кнопки «Play»

Щоб реалізувати функціонал кнопки "Play", ми додамо до функції update логіку, яка буде відповідати за зміну вигляду кнопки при натисканні (рис. 2.23). Коли гравець натискає кнопку, ми відображатимемо натиснуту версію кнопки, а коли кнопка не натиснута – її початковий вигляд. Додаємо перевірку стану натискання кнопки у функцію update. Якщо змінна self.pressing_play встановлена

в True, це означає, що кнопка натиснута, і ми відображаємо відповідне зображення. Якщо змінна self.pressing_play встановлена в False, відображаємо зображення ненатиснутої кнопки.

```
def update(self):
    self.draw_obstacles(self.obstacles_list)
    multi_group.draw(self.display_surface)
    multi_group.update()
    if len(list(prev_multi_group)) > 0:
        prev_multi_group.update()
    if len(list(animation_group)) > 0:
        animation_group.update()
    self.draw_prev_multi_mask()
    if self.pressing_play:
        self.display_surface.blit(self.scaled_play_down, (WIDTH // 16, HEIGHT // 3))
    else:
        self.display_surface.blit(self.scaled_play_up, (WIDTH // 16, HEIGHT // 3))
```

Рис. 2.23 – Функція update класу board, яка містить логіку, що відповідає за правильне відображення кнопки «Play»

Основний цикл while в класі Main відповідає за управління основним процесом гри (рис. 2.24). Він виконується безперервно, забезпечуючи безперервний рендеринг та обробку подій.

Перша частина циклу займається обробкою подій. Цей блок коду обробляє події, які виникають під час гри. Якщо гравець закриває вікно гри (подія pygame.QUIT), програма завершується. Якщо гравець натискає кнопку миші (подія pygame.MOUSEBUTTONDOWN), визначається позиція натискання. Якщо ця позиція співпадає з областю кнопки "Play", змінна self.board.pressing_play встановлюється в True. Це дозволяє нам відслідковувати, чи натиснута кнопка "Play". Коли гравець відпускає кнопку миші (подія pygame.MOUSEBUTTONUP), і натискання було на кнопку "Play", запускається нова кулька. Для цього створюється новий об'єкт Ball з випадковою позицією по осі X, щоб уникнути застрягання кульки на перешкодах. Звук натискання (click.play()) також відтворюється для додаткового зворотного зв'язку.

Далі в основному циклі відтворюється натискання кнопки "Play". Цей блок коду перевіряє стан self.board.pressing_play і відповідно відображає зображення

кнопки "Play". Якщо кнопка натиснута, відображається її натиснута версія; в іншому випадку – початкова версія.

Окрім цього, в основному циклі оновлюється екран. Тут оновлюється група кульок (self.ball_group.update()), обробляються фізичні симуляції в pymunk (self.space.step(self.delta_time)), виконується рендеринг дошки (self.board.draw()) і оновлюється екран (pygame.display.update()). Останній рядок забезпечує обмеження частоти кадрів до 60 кадрів в секунду і оновлює значення delta_time для підтримання стабільної анімації.

Основний цикл while в класі Main забезпечує постійний рендеринг гри та обробку подій. Він обробляє вихід з гри, натискання кнопки "Play" і створення нових кульок, а також оновлює і відображає всі елементи гри. Це забезпечує плавний ігровий процес і взаємодію з гравцем.

```
while True:
    # Handle quit operation
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            # Get the position of the mouse click
            mouse_pos = pygame.mouse.get_pos()

            # Check if the mouse click position collides with the image rectangle
            if self.board.play_rect.collidepoint(mouse_pos):
                self.board.pressing_play = True
            else:
                self.board.pressing_play = False
        # Spawn ball on left mouse button release
        elif event.type == pygame.MOUSEBUTTONUP and event.button == 1 and self.board.pressing_play:
            mouse_pos = pygame.mouse.get_pos()
            if self.board.play_rect.collidepoint(mouse_pos):
                random_x = WIDTH//2 + random.choice([random.randint(-20, -1), random.randint(1, 20)])
                click.play()
                self.ball = Ball((random_x, 20), self.space, self.board, self.delta_time)
                self.ball_group.add(self.ball)
                self.board.pressing_play = False
            else:
                self.board.pressing_play = False
```

Рис. 2.24 – Цикл, який оброблює та поєднує головні події гри

Програма вже повноцінно функціонує, проте є ще деякі деталі, які можуть значно покращити загальний досвід від гри. Одним із таких покращень буде додавання анімації перешкод. Коли кулька стикається з перешкодою, повинне з’являтись характерне світіння, яке візуально підкреслює цей процес. Це додасть

більшої динамічності та привабливості грі. Реалізуємо цей функціонал за допомогою спеціальної функції в класі Ball (рис. 2.25).

Цей блок коду відповідає за виявлення зіткнення кульки з перешкодами та створення анімації при цьому зіткненні. Використовуючи цикл for, перевіряється кожна перешкода в self.board.obstacle_sprites на зіткнення з кулькою за допомогою методу pygame.sprite.collide_rect. Якщо зіткнення виявлено, виконуються подальші дії для створення анімації. Визначається центр перешкоди, яка зіткнулася з кулькою, зберігаючи координати центру. Далі, перевіряється, чи вже існує анімація для цієї перешкоди в animation_group. Якщо така анімація існує, вона видаляється, щоб запобігти накопиченню анімацій на одній перешкоді. Після цього створюється нова анімація AnimatedObstacle з відповідними параметрами (координати центру перешкоди, радіус, колір і delta_time). Нова анімація додається до animation_group. Цей підхід забезпечує, що при кожному зіткненні кульки з перешкодою створюється анімація, яка підсвічує цей момент, роблячи гру більш інтерактивною та візуально привабливою.

```
for obstacle in self.board.obstacle_sprites:
    if pygame.sprite.collide_rect(self, obstacle):
        # Create animation and add to animation_group
        obstacle_centerx, obstacle_centery = obstacle.rect.centerx, obstacle.rect.centery
        obstacle_pos = (obstacle_centerx, obstacle_centery)

        for animating_obstacle in animation_group:
            if obstacle_pos == animating_obstacle.coords:
                animating_obstacle.kill()

        # Instantiate obstacle animation: params -> x, y, radius, color, delta_time
        obs_anim = AnimatedObstacle(obstacle_centerx, obstacle_centery, 16, (255, 255, 255), self.delta_time)
        animation_group.add(obs_anim)
```

Рис. 2.25 – Функція, яка відповідає за анімацію зіткнення кульки та перешкоди.

Клас Obstacle відповідає за створення та налаштування об'єктів перешкод у грі (рис. 2.26). Цей клас є підкласом pygame.sprite.Sprite, що дозволяє легко керувати перешкодами та інтегрувати їх у загальну структуру гри, використовуючи можливості спрайтів Pygame.

Конструктор __init__ приймає два параметри: x і y, які визначають початкові координати перешкоди на ігровому полі. Викликається конструктор батьківського класу pygame.sprite.Sprite за допомогою super().__init__(), що забезпечує коректну ініціалізацію базових властивостей спрайта. Встановлюється колір перешкоди, використовуючи заздалегідь визначену змінну, яка містить колір у форматі RGB. Задається радіус перешкоди за допомогою змінної, яка визначає розмір перешкоди. Встановлюються початкові координати перешкоди, передані в конструктор під час створення об'єкта перешкоди. Створюється поверхня для зображення перешкоди з розмірами, що відповідають діаметру кулі (подвійний радіус), і підтримується прозорість. Створюється прямокутник, що оточує поверхню перешкоди, і центрується за заданими координатами. Це дозволяє легко розташовувати та відслідковувати перешкоди на ігровому полі. Таким чином, клас Obstacle забезпечує створення, налаштування та управління об'єктами перешкод у грі, що робить їх інтеграцію з іншими елементами гри простою та ефективною.

```
class Obstacle(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.color = OBSTACLE_COLOR
        self.radius = OBSTACLE_RAD
        self.pos_x, self.pos_y = x, y
        self.image = pygame.Surface((BALL_RAD * 2, BALL_RAD * 2), pygame.SRCALPHA)
        self.rect = self.image.get_rect(center=(self.pos_x, self.pos_y))
```

Рис. 2.26 – Клас перешкоди у грі

Клас AnimatedObstacle створений для відображення анімації при зіткненні кульки з перешкодою (рис. 2.27). Він також є підкласом pygame.sprite.Sprite, що дозволяє легко використовувати його в контексті гри, інтегруючи з іншими спрайтами та системами Pygame.

Конструктор приймає кілька параметрів: x, y, radius, color та delta_time. Ці параметри визначають початкові координати, радіус, колір анімації та інтервал часу між кадрами анімації. Викликається конструктор батьківського класу

pygame.sprite.Sprite для налаштування базових властивостей спрайта. Отримується поточна поверхня дисплея за допомогою pygame.display.get_surface(). Це дозволяє відображати анімацію на головному екрані гри. Зберігаються початкові координати x та y, а також їх комбінація у вигляді кортежу coords. Це спрощує роботу з позицією анімації. Встановлюється радіус анімації, що визначає розмір анімованого об'єкта. Встановлюється колір анімації, переданий у конструкторі, що визначає, як буде виглядати анімація при зіткненні. Зберігається delta_time, який використовується для управління швидкістю анімації, роблячи її плавною та реалістичною. Створюється прямокутник, який оточує анімований об'єкт. Його розміри залежать від радіуса, і він центрується за заданими координатами x та y. Це дозволяє легко розташовувати та відслідковувати анімацію на ігровому полі. Клас AnimatedObstacle забезпечує анімацію при зіткненні кульки з перешкодою, додаючи візуальні ефекти, які покращують загальне сприйняття гри.

Клас AnimatedObstacle містить додаткову логіку для управління ефектами анімації при зіткненні, зокрема анімацією зникнення. Ця частина коду визначає, як швидко анімація буде зникати, роблячи її більш реалістичною та приємною для ока. Початкова прозорість (alpha) встановлюється на значення 125. Це значення визначає початковий рівень прозорості анімації, де 255 означає повну непрозорість, а 0 – повну прозорість. Встановлюється швидкість зникнення анімації в одиницях на секунду (fade_speed_second). Це визначає, як швидко анімація буде зникати за одну секунду. Обчислюється швидкість зникнення за один кадр (fade_speed_frame), використовуючи значення швидкості зникнення на секунду (fade_speed_second) і час між кадрами (delta_time). Це значення показує, на скільки одиниць буде змінюватися прозорість анімації за один кадр. Визначається дільник (divisor), який використовується для обчислення остаточної швидкості зникнення на кадр. Цей дільник обчислюється шляхом ділення швидкості зникнення на секунду на швидкість зникнення на кадр. Це дозволяє адаптувати швидкість зникнення відповідно до частоти кадрів гри. Остаточне значення швидкості зникнення на кадр обчислюється шляхом ділення

початкової прозорості (alpha) на дільник (divisor). Це значення визначає, на скільки одиниць буде змінюватися прозорість анімації при кожному оновленні кадру. Ця частина коду забезпечує плавне та поступове зникнення анімації, що додає додатковий рівень реалістичності та естетичної привабливості грі.

Для забезпечення плавної зміни розміру анімації перешкоди при зіткненні, в класі AnimatedObstacle додається логіка для зменшення радіуса анімації. Це робиться з метою створення ефекту поступового зникнення, де об'єкт не тільки стає більш прозорим, але й зменшується в розмірах. Встановлюється швидкість зміни радіуса (radius_dec_second) у одиницях на секунду. Це значення визначає, наскільки зменшиться радіус анімації за одну секунду. Обчислюється швидкість зміни радіуса за один кадр (radius_dec_frame), використовуючи значення швидкості зміни радіуса на секунду (radius_dec_second) і час між кадрами (delta_time). Це значення показує, наскільки зменшиться радіус анімації за один кадр. Визначається дільник (divisor_rad), який використовується для обчислення остаточної швидкості зміни радіуса на кадр. Цей дільник обчислюється шляхом ділення швидкості зміни радіуса на секунду на швидкість зміни радіуса за кадр. Це дозволяє адаптувати швидкість зміни радіуса відповідно до частоти кадрів гри. Остаточне значення швидкості зміни радіуса за кадр обчислюється шляхом ділення швидкості зміни радіуса на секунду (radius_dec_second) на дільник (divisor_rad). Це значення визначає, наскільки одиниць зменшиться радіус анімації при кожному оновленні кадру. Ця частина коду забезпечує плавне та поступове зменшення розміру анімації, що створює більш реалістичний та привабливий ефект при зіткненні кульки з перешкодою.

Функції fade, update та draw в класі AnimatedObstacle забезпечують анімацію при зіткненні кульки з перешкодою. Функція fade поступово зменшує прозорість та радіус анімації, а також видаляє її, коли вона стає невидимою. Функція update оновлює стан анімації, викликаючи fade та draw. Функція draw малює коло на екрані з поточним радіусом і прозорістю. Разом ці функції створюють ефект зникнення об'єкта при зіткненні.

```
class AnimatedObstacle(pygame.sprite.Sprite):
    def __init__(self, x, y, radius, color, delta_time):
        super().__init__()
        self.display_surface = pygame.display.get_surface()
        self.x, self.y = x, y
        self.coords = (self.x, self.y)
        self.radius = radius
        self.color = color
        self.delta_time = delta_time
        self.rect = pygame.Rect(x - radius, y - radius, radius * 2, radius * 2)

        # Calculate alpha value to decrement each frame
        self.alpha = 125
        self.fade_speed_second = 250
        self.fade_speed_frame = self.fade_speed_second * self.delta_time
        self.divisor = int(self.fade_speed_second / self.fade_speed_frame)
        self.fade_speed_frame = self.alpha / self.divisor

        # Calculate radius value to decrement each frame
        self.radius_dec_second = 32
        self.radius_dec_frame = self.radius_dec_second * self.delta_time
        self.divisor_rad = int(self.radius_dec_second / self.radius_dec_frame)
        self.radius_dec_frame = self.radius_dec_second / self.divisor_rad
```

Рис. 2.27 – Клас, створений для відображення анімації при зіткненні кульки з перешкодою

Додавання звукових ефектів завершує нашу розробку, створюючи повний звуковий досвід гри. Звук є важливою складовою будь-якого ігрового середовища, оскільки він допомагає поглибити іммерсію гравця та зробити гру більш захоплюючою. Відтворення звукових ефектів під час різних подій у грі, таких як зіткнення з перешкодою або досягнення певного результату, робить геймплей більш динамічним та емоційно зворушливим. Крім того, звукові ефекти можуть слугувати важливими сигналами для гравця, надаючи інформацію про стан гри та досягнення цілей.

Функція, яка має назву hit_sound, відповідає за відтворення звукових ефектів у відповідності до значень мультиплікатора, який представляє результат кульки у грі (рис. 2.28). Кожен мультиплікатор має свій унікальний звуковий ефект, який відтворюється під час зіткнення кульки з перешкодою. Наприклад, коли мультиплікатор має значення 0.2, відтворюється перший звуковий ефект, а для значення 2 - другий звук, і так далі. Це додає геймплею більшу динаміку та емоційний зв'язок з гравцем, оскільки звукові ефекти відображають результати

його гри. Такий підхід забезпечує зручну звукову обробку різних сценаріїв гри, що підвищує загальний рівень іммерсії та задоволення від гри.

```
def hit_sound(self):
    if str(self.multi_amt) == "0.2":
        sound01.play()
    elif str(self.multi_amt) == "2":
        sound02.play()
    elif str(self.multi_amt) == "4":
        sound03.play()
    elif str(self.multi_amt) == "9":
        sound04.play()
    elif str(self.multi_amt) == "26":
        sound05.play()
    elif str(self.multi_amt) == "130":
        sound06.play()
    elif str(self.multi_amt) == "1000":
        sound07.play()
```

Рис. 2.28 – Функція, яка відповідає за звукові ефекти гри

У розділі про програмну реалізацію гри «Плінко» було описано виконання значної кількості роботи, спрямованої на створення віртуального ігрового середовища, яке відтворює класичну аркадну гру. Застосування бібліотек Pygame і Pymunk дозволило точно відтворити фізичні властивості кульок та перешкод, що відіграють ключову роль у геймплеї «Плінко». Додавання звукових ефектів збільшило іммерсію гравців та зробило гру більш цікавою. Разом із графічним ігровим інтерфейсом, ці елементи сприяють створенню атмосфери аркадної розваги, що захоплює гравця з перших хвилин гри.

2.6 Організація тестування та налагодження програмного засобу

У розділі про організацію тестування та налагодження програмного засобу, розглянуті процедури, що спрямовані на перевірку якості та готовності гри до випуску. Цей етап розробки відіграє ключову роль у забезпеченні надійності та стабільності програми перед її публікацією. Проведено різні види тестування, а також перевірку функціональності та взаємодії між елементами гри:

1. Розроблену гру перевіряли на різних етапах розробки шляхом ручного тестування. Це дозволило виявляти потенційні проблеми та дефекти в ранній стадії розробки.

2. Проводили тестування окремих функціональних можливостей гри, включаючи рух кульок, взаємодію з перешкодами та мультиплікаторами, анімацію та звукові ефекти.

3. Важливою частиною було тестування взаємодії між об'єктами гри, такими як кульки, перешкоди та мультиплікатори, щоб переконатися у правильності їхньої взаємодії та відповідності геймплейних правил.

4. Гру тестували на різних операційних системах та пристроях, щоб забезпечити її сумісність та правильну роботу на різних пристроях.

Налагодження програмного засобу є важливим для досягнення оптимальної продуктивності та надійності програми перед її випуском. Налагодження включає в себе виправлення помилок, оптимізацію продуктивності та перевірку на відсутність витоків пам'яті. Цей етап розробки допомагає покращити якість програмного засобу та забезпечити задоволення користувачів.

Під час роботи з програмним засобом було виявлено різноманітні помилки та дефекти, які були ідентифіковані під час тестування. Після цього здійснювалась робота з їх усунення та виправлення в програмному коді.

Для забезпечення оптимальної продуктивності гри проводилась оптимізація швидкодії та зменшення споживання ресурсів. Це допомагало забезпечити плавну роботу програми на різних пристроях та підвищити загальний рівень задоволення користувачів.

Однією з важливих аспектів тестування було перевіряти програму на відсутність витоків пам'яті. Це дозволяло уникнути проблем, пов'язаних з продуктивністю та стабільністю гри, та забезпечити її надійну роботу.

Після завершення процесу тестування та налагодження програмного засобу була проведена фінальна перевірка готовності гри до випуску.

2.7 Рекомендації по використанню та впровадженню програмного засобу

Гра "Плінко" - це програмний продукт, який відкритий для користувачів будь-якого віку та будь-яких інших аспектів. Немає обмежень по тому, хто може користуватися цією грою. Проте, бажано використовувати десктопну версію програми та мати стабільне інтернет-з'єднання для оптимального досвіду гри.

Принцип гри полягає в наступному: користувач натискає кнопку, після чого кулька починає падати зверху вниз по дошці, де на шляху її руху розташовані перешкоди та мультиплікатори. Коли кулька досягає дна дошки, визначається, на який мультиплікатор вона попадає. Від цього мультиплікатора залежить виграш або програш гравця. Наприклад, якщо кулька потрапляє на мультиплікатор, який збільшує виграш у 2 рази, гравець отримує подвоєний виграш. А якщо кулька потрапляє на мультиплікатор, який вказує на програш, гравець не отримує жодного виграшу. Таким чином, виграш або програш гравця визначається випадковим чином, залежно від мультиплікатора, на який потрапляє кулька.

ВИСНОВКИ

В ході дослідження та написання даної дипломної роботи було досягнуто поставленої мети. Під час аналізу роботи популярних азартних додатків, було проведено глибоке дослідження їх функціональності та особливостей. На основі отриманих даних вдалося розробити аналоги цих додатків за допомогою мови програмування Python та спеціалізованих бібліотек. В результаті виконаної роботи, було отримано розширене розуміння процесів та механізмів роботи азартних ігор, що дозволяє зробити внесок у безпеку та відповідальність у галузі азартних розваг.

Дослідження азартних ігор допомагає розуміти їхню природу, функціонування та потенційні ризики, що в свою чергу дає змогу створювати більш безпечні та відповідальні ігрові середовища. Популярність азартних ігор в сучасному світі свідчить про важливість існування механізмів, які регулюють та контролюють їхнє використання, забезпечуючи безпеку та захист користувачів. Розвиток розуміння суспільства про принципи азартних ігор сприяє формуванню культури відповідального геймінгу, де гравці можуть насолоджуватися грою, маючи при цьому свідомість про її можливі наслідки.

Щодо технічної частини, розробка аналогу аркадної азартної гри "Плінко" на мові програмування Python з використанням бібліотек Pygame та Pymunk пройшла успішно. Під час розробки вдалося виконати всі поставлені задачі, що включали в себе створення основного геймплею, відтворення механіки гри, та використання фізичних двигунів для реалістичної симуляції руху об'єктів. Результатом цієї роботи є функціональний та коректно працюючий застосунок, який може бути використаний для навчання та розвитку навичок програмування.

Проте, є кілька аспектів, які можна покращити або додати у гру. Наприклад, можна розширити набір рівнів або впровадити нові елементи геймплею для збільшення різноманітності гри. Також можливе додавання анімаційних ефектів або покращення візуальної привабливості інтерфейсу, щоб зробити гру більш привабливою для гравців.

У цілому, успішна розробка аркадної гри на Python є важливим кроком у напрямку створення цікавих та привабливих аркадних ігор, а також демонструє потенціал мови програмування Python та засобів розробки ігор для створення високоякісних розважальних продуктів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Учасники проектів Вікімедіа. Role-playing game system – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Role-playing_game_system (дата звернення: 01.06.2024).

2. Ігрова механіка: які зараз існують типи в різних жанрах ігор. FoxmindEd. URL: https://foxminded.ua/igrova-mekhanika/ (дата звернення: 01.06.2024).

3. Учасники проектів Вікімедіа. Аркада (гра) – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Аркада_(гра) (дата звернення: 01.06.2024).

4. Вступ до історії геймдизайну. Частина 1: Аркадні ігри • VOKI Games. Voki Games. URL: https://vokigames.com/ua/vstup-do-istoriyi-gejmdyzajnu-chastyna-1-arkadni-igry/ (дата звернення: 01.06.2024).

5. Ігрові жанри. Онлайн-курси від компанії QATestLab | Головна сторінка. URL: https://training.qatestlab.com/blog/technical-articles/games-genres/ (дата звернення: 01.06.2024).

6. Учасники проектів Вікімедіа. Азартні ігри – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Азартні_ігри (дата звернення: 01.06.2024).

7. Rurskiy E. Різновиди азартних ігор онлайн в Україні. LinkedIn: Log In or Sign Up. URL: https://www.linkedin.com/pulse/різновиди-азартних-ігор-онлайн-в-україні-egor-rurskiy (дата звернення: 01.06.2024).

8. Особливості азартних ігор - в чому їх плюси? - Бізнес новини Слов'янська. 6262.com.ua - Сайт міста Слов'янська. URL: https://www.6262.com.ua/list/335678 (дата звернення: 01.06.2024).

9. Огляд на демо гру Плінко- Новини Кривого Рогу. Новини Кривого Рогу. URL: https://kryvyirih.dp.ua/demo-gra-plinko/ (дата звернення: 01.06.2024).

10. Що таке гра Plinko і як в неї грати?. BIN.ua. URL: https://bin.ua/partners/287850-shho-take-gra-plinko-i-yak-v-neyi-grati.html (дата звернення: 01.06.2024).

11. Plinko - найкраща азартна онлайн-гра | Без Купюр - Новини Кропивницького і Кіровоградщини. Без Купюр. URL:

https://www.kypur.net/plinko-najkrashha-azartna-onlajn-gra/ (дата звернення: 01.06.2024).

12. Stake. Stake.com. URL: https://stake.com/casino/games/plinko (дата звернення: 01.06.2024).

13. SPRIBE INNOVATIVE GAMES. SPRIBE INNOVATIVE GAMES. URL: https://www.spribe.co/games/plinko (дата звернення: 01.06.2024).

14. Plinko (Spribe) Інформація про гру. Online Slots Reviews. URL: https://slotcatalog.com/uk/slots/Plinko-Spribe (дата звернення: 01.06.2024).

15. BC.GAME: Crypto Games & Slot Games - Crypto Gambling. URL: https://bc.game/game/plinko (дата звернення: 01.06.2024).

16. Учасники проектів Вікімедіа. Методологія розробки програмного забезпечення – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Методологія_розробки_програмного_забезпечення (дата звернення: 01.06.2024).

17. 12 кращих методологій розробки програмного забезпечення з перевагами та недоліками - SMART business. SMART business. URL: https://www.smart-it.com/uk/2021/08/12-best-software-development-methodologies-with-pros-and-cons/ (дата звернення: 01.06.2024).

18. Підходи до розробки програмного забезпечення: 4 основні. FoxmindEd. URL: https://foxminded.ua/pidkhody-do-rozrobky-prohramnoho-zabezpechennia/ (дата звернення: 01.06.2024).

19. Учасники проектів Вікімедіа. Python – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 01.06.2024).

20. Python. Python documentation. URL: https://docs.python.org/uk/3/tutorial/index.html (date of access: 01.06.2024).

21. Academy O. Розробка ігор на Python для дітей | Optima Academy. Академія онлайн | Optima Academy. URL: https://optima.study/rozrobka-igor-na-python (дата звернення: 01.06.2024).

22. Курс "Програмування ігор на Python" URL: https://coddyschool.com/ua/courses/programmirovanie-igr-na-python/ (дата звернення: 01.06.2024).

23. Лешик П. Бібліотека Pygame / Частина 1. Введення. Хабр. URL: https://habr.com/ru/articles/588605/ (дата звернення: 01.06.2024).

24. Pygame Front Page – pygame v2.6.0 documentation. pygame news. URL: https://www.pygame.org/docs/ (дата звернення: 01.06.2024).

25. PyGame – Python documentation. Python Course. URL: https://python-course.readthedocs.io/projects/elementary/en/latest/lessons/18-pygame.html (дата звернення: 01.06.2024).

26. Pymunk – Pymunk documentation. Pymunk – Pymunk documentation. URL: https://www.pymunk.org/en/latest/ (дата звернення: 01.06.2024).

27. Микита А. Фізика в Python за допомогою Pymunk. Хабр. URL: https://habr.com/ru/articles/593547/ (дата звернення: 01.06.2024).

28. Pymunk. Pygame news. URL: https://www.pygame.org/project-pymunk-780-.html (date of access: 01.06.2024).

29. Жаріков Д. Мова програмування Python: легкий старт в складному КОДІНГ - блог академії Wezom. URL: https://wezom.academy/ua/yazyk-programmirovaniya-python-legkij-start-v-slozhnom-kodinge/ (дата звернення: 01.06.2024).

30. Program Arcade Games With Python And Pygame. URL: http://programarcadegames.com/ (дата звернення: 01.06.2024).

ДОДАТКИ
Додаток А
Технічне завдання на розробку програмного засобу «Плінко»

Аркадна азартна гра "Плінко" є популярною розвагою, в якій гравці випускають кульку з верхньої частини вертикальної дошки, на якій розташовані перешкоди. Кулька, падаючи вниз, відскакує від цих перешкод, змінюючи свою траєкторію, і врешті-решт потрапляє в один із слотів, розташованих внизу. Кожен з цих слотів має свою власну вартість. Ідея гри полягає в тому, щоб передбачити, у який слот потрапить кулька, що додає елемент випадковості та азарту.

Мета створення аналогу цієї гри на мові програмування Python полягає в дослідженні механізмів роботи азартних ігор, а також у наданні навчальних матеріалів для тих, хто хоче покращити свої навички програмування. Такий підхід дозволяє зрозуміти принципи розробки ігор, алгоритми симуляції фізичних явищ і забезпечення взаємодії між об'єктами в ігровому середовищі.

Особливості гри "Плінко" включають її простоту та захопливість, що робить її привабливою для широкої аудиторії. Відтворення фізики падіння кульки та випадковості її руху надає грі елемент непередбачуваності, який тримає гравців у напрузі до останнього моменту. Розробка цифрового аналогу гри "Плінко" на Python дозволяє застосувати сучасні інструменти програмування для створення реалістичного та захопливого ігрового досвіду, що водночас служить навчальним ресурсом для розробників-початківців та досвідчених програмістів.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підстави для розробки цього програмного засобу полягають у необхідності виконання вимог бакалаврської роботи, що включає в себе практичне застосування знань та навичок, отриманих під час навчання.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення аналогу аркадної гри "Плінко" на мові програмування Python з використанням бібліотек Pygame та Pymunk. Проект спрямований на практичне застосування теоретичних знань, вдосконалення навичок програмування та розробки ігор. Він також слугує для вивчення ігрової механіки та фізики в цифрових середовищах, демонструючи здатність розробника до вирішення технічних задач і оптимізації продуктивності.

ВИМОГИ ДО ПРОГРАМИ ЧИ ПРОГРАМНОГО ПРОДУКТУ

Програмний продукт "Плінко" повинен відповідати наступним вимогам:

Функціональні вимоги:

1. Реалізація основної механіки гри, яка включає падіння кульки через серію відбивачів.

2. Відображення результатів гри в реальному часі.

3. Підтримка звукових ефектів при зіткненні кульки з відбивачами.

4. Можливість запуску гри за допомогою кнопки.

5. Відображення останніх результатів гри.

6. Нефункціональні вимоги:

7. Продукт має бути написаний мовою програмування Python з використанням бібліотек Pygame та Pymunk.

8. Програма повинна працювати стабільно і без помилок.

9. Інтерфейс користувача має бути інтуїтивно зрозумілим і зручним.

10. Висока продуктивність та плавність роботи на різних платформах (Windows, macOS, Linux).

11. Використання оптимізованих алгоритмів для зменшення споживання ресурсів.

Естетичні вимоги:

1. Графіка гри повинна бути яскравою та привабливою.
2. Дизайн інтерфейсу має бути сучасним і відповідати стилю аркадних ігор.
3. Звукові ефекти повинні бути чіткими і відповідати діям в грі.

ТЕХНІКО-ЕКОНОМІЧНІ ПОКАЗНИКИ

Розробка програми "Плінко" не потребує фінансових витрат, оскільки використовується безкоштовне програмне забезпечення та бібліотеки Python, Pygame і Pymunk. Основні ресурси витрачаються на час розробки, тестування та налагодження програми. Економічна ефективність полягає в тому, що проект дозволяє отримати цінний досвід в програмуванні та розробці ігор, що може бути використано для подальших комерційних проектів. Використання відкритих інструментів також знижує витрати на ліцензування та забезпечує широкі можливості для модифікації та покращення програми в майбутньому.

СТАДІЇ І ЕТАПИ РОЗРОБКИ

Процес розробки програмного засобу "Плінко" включає декілька ключових стадій і етапів:

Планування та аналіз вимог - визначаються цілі та завдання проекту, здійснюється збір і аналіз вимог до програми. Здійснюється вивчення існуючих аналогів гри та визначення функціональних і нефункціональних вимог.

Проектування - розробляється архітектура програми, визначаються основні компоненти та їх взаємодія. Створюються схеми, алгоритми і структури даних, які будуть використовуватися у грі.

Розробка - відбувається написання коду згідно з проектною документацією. Використовуються мови програмування Python і бібліотеки Pygame та Pymunk для реалізації ігрової механіки, графіки та фізики.

Тестування - проводиться тестування програмного забезпечення для виявлення та виправлення помилок. Здійснюється ручне тестування, функціональне тестування та перевірка взаємодії між компонентами гри.

Налагодження - здійснюється виправлення виявлених помилок, оптимізація продуктивності програми та забезпечення стабільності її роботи. Перевіряється відсутність витоків пам'яті та інших потенційних проблем.

Документування - створюється технічна документація, яка описує архітектуру програми, використані алгоритми, структури даних та інтерфейси. Також готується користувацька документація.

Кожен з цих етапів є критично важливим для успішної реалізації проекту та забезпечення високої якості кінцевого продукту.

ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Приймання результатів роботи програми здійснюється в процесі здачі бакалаврської роботи.

Додаток Б
Інструкція користувача

У цьому розділі надано докладні інструкції щодо користування програмним засобом "Плінко". Ці вказівки допоможуть вам ознайомитися з основними функціями та можливостями гри, а також надати інформацію про використання різних функцій та налаштувань програми. Дотримуючись цих інструкцій, ви зможете насолодитися грою та використовувати всі її можливості безпроблемно.

1. Загальні відомості

Програмний продукт "Плінко" розроблено мовою програмування Python з використанням бібліотек Pygame та Pymunk. Ці бібліотеки дозволили створити гру з аркадними елементами та фізичними ефектами, що надають грі реалістичності та динамічності.

2. Функціональне призначення

Програмний продукт "Плінко" розроблено з метою дослідження механізмів роботи аркадних азартних ігор. У грі гравець спостерігає за рухом кульки у віртуальному середовищі. Це дозволяє користувачам отримати унікальний ігральний досвід, а також допомагає розуміти принципи роботи азартних ігор. Розробка такого програмного засобу важлива для того, щоб гравці краще розуміли механіки гри, її потенційні ризики та переваги, а також зменшити ризик негативних наслідків від її використання.

3. Умови застосування програми

Програмний продукт "Плінко" призначено для використання на персональних комп'ютерах з операційною системою Windows, macOS або Linux.

Для коректної роботи програми потрібне наявність Python версії 3.6 або вище, а також встановлені бібліотеки Pygame та Pymunk. Гравцеві необхідно мати стабільне Інтернет-з'єднання для завантаження програми та отримання оновлень.

4. Опис роботи програми

Головною метою цієї програми є дослідження механізмів роботи аркадних азартних ігор та навчання користувачів розуміти їхні особливості та можливі ризики.

Натисканням на кнопку "Play" посередині верхнього ряду перешкод ініціює появу кульки, яка починає падати, відбиваючись від перешкод та потрапляючи в один із слотів, який визначає виграш або програш гравця. Цей процес супроводжується візуальними та звуковими ефектами, що робить гру більш захоплюючою. Після завершення кожного раунду результати зберігаються в спеціальній секції результатів, де гравець може оглянути свою історію ігор.

АНОТАЦІЯ

Малащук В.А. – **Дослідження та розробка мережевих додатків у сфері азартних ігор**.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2024р.

Робота присвячена дослідженню та розробці мережевих додатків у сфері азартних ігор. Тема є актуальною, оскільки індустрія азартних ігор швидко зростає і потребує сучасних технологічних рішень для залучення та утримання користувачів. Продукт було розроблено за допомогою Python, Pygame та Pymunk. Застосунок забезпечує функцію кидка кульки після натискання відповідної кнопки, відтворення геймплею популярної гри, аналіз результатів та їх відображення. Особлива увага приділяється інтерфейсу і простоті використання. Робота охоплює всі етапи розробки застосунку від проєктування до програмної реалізації та тестування.

Ключові слова: аркадна гра, випадковість, розробка аналогів, Python, Pygame, Pymunk, Agile, дослідження механіки ігор, розробка.