Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: Розроблення гри "Breakneck Races" засобами Java

Виконав: студент 2 курсу групи ІСТС-21
спеціальності
126 "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)

Ільницький Віталій Ігорович
(прізвище та ініціали)

Керівник Борецька І.Б.
(прізвище та ініціали)

Рецензент Яцишин С. І.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСКМ
_____Сторожук О.Л.
"06" 02 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ільницькому Віталію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення гри "Breakneck Races" засобами Java
керівник роботи *Борецька Ірина Богданівна, кандидат технічних наук, доцент,*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "06" лютого 2024 року №С-87
2. Термін подання студентом роботи 10 червня 2024 р.
3. Вихідні дані до роботи Розробити розважально-розвиваючу комп'ютерну гру "Breakneck races" засобами Java. Гра має бути створена у 2D форматі у стилі гонок. Програма має містити наступні елементи: головне меню та вікно відображення процесу гри з пояснюючими елементами. Гра повинна розвивати розум, реакцію та логіку. Програма має бути створена у середовищі програмування мови Java.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
Розділ 1. Стан проблемної області
1.1. Актуальність поставленого завдання.
1.2. Проблематика області 2D ігор.
1.3. Розвиток жанру гонок.
Розділ 2. Інформаційне та математичне забезпечення
2.1. Чинники якості програмного продукту
2.2. Вхідні та вихідні дані програми
2.3. Математичне забезпечення механіки гри
Розділ 3. Програмне та технічне забезпечення
3.1. Вибір засобів розроблення
3.2. Розроблення структури програми
3.3. Розроблення макетів для елементів програми
3.4. Опис роботи гри
3.5. Тестування створеної гри
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Дата видачі завдання 07 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Збір та опрацювання матеріалу за темою дипломної роботи	07.02.2024-12.03.2024	Виконано
2	Проєктування програми	12.03.2024-15.04.2024	Виконано
3	Розробка програми	15.04.2024-01.05.2024	Виконано
4	Тестування програми	01.05.2024-08.05.2024	Виконано
5	Внесення правок у програму	08.05.2024-12.05.2024	Виконано
6	Розробка першого розділу пояснювальної записки	12.05.2024-19.05.2024	Виконано
7	Розробка другого розділу пояснювальної записки	19.05.2024-25.05.2024	Виконано
8	Розробка третього розділу пояснювальної записки	25.05.2024-31.05.2024	Виконано
9	Оформлення пояснювальної записки	31.05.2024-10.06.2024	Виконано

Студент _____ **Ільницький В.І**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Борецька І.Б.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 37 сторінок пояснювальної записки, 35 рисунків, 2 таблиці, 1 додаток, 18 джерел.

Розроблено розважально-розвиваючу комп’ютерну гру "Breakneck races" засобами Java. Гра створена у 2D форматі у стилі гонок. Реалізовано можливість виходу, як з програми так і з ігрового вікна. Інтерфейс гри чіткий та зрозумілий. Створена зручна навігація застосунку. Гра розрахована для різної вікової категорії.

Ключові слова: *Java, інтерфейс, 2D, 3D, image, swing, wav, awt, listeners, player, enemy*.

ANNOTATION

The thesis contains 37 pages of explanatory notes, 35 figures, 2 tables, 1 appendix, and 18 references.

An entertaining and educational computer game "Breakneck races" has been developed using Java. The game is created in a 2D format in the racing style. The game implements the ability to exit both the program and the game window. The game interface is clear and understandable. A convenient application navigation has been created. The game is designed for various age groups.

Keywords: *Java, interface, 2D, 3D, image, swing, wav, awt, listeners, player, enemy.*

ТЕХНЧНЕ ЗАВДАННЯ

Розробити розважально-розвиваючу комп’ютерну гру. Гра має бути створена у 2D форматі у стилі гонок. Програма має містити наступні елементи:

— головне меню українською мовою;

— вікно відображення процесу гри з пояснюючими елементами;

— користувач має переміщуватись по головному меню за допомогою миші, а сам процес гри має відбуватись за допомогою клавіатури;

— здійснити процес виходу як з програми так і з ігрового рушія.

Програма має бути створена у середовищі програмування мови Java. Технічні вимоги до роботи:

· зручна навігація по програмі;

· 2D-дизайн.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

IT	інформаційні технології
ОС	операційна система
SDK	Software Development Kit (комплект розробки програмного забезпечення)
JDK	Java Development Kit — безкоштовний розповсюджуваний Oracle комплект розробника застосунків на мові Java
ПЗ	програмне забезпечення
2D	двовимірна графіка
3D	тривимірна графіка
Swing	інструментарій для створення графічного інтерфейсу користувача мовою програмування Java
WAV	формат аудіофайлу
AWT	пакет класів мови програмування Java, що слугує для створення графічного інтерфейсу користувача
Гравець	зображення персонажа, за якого грає користувач
Ворог	зображення другорядного об’єкта в грі
Дорога	зображення на задньому фоні, з дорогою, яке постійно переміщується, створюючи ефект руху

ВСТУП

У сучасному швидкоплинному світі люди постійно перебувають у пошуку способів розслабитися та розважитися після напруженого робочого дня чи навчання. Одним із найпопулярніших видів дозвілля, особливо серед підлітків та молоді, є комп'ютерні ігри. Багатогранність жанрів та доступність цифрових розваг на персональних комп'ютерах, смартфонах та планшетах зробили їх невід'ємною частиною повсякденного життя у XXI столітті Вибір гри зазвичай зумовлений особистими вподобаннями та інтересами користувача. Наприклад, шанувальники спортивних змагань, імовірно, надаватимуть перевагу симуляторам спортивних дисциплін чи менеджерам, водночас любителі пригод можуть обрати шутери чи ігри в жанрі survival horror. Однак існують певні категорії ігор, які є універсальними та здатні зацікавити широке коло гравців різного віку та уподобань. До таких належать, зокрема, аркадні ігри, серед яких вагоме місце посідають гоночні симулятори.

Об’єктом дослідження є процес створення розважально-розвиваючої комп’ютерної гри у жанрі гонок засобами Java. Основною метою є розроблення застосунку, який не тільки буде корисним для відпочинку, а і для розвитку, а саме який допоможе розвивати моторику рук, швидкість прийняття рішень та логічне мислення.

Предмет дослідження – гра «Breakneck races», яка буде розроблена за допомогою мови програмування Java, з використанням бібліотек Swing та AWT. У рамках дослідження буде розглянуто ключові аспекти розроблення, такі як дизайн гри, робота з Java AWT, Swing та іншими бібліотеками.

Метою даної роботи є розробка розважально-розвиваючої комп’ютерної гри. Гра створена у 2D форматі, виконаної в жанрі аркадних перегонів.У програмі є інтуїтивний інтерфейс, який забезпечує зручність користування, надаючи гравцям можливість розпочинати нову гру, робити паузу чи перезавантажувати програму простим натисканням відповідних кнопок.

Дана програма призначена для розважальних та розвиваючих цілей, тому підійде для будь-якої вікової групи. Далі, для зручності, ми будемо використовувати назву для гри – «Breakneck races».

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Актуальність поставленого завдання

У сучасному світі ігрові інновації впроваджуються з неймовірною швидкістю. Варто лише згадати ті часи, коли Half-Life був справжньою вершиною графіки та ігрового процесу, а зараз про нього тільки згадують. Інтерактивне середовище, де можна знімати та брати предмети до рук, справді вражали. Сьогодні можливості ігрових технологій йдуть набагато далі, що ми не могли передбачити у наших дитячих фантазіях. Проте попит на «старі» ігри нікуди не зник. Попри те, що зараз набирають популярність VR технології чи звичайні 3D ігри, у світі залишається багато прихильників 2D чи навіть консольних ігор.

Ігрова індустрія сьогодні є найбільшою індустрією розваг у світі, налічуючи близько 3 мільярдів геймерів у всьому світі. Очікується, що в 2023 році ігрова індустрія отримає прибуток у розмірі близько 175,8 млрд доларів США, що в річному обчисленні скоротиться на -1,1% після сильного зростання в 2022 році.

Обсяг світового ринку відеоігор оцінювався в 151,06 мільярдів доларів США в 2019 році і, як очікується, буде рости із середньорічним темпом зростання (CAGR) в 12,9% з 2020 по 2027 рік (рис.1.1). Очікується, що технологічне поширення та інновації як в апаратному, так і в програмному забезпеченні будуть ключовими факторами росту. Зростаюче поширення інтернет-послуг в поєднанні з легкістю доступу до ігор в Інтернеті по всьому світу також підтримає оптимістичні перспективи зростання ринку в майбутні роки.

Розробники ігор також постійно імпровізують і перевершують технологічні обмеження щодо рендерингу графіки в реальному часі в індустрії відеоігор, що, як очікується, буде сприяти її зростанню. [15]

Рис.1.1. Динаміка розвитку ігрової індустрії з 2016 по 2027 роки.

1.2. Проблематика області 2D ігор

Перехід від фізичних до онлайн-ігор стає все більш очевидним, вимагаючи учасників галузі зосередитися на сумісності обладнання та ефективності. Безкоштовні Free2Play (F2P) ігри та ігри з великою кількістю гравців Massively Multiplayer Online (MMO) здобувають популярність, що спричиняє зростання доходів та передбачається, що ця тенденція протримається надалі. Підвищений рівень доходу також підтримує збільшення витрат на ігрові продукти, а удосконалені ігрові консолі, обладнані такими функціями, як запис і спільне використання, стають популярними серед споживачів. Але для реалізації 2D ігор не потрібні такі ресурси, як для розробки 3D ігор. Також, варто зазначити, що більшість 2D ігор є одиночними, тому жанр розробки таких ігор є відносно обмеженим.

В епоху швидкісного Інтернету більшість ігор так чи інакше, орієнтовані на онлайн. І хоча існують безліч різновидів багатокористувацьких ігор, 2D ігри залишаються вірними своєму корінню, не вимагаючи постійного підключення до Інтернету, що було характерно для перших ігор на ПК.

Тенденція 2D ігор в соціальних мережах матиме позитивний вплив на зростання ринку. Наприклад, значний відсоток світового населення використовує для ігор такі сайти соціальних мереж, як Facebook і Reddit. Очікується, що доступність різних жанрів ігор, таких як бойовик, рольові ігри, симулятори та стратегії, приверне більше клієнтів. Зростаюча популярність кіберспортивних турнірів і збільшення числа професійних гравців приведе до збільшення продажів відеоігор і аксесуарів, а також ігрового обладнання та програмного забезпечення. З власного досвіду, хотів би зазначити, що складні ігри недосконалі у соцмережах. А саме, проблема у рушіях, на яких написані ці ігри, вони не адаптовані для таких ресурсів. Але через простоту реалізації 2D ігор на різних платформах і їх розміщення на «хмарах» вони перевершують у продуктивності 3D ігри.

1.3. Розвиток жанру гонок

Першою визначною грою у стилі гонок, на мою думку, стала гра від Atari-Space Race (Космічна гонка). Загалом гонки розвивалися протягом десятиліть. Але у 1973 році народилася перша гоночна гра: Atari Space Race (рис.1.2), де два гравці взяли під

контроль космічні кораблі, намагаючись першими перетнути екран. Варто сказати, що це була гра на консольному апараті (рис.1.3.) У тому ж році розробник Taito випустив подібну гру під назвою Astro Race . Їхній вихід на цей ринок виявився важливим у наступному році. Швидкісні гонки, випущені у 1974 році, перенесли геймплей космічної гонки на землю. У цій грі вперше використана графіка прокручування, представлений вид зверху іподрому, а кермо (рис.1.3.) використовувалося для керування спрайтом автомобіля. Це був гігантський стрибок уперед від регуляторів гучності в попередніх іграх. Протягом решти років десятиліття вийшло багато гоночних ігор, в яких були представлені не тільки автомобілі, а й мотоцикли.

Рис.1.2. Процес гри.

Рис.1.3. Вигляд апарату, на якому проходила гра.

У 1982 році була випущена найвпливовіша гоночна гра всіх часів: Pole Position, гра для одного гравця з гоночними автомобілями Формули 1, де гравець грав проти семи інших керованих комп’ютером автомобілів на японській трасі Fuji Speedway.

У 1987 році Namco випустила наступника Pole Position: Final Lap. Це була

перша гра, в якій до восьми гравців-людей грали разом, об’єднавши чотири сидячі кабінети для двох гравців. Гравці могли вибирати з вибору автомобілів Формули-1 і трас. У режимі одиночної гри оцінка гравців ґрунтувалася на тому, як далеко вони проїхали або поки не закінчився час або не були завершені чотири кола. Згідно інтерфейсу гри (рис.1.4.) можна сказати, що це вже була непогана гра на той час. До речі, схожий інтерфейс зберігся надовго у 2D іграх, тому більшість людей, які запам’ятали 2D дизайн, пов’язують його саме з таким.

Рис.1.4. Гра «Pole Position: Final Lap».

Засновником сучасних гоночних, проте 3D ігор, стала серія відеоігор «The Need for Speed». Перша серія гри з’явилась у 1994, а остання – з'явилась у 2022 році та мала назву «Need for Speed Unbound» (рис.1.5.). [7]

На мою думку, це найпопулярніша серія гонок за всі часи. Те, як вона змінювалась роками і яку велику аудиторію зібрала навколо себе просто вражає.

Рис.1.5. Гра «Need for Speed Unbound».

Що ж можна сказати про сучасні 2D ігри? Звісно, як було сказано раніше, сучасні ігри витісняють «старі». Але всеодно залишаються ігри, які мають свою популярність в деяких колах.

Наприклад, у 2021 році ігрову індустрію сколихнула гра «Vampire Survivors» (рис.1.6.). Варто сказати, що гру написав розробник-одинак за £1100 (приблизно 1500 доларів). А у відеомагазині Steam середня вартість гри близько 2 доларів, не складно порахувати його заробіток, якщо на межі популярності середній онлайн у грі був близько 50 тисяч людей. Це тільки онлайн, а скільки загалом людей придбало гру і уявити складно. Також важливо те, що гра 2021 року стала відомою у середині 2022 року, охопивши не тільки підлітків, але і старших людей. Цей приклад показує те, що навіть найпростіша гра може дати дохід більший ніж продумана високоякісна гра. [11]

Рис.1.6. Vampire Survivors.

Самі розробники пишуть про цю гру так: «Vampire Survivors — це гра на виживання у часі з мінімалістичним ігровим процесом та елементами «roguelike». Немає де сховатися, все, що ви можете зробити, це спробувати пережити прокляту ніч і отримати якомога більше золота для наступного вцілілого, перш ніж Смерть неминуче покладе край вашій боротьбі». [10] Як зрозуміло, це гра, де потрібно виживати, та через те, що, щоб пройти хоча б один рівень потрібно витратити близько тридцяти хвилин, багатьох користувачів ця гра захопила надовго.

В результаті можна сказати, що перспективи на 2D ігри залишаються актуальними і понині. Головне щоб в результаті гра була добре продуманою та захопливою, а також, щоб не була дуже напруженою у моральному плані. Важливо зазначити, що не слід забувати про вартість гри, її підтримку у майбутньому, зворотній зв’язок із гравцями та рекламу, якщо потрібно.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Чинники якості програмного продукту

Перед початком розробки потрібно визначити, яким чинники має відповідати програма. На якість «Breakneck races» впливають наступні чинники, що подані нижче.

Продуктивність. Одним з найбільш цінних аспектів програми є швидка обробка даних. Чим менше часу програма витрачає на виконання завдань, тим краще. Через те, що «Breakneck races» 2D гра вона не витрачає багато часу на виконання завдань.

Сортування даних. Можна використовувати метрики, щоб зменшити непорозуміння та неоднозначність у складних проєктах. Про це згадаємо пізніше, коли будемо говорити про якість коду.

Надійність. Аналізуючи кількість дефектів, можна побачити, наскільки добре буде працювати програмне забезпечення і як довго система буде працювати без збоїв. «Breakneck races» має достатню надійність, оскільки не прив’язаний до інтернету та немає даних, які були би корисними для зловмисника.

Ремонтопридатність. Чи важко підтримувати програмне забезпечення? На це питання можна відповісти наступним чином. Загалом ні, проте, через те, що «Breakneck races» не прив’язаний до інтернету, для оновлення постійно потрібно буде скачувати оновлену програму чи інсталятор для оновлення.

Перевірка. Наскільки добре продукт підтримує тестування? У теперішній час з тестування ігор для розробників все дуже просто. Потрібно випустити бета-версію програми, а після тривалого використання користувачів виправити потрібні недоліки та випустити головну версію програми.

Портативність. Він показує, чи можна використовувати програмне забезпечення в різних середовищах. Нажаль з цим чинником у «Breakneck races» не все добре, оскільки він розроблений на Java та подальші дії можуть виконуватись тільки у цьому середовищі.

Проте, наприкінці потрібно сказати, що найважливіше для будь-якого ПЗ це його код. Далі наведено найважливіші показники коду.

Час виконання. Це час, який інженери витрачають на створення ідей, проєктування, розробку та завершення проєкту програмного забезпечення. Тобто чим швидше вийде проєкт, тим швидше ним зацікавляться клієнти.

Час циклу. Може бути важко зрозуміти різницю між цими двома визначеннями, але вони не збігаються. Період циклу починається з розробки програми і закінчується після її завершення, а час виконання починається з отримання замовлення та закінчується його доставкою. Тобто, фактично час циклу закінчиться або першою покупкою гри, або останньою проданою копією.

Швидкість. Цей чинник оцінює час, який знадобиться програмістам для розробки продукту. Це допомагає зрозуміти, скільки часу потрібно команді на кожен етап. Таким чином, ви можете скласти план майбутніх продуктів відповідно до вже наявних аналізів. [18]

2.2. Вхідні та вихідні дані програми

Код гри складається з 7 Java-класів: Back, Enemy, Listeners, Main, Menue, Playergame, Road. Усі класи мають в собі методи, змінні, а деякі навіть і підкласи. Типи даних варіюються від типових, як от double, integer, string, та до об’єктів класу (Image, Rectangle та інші).

Клас “Back”. Вхідні дані: img1, img2. Вихідні дані: об’єкт типу Back, img1, якщо вікно Гри; img2, якщо вікно Меню. Цей клас був створений для того, що перевірити, який стан зараз має програма, якщо це стан гри, тоді він віддає img1, інакше img2.

Клас “Enemy”. Вхідні дані: img- малюнок ворога, x,y,v – координати та швидкість, об’єкт типу Road. Вихідні дані: метод move, конструктор класу, прямокутник з шириною 180 та висотою 65 пікселі. У цьому класі описується структура ворога, а реалізація буде відбуватись у іншому місці. У ньому передається інформація конструктору про ворога. А саме у конструкторі є інформація про координати ворога, швидкість та рух, тобто часткова реалізація швидкості ворога.

Клас “Listeners”. Вхідні дані: перевірка натискання на певну клавішу, перевірка на натискання кнопок у Меню. Вихідні дані: вихід у Меню при натисканні на клавішу Esc, перехід у вікно Грати, якщо натиснуто на кнопку Грати. Вихід з програми, якщо натиснуто кнопку Вихід.

Клас “Menue”. Вхідні дані: button1,button2 типу ButtMenue, об’єкт типу Road. Вихідні дані: Об’єкт типу Menue, клас ButtMenue, метод draw, moveButt. Цей клас відповідає за відмальовування елементів меню гри. Він має підклас та метод для відмальовування саме кнопок для меню. Також, як згадувалось раніше, саме тут відмальовуються надписи для кнопок. Оскільки ми малюємо кнопки не як елемент Java- JButton, а просто як зображення, ми додатково перевіряємо чи курсор миші перетинає наше зображення для його зміни. Якщо так – тоді зображення змінюється на інше, та коли користувач забирає курсор з цих координат, знову повертається перше зображення.

Клас “Playergame”. Вхідні дані: img_c, img_l, img_r, x, y, v, s, dv, dy. Вихідні дані: об’єкт типу Playergame, img_c- зображення коли гравець їде по центру, img_l- при повороті на ліво, img_r- направо, dv,dy, методи keyReleased і keyPressed, який має в собі попередні дані, перевірка натискання на певні клавіші. У цьому класі відбуваються всі зміні, щодо гравця. А саме тут задається початкова та максимальна швидкість, прискорення та пройдений шлях. Також додатково тут перевіряється вихід за кордони дороги. Крім цього тут відбуваються деякі дії з дорогою. А саме це те, що одне зображення змінює інше і здається ніби гравець їде по дорозі, хоча насправді він стоїть на місці.

Клас “Road”. Вхідні дані: об’єкт типу Back, Menue, Enemy, Playergame, таймер, потоки. Вихідні дані: конструктор Road, метод paint, який відповідає за малювання, actionPerformed, який відповідає за перемальовування режимів (Гра, Меню), testWin- перевірка виграшу, testCollisionWithEnemies- перевірка програшу. Це основні методи, які є у цьому класі. Можна вважати, що це основний клас гри. Проте треба сказати, що у цьому класі відбуваються решта усіх змін, зв’язаних з грою. Наприклад тут обчислюється програвання музики під час гри, описується створення ворогів та перевірка виграшу та програшу. Також сюди підтягуються дані з класу Listeners. Важливо, що тут відмальовуються поточна швидкість гравця, потрібна відстань для виграшу та поточна пройдена відстань.

У гри є три режима складності, про які ми поговоримо згодом, але зараз варто зазначити, що, для того, щоб виграти, у кожному з рівнів потрібно проїхати свій шлях.

При виграші користувач побачить вікно виграшу. Варто зазначити, що якщо користувач захоче перезапустити рівень – усі дані оновляться і будуть мати початкові значення. Якщо ж він захоче вийти у меню – ці дані оновляться тільки після повторного входження у режим гри. Програш відбувається при зіткнені гравця з одним із ворогів, після чого користувач отримає вікно програшу.

Клас “Main”. Вхідні дані: Об’єкт типу Road, JFrame. Вихідні дані: Об’єкт типу JFrame. Саме з цього класу запускається наша програма, беручи усі дані з класу Road. Також тут відбувається налаштування самого вікна у якому відбувається гра. Тут виставляється розмір вікна, позиція, де воно буде відкриватись, видимість та заборона зміни розміра вікна.

2.3. Математичне забезпечення механіки гри

Під час написання коду для гри я використовував різні обрахунки, але слід виділити кілька важливих формул, які використовуються. Вони забезпечують взаємодію користувача з застосунком, плавність рухів у грі та сам рух моделей гравця та ворога. Розглянемо їх детально нижче.

Формула для обчислення швидкості відображення**:**

```
double v = (250 / Playergame.MAX_V) * p.v;
```
(2.1)

Ця формула використовується для перетворення внутрішньої швидкості гравця p.v у значення швидкості в км/год для відображення на екрані. Константа Playergame.MAX_V задає максимально можливу швидкість гравця, а значення 250 - це максимальна швидкість у км/год, яка відображатиметься. Таким чином, формула масштабує внутрішнє значення швидкості p.v до діапазону від 0 до 250 км/год.

Формула для обчислення координат гравця:

```
p.x = 30;
p.y = 250;
p.y -= dy;
if (p.y <= Playergame.MAX_TOP)
 p.y = Playergame.MAX_TOP;
if (p.y >= Playergame.MAX_BOTTOM)
p.y = Playergame.MAX_BOTTOM;
```
(2.2)

Ця група формул відповідає за обчислення та обмеження координат гравця. Початкові координати задаються явно: p.x = 30 та p.y = 250. Координата y змінюється шляхом віднімання значення dy, яке визначається натисканням клавіш керування. Потім координата y обмежується значеннями Playergame.MAX_TOP та Playergame.MAX_BOTTOM, щоб гравець не виходив за межі екрану.

Формула для обчислення прискорення гравця:

$$p.dv = 2.0; \quad p.dv = -0.7; \tag{2.3}$$

Ця проста формула задає значення прискорення гравця p.dv в залежності від натиснутих клавіш керування. Якщо натиснута клавіша вперед, то p.dv = 2.0, що призводить до збільшення швидкості. Якщо натиснута клавіша назад, то p.dv = -0.7, що призводить до уповільнення руху.

Формула для обчислення шляху, пройденого гравцем:

$$p.s \mathrel{+}= p.v; \tag{2.4}$$

Ця проста формула обчислює шлях p.s, пройдений гравцем, шляхом додавання поточної швидкості p.v до вже пройденого шляху на кожному кроці гри.

Ці формули є ключовими для реалізації основної механіки гри, такої як рух гравця, обчислення швидкості та пройденого шляху. Вони відповідають за коректне відображення та оновлення стану гри на кожному кроці циклу оновлення.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Вибір засобів розроблення

Гра написана на мові програмування Java (рис. 3.1.). Java — це об'єктно-орієнтована мова програмування. У офіційній реалізації, Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Oracle надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцезією GNU General Public License.

Рис. 3.1. Мова програмування Java.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в C/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім, Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням. За необхідності таких дій java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java Development Kit — безкоштовний розповсюджуваний Oracle комплект розробника застосунків на мові Java, який включає до себе компілятор Java, стандартні бібліотеки класів Java, приклади, документацію, різноманітні утиліти і виконавчу систему Java. В склад JDK не входить інтегроване середовище розробки на Java (IDE), тому розробник, що використовує тільки JDK, повинен використовувати текстовий редактор і компілювати та виконувати свої програми через утиліти командного рядка. [2, 6, 8, 13, 16, 17]

IntelliJ IDEA (рис.3.2.) — комерційне інтегроване середовище розробки для Java від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безплатної версії "Community Edition" і повнофункціональної комерційної версії "Ultimate Edition", для якої активні розробники відкритих проектів мають можливість отримати безплатну ліцензію. [5]

Рис.3.2. Середовище розробки IntelliJ IDEA.

Eclipse (рис.3.3.) це також середовище для розробки для Java, проте він має низку особливостей:

- можливість розробки ПЗ на багатьох мовах програмування (рідною є Java);
- платформонезалежна;
- модульна, призначена для подальшого розширення незалежним розробниками;
- з відкритим сирцевим кодом;
- розробляється і підтримується фондом Eclipse, куди входять такі постачальники ПЗ, як IBM, Oracle, Borland. [3]

Рис.3.3. Середовище розробки Eclipse.

Отже, під час написання роботи паралельно використовувались Eclipse та IntelliJ для різних цілей та тестування ПЗ. А також додаткові бібліотеки Swing та AWT.

Swing — інструментарій для створення графічного інтерфейсу користувача мовою програмування Java. Це частина бібліотеки базових класів Java ().

Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Тобто Swing включає в себе AWT та наслідує його. [1, 2, 8, 9, 12, 14]

3.2. Розроблення структури програми

Програма складається з 1 основного вікна, яке перемальовується у вікно з Меню або у Ігрове вікно. В свою чергу вікно Меню перемальовується у вікно з вибором рівней складності, далі вікно «Складності». Детальніша структурна схема зображена на рисунку нижче.

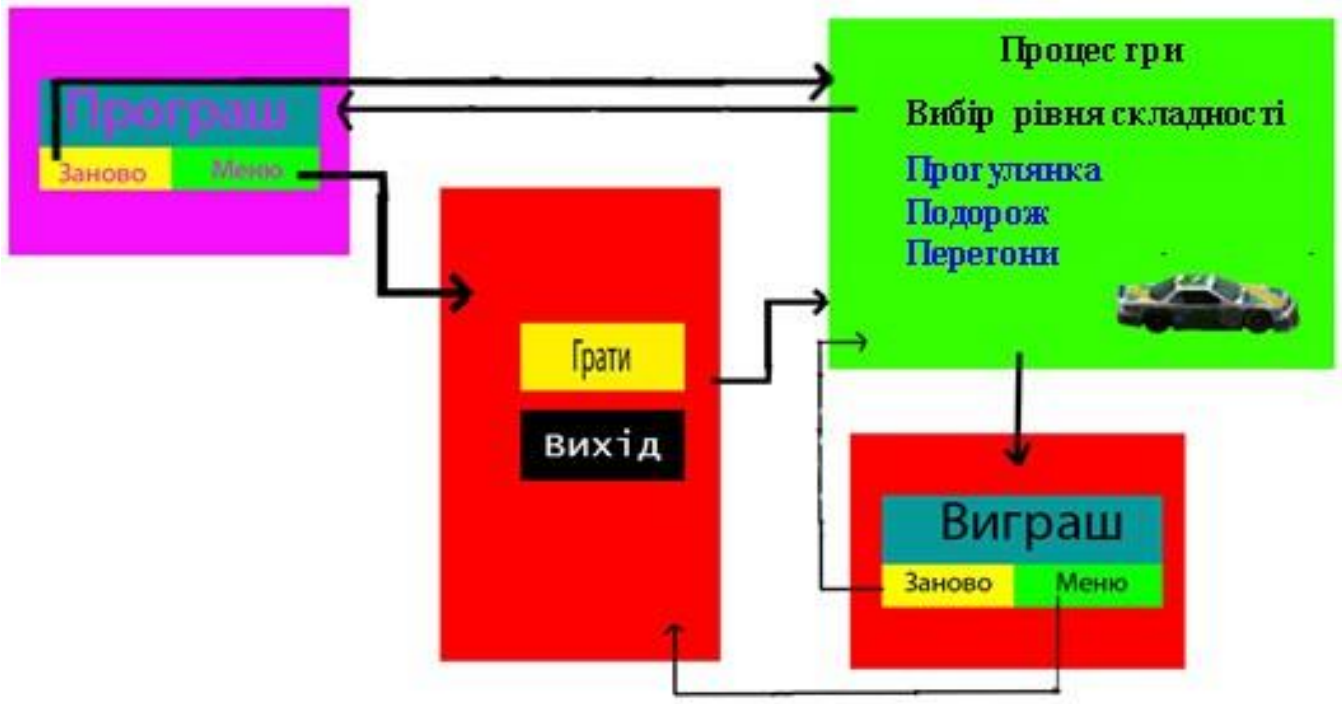

Рис. 3.4. Схематичне зображення вікон та їхньої взаємодії.

При натисканні на кнопку «Грати» у головному меню, вікно буде перемальоване у вікно «Складності», а після вибору відповідної кнопки у цьому вікні дані про складність гри буде передано до вікна «Процес гри». Після чого, якщо гравець виграє, то він отримає вікно «Виграш», якщо ж у ньому натиснути «Заново», вікно гри буде перемальоване знову у «Процес гри», якщо ж натиснути «Меню», вікно перемалюється в головне меню.

Інакше ж (якщо гравець програє), користувач отримає вікно «Програш», яке виконує ті ж функції, що і «Виграш». При натисканні на кнопку «Вихід» користувач вийде з програми. Як згадувалось раніше, для виграшу по замовчуванню стоїть режим «Прогулянка», у якому для виграшу потрібно проїхати 10000 пікселів, а для програшу зіткнутись з ворогом, як і у інших режимах.

Якщо просто навести курсор на кнопки «Грати» та «Вихід», то вони змінять свій опис на наступний (рис.3.6): Грати- Play, Вихід- Exit. Такої функції у додаткових вікнах «Виграш» та «Програш» не передбачено. Проте на кнопках у меню Складності є такий функціонал, з наступними значеннями:

Як сказано вище, головне вікно «Меню» (рис.3.5) має дві кнопки – «Грати» та «Вихід».

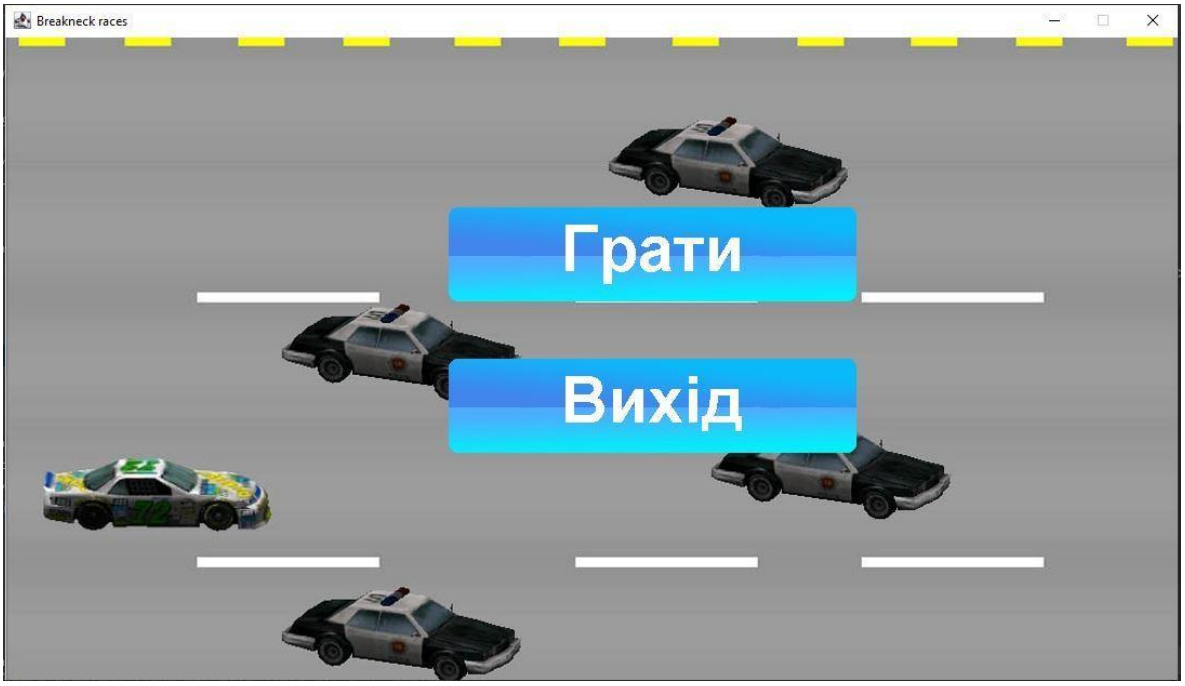

Рис. 3.5. Загальний вигляд головного вікна.

Рис.3.6. Вигляд кнопок Грати та Вихід при наведені курсора.

Детальніше компоненти вікна зображено у таблиці нижче.

Таблиця 3.1

Компоненти головного вікна

Назва компоненту	**Призначення**	**З якою метою використовується**
Кнопка «Грати»	Перехід у ігрове вікно з вибором Складності	Для перемальовування вікна у вікно із грою.
Кнопка «Вихід»	Вихід із програми	Для повного виходу із програми, тобто її закриття.

Як зазначалось раніше у нас є вікно Складностей. Гра має три рівні складності (рис.3.7.):

Прогулянка (Chill), Подорож (Travel), Перегони (Drag race).

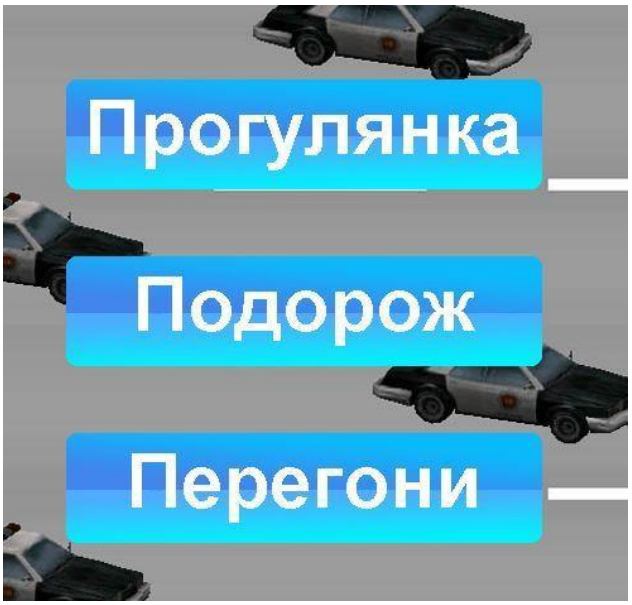

Рис.3.7. Вигляд кнопок вікна Складності.

При наведені курсора на кожну з них вони будуть виглядати так, як показано на рисунку нижче.

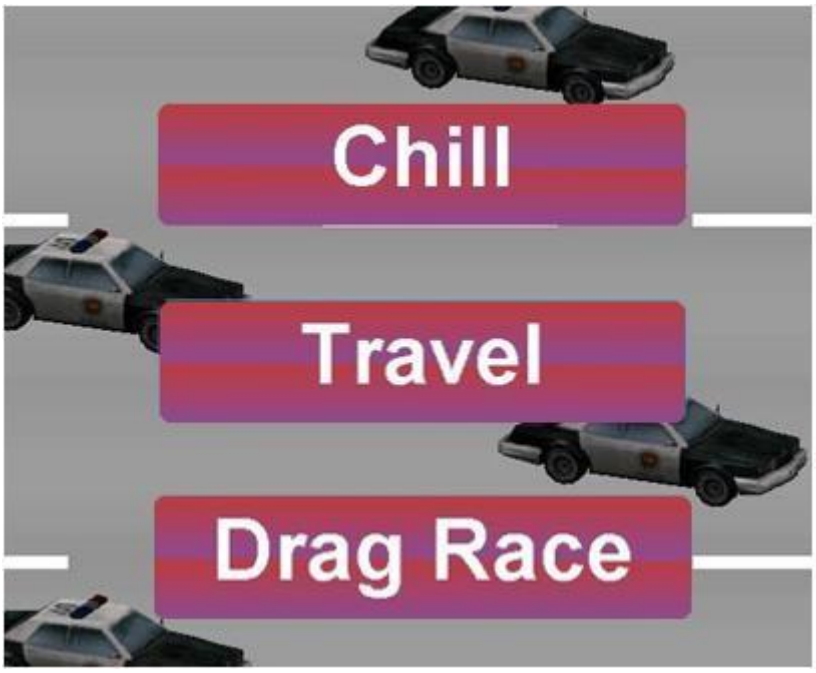

Рис.3.8. Вигляд кнопок вікна Складності при наведені на них курсора.

В загальному Ігрове вікно також має декілька елементів, а саме зображень, які змінюються у процесі гри. В першу чергу слід зазначити, що в залежності від Складності рівня буде дещо видозмінене вікно, а саме його верхня частина (рис.3.9).

а) вигляд ігрового вікна складності Прогулянка

б) вигляд ігрового вікна складності Подорож

в) вигляд ігрового вікна складності Перегони

Рис.3.9. Вигляд Ігрового вікна в залежності від Складності: а) Прогулянка, б) Подорож, в) Перегони.

Можна побачити, що ігрове вікно складається з декількох текстових позначок. Ці позначки позначають: поточну швидкість гравця, додатково вибрану складність та максимальну швидкість, пройдену відстань та потрібну відстань для виграшу.

Також ігрове вікно складається з зображень гравця, у нього є декілька позицій, в залежності куди рухається гравець. А саме це вперед, наліво (вгору на екрані), направо (вниз), чи стоїть на місці.

Також у цьому вікні є зображення Ворога та дороги, яке постійно рухається. Детальніше опис цих елементів можна побачити у таблиці нижче. Зображення Ворога та Гравця буде показано у наступному підрозділі. Детальніше усі елементи ігрового вікна можна побачити у таблиці нижче.

Таблиця 3.2

Компоненти ігрового вікна

Назва компоненту	**Призначення**	**З якою метою використовується**
Зображення 1 (дорога)	Для візуального відображення	Для відображення руху дороги
Зображення 2 (гравець)	Для візуального відображення	Для відображення руху гравця
Зображення 3 (ворог)	Для візуального відображення	Для відображення ворога
Мітка 1 (Швидкість)	Для змінного відображення	Для відображення поточної швидкості
Мітка 2 (Дистанція)	Для змінного відображення	Для відображення пройденої дистанції
Мітка 3 (Потрібна дистанція)	Для візуального відображення	Для відображення дистанції, яка потрібна для виграшу
Мітка 4 (Складність)	Для візуального відображення	Для відображення вибраного рівня складності
Мітка 5 (Максимальна швидкість)	Для візуального відображення	Для відображення максимально-допустимої швидкості

Як було сказано раніше, при виграші та програші ми отримуємо відповідні додаткові вікна (рис.3.10, рис.3.11). Вже зрозуміло, що виграш відбувається при перетині гравцем потрібної дистанції. Програш відбувається при зіткнені гравця та ворога.

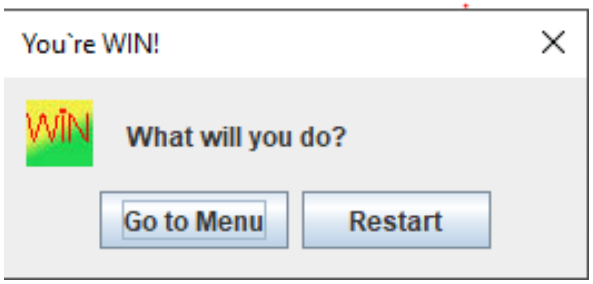

Рис.3.10. Вікно виграшу.

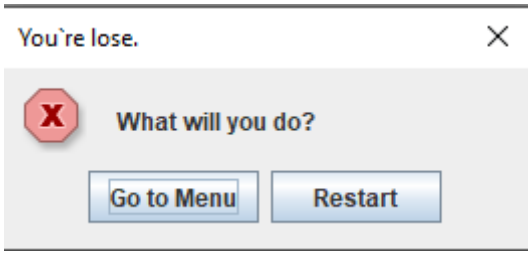

Рис.3.11. Вікно програшу.

Видно, що ці вікна виконують однакові функції, але при різних діях. Отже, кнопка «Go to Menu» повертає нас у вікно «Меню», а кнопка «Restart» розпочинає гру спочатку. Детальніше про процес гри ми поговоримо у наступних розділах.

3.3. Розроблення макетів для елементів програми

У грі є деякі елементи, які потребують власного зображення, а саме це дорога, гравець та вороги вороги. Усі початкові зображення відбирались з відкритого доступу у Інтернеті. При пошуку зображень для гравця я зупинився на макеті, який зображено на рисунку нижче. [4]

Рис.3.12. Загальний макет для зображень гравця.

Для того, щоб під час руху гравець не стояв на місці, а повертався було відібрано декілька елементів, з представлених вище. А саме такі елементи, при яких макет гравця буде двитись униз по діагоналі та аналогічно вверх. Крім цих двох позицій, третя – просто, коли гравець їде прямо. В результаті вийшло 3 спрайти для гравця. Щоб вирізати елементи, які зображено на рисунку 3.13, було використано спеціальне програмне забезпечення з відкритим доступом.

Рис.3.13. Готові макети для гравця.

Для ворогів було вибрано один-єдиний шаблон (рис.3.14), який не матиме функціоналу повороту, тому він дивиться просто прямо. [4]

Рис.3.14. Макет зображення для ворогів.

Дорогу довелось намалювати власноруч, оскільки гідного варіанту не знайшлось. Результат можна побачити на малюнку нижче.

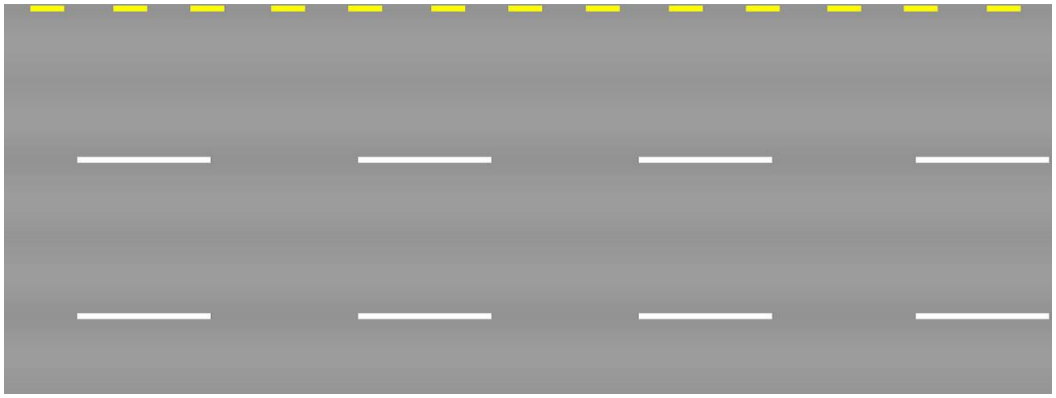

Рис.3.15. Макет для дороги.

Для кнопок меню потрібно 4 шаблони, а саме 2 з надписом українською мовою та 2 англійською. Самі надписи слів відмальовуються програмного, а задній фон (рис.3.16, рис.3.17) задається файлом з зображенням. Ці кнопки було створено власноруч, за допомогою градієнтної заливки.

Рис.3.16. Задній фон кнопки для надпису українською мовою.

Рис.3.17. Задній фон кнопки для надпису англійською мовою.

Аналогічні шаблони використовуються для кнопок у вікні Складності.

Фон головного меню створено під час гри, а саме коли з'явилось декілька ворогів, зображення було збережено та використано його як заставку (рис.3.18) для гри.

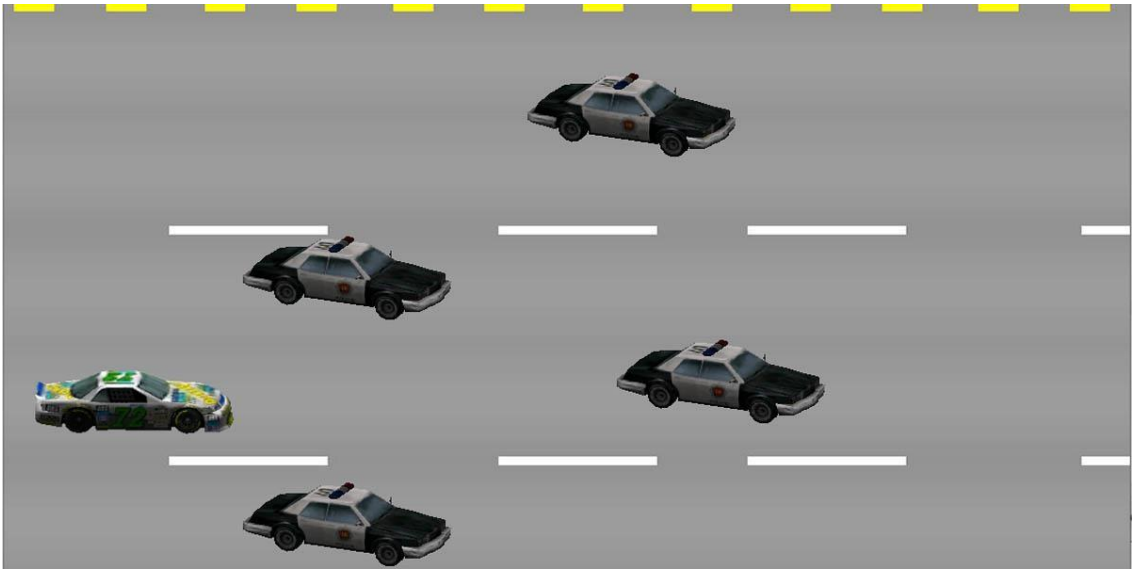

Рис.3.18. Фон ігрового меню.

Також для вікна з виграшем додано особливу іконку (рис.3.19).

Рис.3.19. Іконка для вікна з виграшем.

3.4. Опис роботи гри

Як згадувалось раніше при заході у гру користувач потрапляє у ігрове меню. У користувача з'являється два віріанта розвитку подій: натиснути кнопку Грати або кнопку Вихід. Детальніше цей процес зображено у блок-схемі нижче.

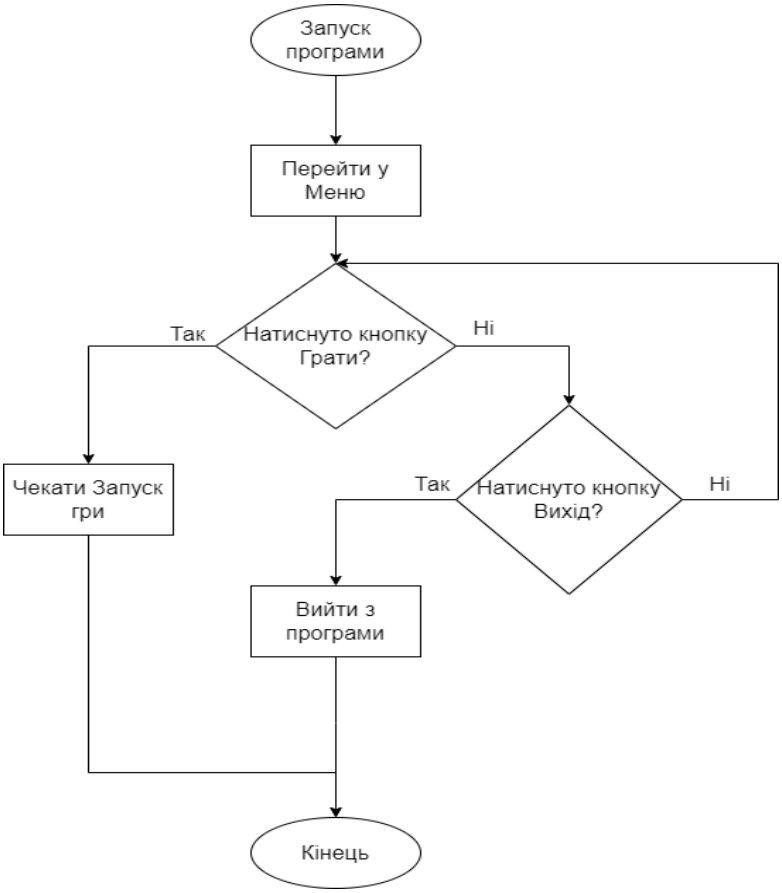

Рис.3.20. Блок-схема запуску програми.

Після того, як користувач натисне кнопку Грати він буде чекати запуску програми. Маєте на увазі, що гра не розпочнеться, допоки він не вибере потрібну складність. Після вибору складності кооритсувач потрапить у вікно з грою і він зможе натискати певні клавіші для того, щоб рухатись автомобілем. Варто зазначити, що додатково він може натиснути клавішу Esc, щоб вийти у вікно Меню. Блок-схема дій у вікні з грою зображено на рисунку 3.21.

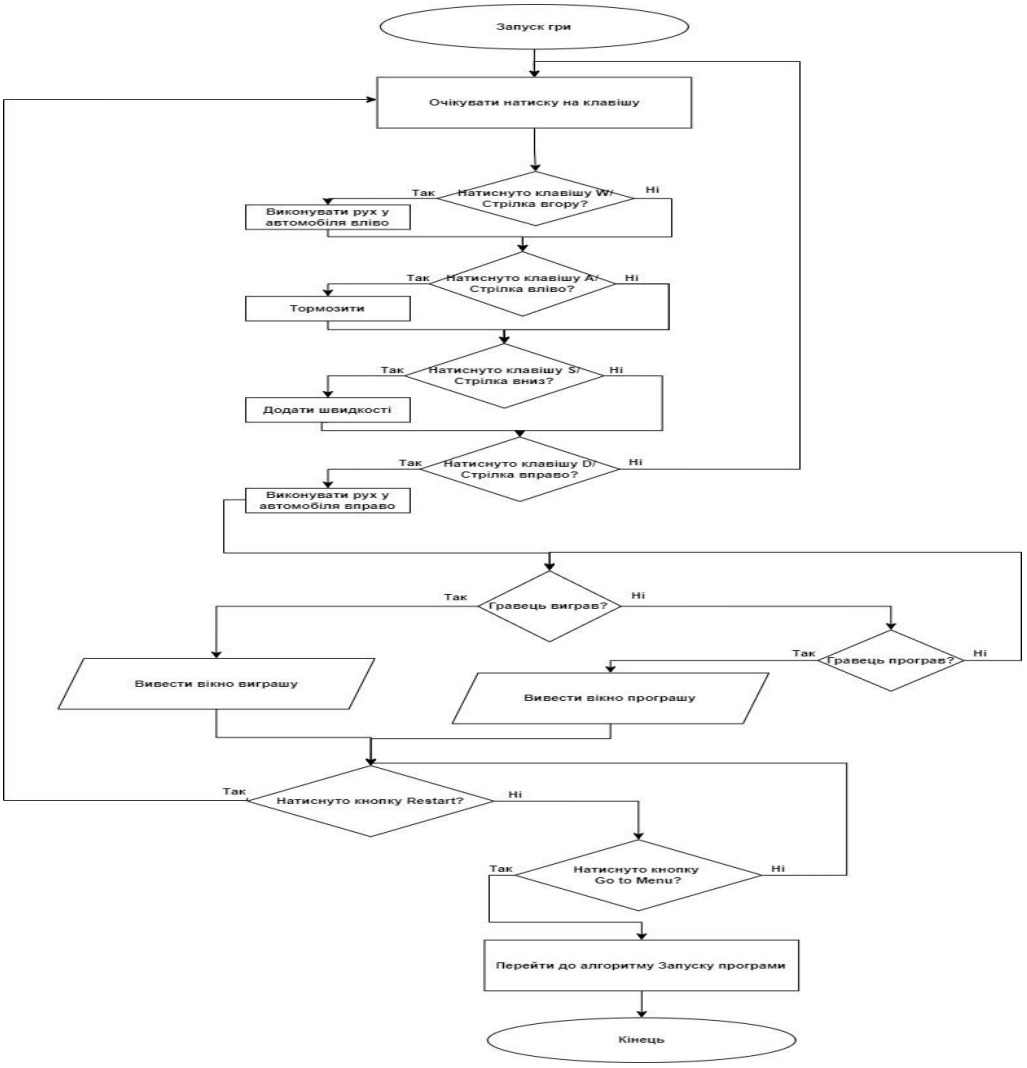

Рис.3.21. Блок-схема процесу гри.

Як можна побачити натискаючи певні клавіші гравець буде пересуватись по екрані. Детальний рух по клавішам зображено на рисунку нижче.

Рис.3.22. Клавіші для руху гравця.

3.5. Тестування створеної гри

Тестування гри відбувалось під час її проходження, тобто тестувався сам процес гри. Нижче описано деякі види тестування та їх опис.

Тестування на вихід за кордони дороги (рис.3.23). Якщо гравець намагається виїхати за верхній або нижній кордон, він буде впиратися у ці кордони, якщо він захоче їхати назад, то просто зупиниться, якщо ж він захоче їхати більше дозволеної швидкості, то гра цього не дозволить і він буде їхати з максимальною швидкістю, яку вказано у коді. Варто зазначити, що у кожного з рівнів складності власна максимальна швидкість, та під час тестування усі режими пройшли випробування. Тобто автомобіль у всіх складностях не їхав зі швидкістю більшою за дозволену. Можна побачити, що при спробі виконати дії, зазначені вище, гра цього не дозволяє та виконуються обмеження, отже тестування на вихід за кордони дороги успішне.

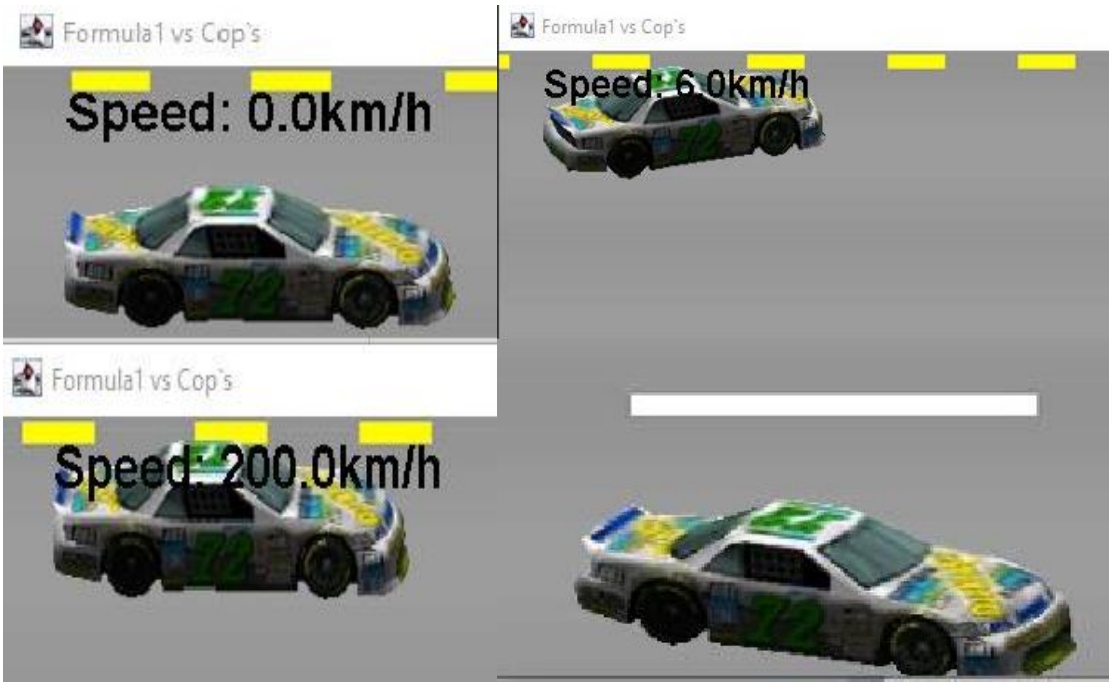

Рис. 3.23. Перевірка кордонів гри.

Тестування на виграш та програш гравця. Як зазначалось раніше, щоб виграти у нашій грі, в залежності від типу складності, потрібно проїхати різну кількість пікселів. У першому рівні складності - 10000 пікселів, другому – 30000, а третьому – 100000. Після того, як це зробити, користувач отримає вікно з виграшем (рис.3.24). Тобто тестування на виграш працює успішно.

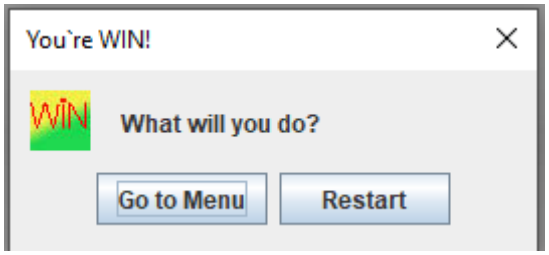

Рис. 3.24. Тестування виграшу.

Програшем рахується зіткнення гравця з одним із ворогів. Це відбувається шляхом обчислення прямокутників зображень двох об’єктів. Варто зазначити, що зіткнення може відбутися зі всіх сторін. А саме – ззаду, коли гравець заїде ззаду (рис.3.25); спереду, коли ззаду в’їдуть у гравця (рис.3.26); зверху, коли гравець буде переміщуватись униз і потрапить на ворога (рис.3.27); а також знизу, коли гравець буде пересуватись угору (рис.3.28).

Рис.3.25. Перевірка зіткнення спереду.

Рис.3.26. Перевірка зіткнення ззаду.

Рис.3.27. Перевірка зіткнення зверху.

Рис.3.28. Перевірка зіткнення знизу.

Отже, після усіх цих зіткнень користувач отримає вікно програшу (рис.3.29), а отже все працює вірно.

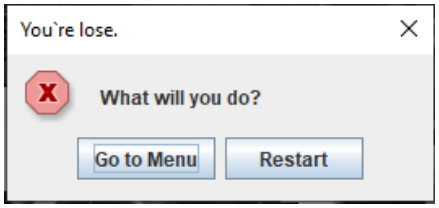

Рис.3.29. Вікно програшу.

Таким чином, усі тести перевірили працездатність гри та умови тих чи інших подій, та визначили, що застосунок працює без помилок.

ВИСНОВКИ

Під час розроблення програми здійснено аналітичний огляд наявних програм-аналогів на ринку 2D та 3D ігор, а також гонок. Ця програма буде корисною як для відпочинку, так і для розвитку, оскільки вона розвиває моторику рук, швидкість прийняття рішень та логічне мислення.

При проектуванні здійснено вибір базового програмного забезпечення, необхідного для розробки гри. Середовищем для роботи було вибрано Intellij IDEA, проте деколи для тестування використовувався Eclipse. У програмі створено 7 класів, які відповідають за функціонал гри. Гра складається з вікна з меню та вікна з грою. При переході з режиму меню у режим гри, користувач натискаючи клавіші WASD або стрілки на клавіатурі може пересуватись у грі. Також процес гри супроводжуються музикальним супроводом.

Протестовано обмеження гравця в межах екрану, максимальну швидкість (250 км/год, проте у кожному рівні складності – різні швидкості) та дистанцію для виграшу. Коректно працюють умови виграшу/програшу. Тестування показало успішні результати, отже можна сказати, що гра працює без нарікань.

Отже, «Breakneck races» є конкурентоспроможним проєктом та має великі шанси успішно вийти на ринок. Також гра підійде для будь-якого віку користувачів. «Breakneck races» відповідає усім поставленим завданням.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. AWT and Swing in Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.javatpoint.com/awt-and-swing-in-java.
2. Burd B. Java For Dummies, 7th Edition / Barry Burd., 2017. – 504 с.
3. Eclipse [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Eclipse.
4. Free 2d Car Sprites [Електронний ресурс] – Режим доступу до ресурсу: https://forum.thegamecreators.com/thread/211001.
5. IntelliJ IDEA. [Електронний ресурс] – Режим доступу до ресурсу: https://javarush.com/ua/groups/posts/uk.2311.stvorennja-proektu-v-intellij-idea.
6. Java [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Java.
7. Need for Speed (серія ігор) [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Need_for_Speed_(серія_ігор).
8. Schildt H. Java: A Beginner\'s Guide, Eighth Edition / Herbert Schildt., 2018. – 720с.
9. Swing Java [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Swing_(Java).
10. Vampire Survivors в магазині відеоігор. [Електронний ресурс] – Режим доступу до ресурсу: https://store.steampowered.com/app/1794680/Vampire_Survivors/.
11. Vampire Survivors. Загальна інформація. [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Vampire_Survivors.
12. Zukowski J. The Definitive Guide to Java Swing: 3rd Edition / John Zukowski., 2005. – 899 с.
13. Бейтс Б. Head First Java / Б. Бейтс, К. Сьєрра., 2023. – 720 с.
14. Екштайн Р. Java Swing / Р. Екштайн, Д. Вуд, М. Лой., 1998. – 1221 с.
15. Розмір ринку відеоігор, частка та звіт про аналіз тенденцій за пристроями, за типом, за регіоном та прогнозами за сегментами, 2023–2030 рр. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.grandviewresearch.com/industry-analysis/video-game-market.

16. Технологія Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/en/download/help/whatis_java.html.
17. Що таке Java і в чому її особливості [Електронний ресурс] – Режим доступу до ресурсу: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/.
18. Якість програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Якість_програмного_забезпечення.

ДОДАТКИ

Текст коду програми

Клас Back

```
package diplom;
import java.awt.*;
import javax.swing.*;
public class Back {
   Image img1 = new ImageIcon("res/bg_road.jpg").getImage();//загрузка картинки
   Image img2 = new ImageIcon("res/fon2.jpg").getImage();//загрузка картинки
   public void draw(Graphics2D g) {  //малювання  в Graphics2D
      if (Road.state.equals(Road.STATES.MENUE))  g.drawImage(img2,(int)0,(int)0,null);
      if (Road.state.equals(Road.STATES.PLAY)) g.drawImage(img1,(int)0,(int)0,null);// відмальовуємо елемент
в координатах
   }
}
```

Клас Enemy

```
package diplom;

import javax.swing.*;
import java.awt.*;

public class Enemy {

   Image img = new ImageIcon("res/enemy.png").getImage();
   int x;
   int y;
   int v;
   Road road;

   public  Rectangle getRect() {
      return new Rectangle(x,y,180,65);
   }

   public Enemy (int x, int y, int v, Road road) {
      this.x = x;
      this.y = y;
      this.v = v;
```

```
        this.road = road;
    }
    public void move() {
        x = x - road.p.v + v;
    }
}
```

Клас Listeners

```
package diplom;
import java.awt.event.*;
public class Listeners implements MouseListener, MouseMotionListener, KeyListener {
    public void mouseClicked(MouseEvent e) {
    }
    public void mousePressed(MouseEvent e) {
        if (e.getButton() == MouseEvent.BUTTON1){
            if (Road.state == Road.STATES.MENUE){
                Menue.click= true;
            }
        }

    }
    @Override
    public void mouseReleased(MouseEvent e) {

        if (e.getButton() == MouseEvent.BUTTON1){
            if (Road.state == Road.STATES.MENUE){
                Menue.click= false;
            }
        }
    }
    @Override
    public void mouseEntered(MouseEvent e) {
    }
    @Override
    public void mouseExited(MouseEvent e) {
    }
```

```
    @Override
    public void mouseDragged(MouseEvent e) {
    }
    public void mouseMoved(MouseEvent e) {
        Road.mouseX = e.getX();
        Road.mouseY = e.getY();    }
    @Override
    public void keyTyped(KeyEvent e) {
    }
    @Override
    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();
        if (key == KeyEvent.VK_ESCAPE){
            if (Road.state == Road.STATES.PLAY) Road.state = Road.STATES.MENUE;
        }
    }
    @Override
    public void keyReleased(KeyEvent e) {
    }
}
```

Клас Main

```
package diplom;

import java.io.IOException;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.swing.JFrame;



public class Main {
    public static void main(String[] args) throws LineUnavailableException, IOException,
UnsupportedAudioFileException {
        JFrame f = new JFrame("Breakneck races");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLocation(250, 120);
        f.setSize(1100, 633);
```

```
        f.add(new Road()); // створення дороги
        f.setResizable(false); // заборона зміни розміра вікна
        f.setVisible(true);
    }


}
```

Клас Menue та його підклас

```
package diplom;
import java.awt.*;
import javax.swing.*;
public class Menue {
    public static boolean click = false;
    private boolean showingLevels = false;

    ButtMenue button1 = new ButtMenue(400,150,396,100,"res/but1.png","Грати");
    ButtMenue button2 = new ButtMenue(400,290,396,100,"res/but1.png","Вихід");

    ButtMenue buttonChill = new ButtMenue(400, 150, 396, 100, "res/but1.png", "Прогулянка");
    ButtMenue buttonTravel = new ButtMenue(400, 290, 396, 100, "res/but1.png", "Подорож");
    ButtMenue buttonDragRace = new ButtMenue(400, 430, 396, 100, "res/but1.png", "Перегони");
    public void draw(Graphics2D g) {
        if (showingLevels) {
            buttonChill.draw(g);
            buttonTravel.draw(g);
            buttonDragRace.draw(g);
        } else {
            button1.draw(g);
            button2.draw(g);
        }
    }
    public void moveButt(Menue.ButtMenue e) {
        if (Road.mouseX > e.getX() && Road.mouseX < e.getX() + e.getW() &&
                Road.mouseY > e.getY() && Road.mouseY < e.getY() + e.getH()) {
            e.s = "res/but2.png";

            if (showingLevels) {
                if (e.equals(buttonChill)) {
```

```
            e.f = "Chill";
            if (Menue.click) {
                Road.setDifficultyLevel(Road.DifficultyLevel.CHILL);
                startGame();
            }
        } else if (e.equals(buttonTravel)) {
            e.f = "Travel";
            if (Menue.click) {
                Road.setDifficultyLevel(Road.DifficultyLevel.TRAVEL);
                startGame();
            }
        } else if (e.equals(buttonDragRace)) {
            e.f = "Drag Race";
            if (Menue.click) {
                Road.setDifficultyLevel(Road.DifficultyLevel.DRAG_RACE);
                startGame();
            }
        }
    } else {
        if (e.equals(button1)) {
            e.f = "Play";
            if (Menue.click) {
                showingLevels = true;
                Menue.click = false;
            }
        } else if (e.equals(button2)) {
            e.f = "Exit";
            if (Menue.click) {
                System.exit(1);
            }
        }
    }
} else {
    e.s = "res/but1.png";
    if (e.equals(button1)) {
        e.f = "Грати";
    } else if (e.equals(button2)) {
        e.f = "Вихід";
```

```
        } else if (e.equals(buttonChill)) {
            e.f = "Прогулянка";
        } else if (e.equals(buttonTravel)) {
            e.f = "Подорож";
        } else if (e.equals(buttonDragRace)) {
            e.f = "Перегони";
        }
    }
}

private void startGame() {
    showingLevels = false;
    Road.state = Road.STATES.PLAY;
    Menue.click = false;
    Road.getInstance().startNewGame();
}
class ButtMenue {
    private double x;
    private double y;
    private double w;
    private double h;
    public Color color1;
    public  String f;
    public  String s;
    public ButtMenue(int x, int y, int w, int h, String s, String f) {
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
        this.f = f;
        this.s = s;
        color1 = Color.WHITE;
    }
    public double getX(){
        return this.x;
    }
    public double getY(){
        return this.y;
```

```
        }
        public double getW(){
            return this.w;
        }
        public double getH(){
            return this.h;
        }
        public void draw(Graphics2D g){
            g.drawImage(new ImageIcon(s).getImage(),(int)x,(int)y,null);
            g.setColor(color1);
            Font font = new Font("Arial",Font.BOLD,60);
            g.setFont(font);
            long length = (int) g.getFontMetrics().getStringBounds(f,g).getWidth();
            g.drawString(f,(int)(x+w/2)- (int)(length/2),(int)y +(int) (h/3)*2 );
        }
    }
}
```

Клас Playergame

```
package diplom;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.event.KeyEvent;
import javax.swing.ImageIcon;

import static diplom.Road.currentDifficultyLevel;

public class Playergame {

    public static int MAX_V = currentDifficultyLevel.maxSpeed;
    public static final int MAX_TOP = 10;
    public static final int MAX_BOTTOM = 520;

    Image img_c = new ImageIcon("res/player.png").getImage();
    Image img_l = new ImageIcon("res/player_left.png").getImage();
    Image img_r = new ImageIcon("res/player_right.png").getImage();
```

```
Image img = img_c;



public  Rectangle getRect() { return new Rectangle(x,y,180,60);}

int v = 1; // швидкість
double  dv = 0; // прискорення
int s = 0; // шлях

int x = 30;// початкові координати
int y = 250;
int dy = 0; // змінна координат


int layer1 = 0; // зображення 1
int layer2 = 1200; // зображення 2
public void move() { // метод руху
   s += v;
   v += dv;
   if (v <= 0) v = 0;
   if (v >= MAX_V) v = MAX_V;
   y -= dy;
   if (y <= MAX_TOP) y= MAX_TOP;
   if (y >= MAX_BOTTOM) y= MAX_BOTTOM;

   if(layer2 - v <=0) {
      layer1 = 0;
      layer2 = 1200;
   } else {

      layer1 -= v;
      layer2 -= v;
   } // малювання без пробілів


}
```

```
    public void keyPressed(KeyEvent e){ // коли натиснута клавіша
        int key = e.getKeyCode();
        if (key == KeyEvent.VK_RIGHT || key == KeyEvent.VK_D) {
            dv = 2.0;
        }
        if (key == KeyEvent.VK_LEFT || key == KeyEvent.VK_A) {
            dv = -0.7;
        }
        if (key == KeyEvent.VK_UP || key == KeyEvent.VK_W) {
            dy = 9;
            img = img_l;
        }
        if (key == KeyEvent.VK_DOWN || key == KeyEvent.VK_S) {
            dy = -9;
            img = img_r;
        }
    }
    public void keyReleased(KeyEvent e){ // коли клавіша відпущена
        int key = e.getKeyCode();
        if (key == KeyEvent.VK_RIGHT || key == KeyEvent.VK_D || key == KeyEvent.VK_LEFT || key ==
KeyEvent.VK_A) {
            dv =  0;

        }
        if (key == KeyEvent.VK_UP || key == KeyEvent.VK_W || key == KeyEvent.VK_DOWN || key ==
KeyEvent.VK_S) {
            dy = 0;
            img = img_c;
        }
      }
 }
```

Клас Road

```
package diplom;

import java.awt.Color;
```

```
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.UnsupportedAudioFileException;
import javax.swing.ImageIcon;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.Timer;

public class Road extends JPanel implements ActionListener, Runnable {
    private static Road instance;
    public static int WIDTH = Toolkit.getDefaultToolkit().getScreenSize().width;
    public static int HEIGHT = Toolkit.getDefaultToolkit().getScreenSize().height;
    public static int endDist;
    public static int mouseX;
    public static int mouseY;

    public static enum STATES {
        MENUE, PLAY
```

```
}

File file = new File("res/Masked-Wolf-Astronaut-In-The-Ocean.wav");
AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);
Clip clip = AudioSystem.getClip();



public static STATES state = STATES.MENUE;

private BufferedImage image;
private Graphics2D g;

ImageIcon icon = new ImageIcon("res/icon.png");

Timer mainTimer = new Timer(30, this);

Back back = new Back();
Menue menue = new Menue();

Image img = new ImageIcon("res/bg_road.jpg").getImage();

Playergame p = new Playergame();

Thread enemiesFactory = new Thread(this);



List<Enemy> enemies = new ArrayList<Enemy>();

public Road() throws LineUnavailableException, IOException, UnsupportedAudioFileException {
    instance = this;
    clip.open(audioStream);

    setFocusable(true);
    requestFocus();
    mainTimer.start();
    enemiesFactory.start();
    if (state.equals(STATES.PLAY)) ;
    addKeyListener(new MyKeyAdapter());
```

```
    image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);
    g = (Graphics2D) image.getGraphics();
    addMouseListener(new Listeners());
    addMouseMotionListener(new Listeners());
    addKeyListener(new Listeners());

}
public static Road getInstance() {
    return instance;
}
public void startNewGame() {


    Road.state = Road.STATES.PLAY;

    p.img = p.img_c;
    p.v = 1; // ��������
    p.dv = 0; // ����������
    p.s = 0; // ����
    p.x = 30;// ��������� ������������
    p.y = 250;
    p.dy = 0; // ������ ������������
    p.layer1 = 0; // ������������ 1
    p.layer2 = 1200; // ������������ 2
    enemies.clear();
    mainTimer.restart();
    if (!enemiesFactory.isAlive()) {
        enemiesFactory = new Thread(this);
        enemiesFactory.start();
    }
    repaint();
}

private class MyKeyAdapter extends KeyAdapter {
    @Override
    public void keyPressed(KeyEvent e) {
        p.keyPressed(e);
```

```
        }
        public void keyReleased(KeyEvent e) {
            p.keyReleased(e);
        }

    }
    public void paint(Graphics g) {
        g = (Graphics2D) g; // ������������ ����
        g.drawImage(img, p.layer1, 0, null);
        g.drawImage(img, p.layer2, 0, null);
        g.drawImage(p.img, p.x, p.y, null);
        double v = p.v;
        g.setColor(Color.BLACK);
        Font font = new Font("Arial", Font.BOLD, 25);
        g.setFont(font);
        g.drawString("Speed: " + v + "km/h", 30, 30);
        g.drawString("Need drive: " + endDist, 850, 30);
        g.drawString("You`re distance: " + p.s, 500, 30);
        font = new Font("Arial", Font.BOLD, 12);
        g.setFont(font);
        g.drawString("Difficulty: " + currentDifficultyLevel, 250, 30);
        g.drawString("Max Speed: " + p.MAX_V + "km/h", 380, 30);
        Iterator<Enemy> i = enemies.iterator();
        while  (i.hasNext())  {
            Enemy e = i.next();
            if (e.x >= 2400 || e.x <= -2400) {
                i.remove();
            } else {
                e.move();
                g.drawImage(e.img, e.x, e.y, null);
            }

}

}
    public void actionPerformed(ActionEvent e) {
        if (state.equals(STATES.MENUE)) {
            gameRender();
            menue.draw(g);
            menue.moveButt(menue.button1);
```

```
            menue.moveButt(menue.button2);
            menue.moveButt(menue.buttonChill);
            menue.moveButt(menue.buttonTravel);
            menue.moveButt(menue.buttonDragRace);
            gameDraw();
            clip.stop();
        }
        if (state.equals(STATES.PLAY)) {
            clip.start();
            p.move();
            testCollisionWithEnemies();
            testWin();
            repaint(); // ����������������
        } }
    public void gameRender() {
        back.draw(g);
    }
    private void gameDraw() {
        Graphics g2 = this.getGraphics();
        g2.drawImage(image, 0, 0, null);
        g2.dispose();
    }
    private void testWin() {
        if (p.s >= currentDifficultyLevel.endDistance) {
            clip.stop();
            enemies.clear();
            mainTimer.stop();
            String[] wins = { "Go to Menu", "Restart" };
            int answer =
                JOptionPane.showOptionDialog(null, "What will you do?", "You`re WIN!",
                    JOptionPane.YES_NO_OPTION, JOptionPane.INFORMATION_MESSAGE, icon,
wins, null);
            if (answer == 0 || answer== -1) {
                clip.stop();
                clip.setMicrosecondPosition(0);
                repaint();
                mainTimer.restart();
                state = STATES.MENUE;
```

```
            back.draw(g);
            menue.draw(g);
            menue.moveButt(menue.button1);
            menue.moveButt(menue.button2);
            gameDraw();
            p.img = p.img_c;
            p.v = 1; // ��������
            p.dv = 0; // ����������
            p.s = 0; // ����
            p.x = 30;// ��������� ����������
            p.y = 250;
            p.dy = 0; // ������ ������������
            p.layer1 = 0; // ������������ 1
            p.layer2 = 1200; // ������������ 2
        }
        if (answer == 1) {
            clip.setMicrosecondPosition(0);
            p.img = p.img_c;
            p.v = 1; // ��������
            p.dv = 0; // ����������
            p.s =0; // ����
            p.x = 30;// ��������� ����������
            p.y = 250;
            p.dy = 0; // ������ ������������
            p.layer1 = 0; // ������������ 1
            p.layer2 = 1200; // ������������ 2
            repaint();
            mainTimer.restart();
        }            }  }
    private void testCollisionWithEnemies() {
        Iterator<Enemy> i = enemies.iterator();
        while (i.hasNext()) {
            Enemy e = i.next();
            if (p.getRect().intersects(e.getRect())) {
                mainTimer.stop();
                clip.stop();
                enemies.clear();
```

```
String[] responses = { "Go to Menu", "Restart" };

int answer = JOptionPane.showOptionDialog(null, "What will you do?", "You`re lose.",
        JOptionPane.YES_NO_OPTION, JOptionPane.ERROR_MESSAGE, null, responses, null);
if (answer == 0 || answer == -1) {
    enemies.clear();
    clip.stop();
    clip.setMicrosecondPosition(0);
    repaint();
    mainTimer.restart();
    state = STATES.MENUE;
    back.draw(g);
    menue.draw(g);
    menue.moveButt(menue.button1);
    menue.moveButt(menue.button2);
    gameDraw();
    p.img = p.img_c;
    p.v = 1; // ��������
    p.dv = 0; // ����������
    p.s = 0; // ����
    p.x = 30;// ��������� ����������
    p.y = 250;
    p.dy = 0; // ������ ������������
    p.layer1 = 0; // ������������ 1
    p.layer2 = 1200; // ������������ 2
}
if (answer == 1) {
    clip.setMicrosecondPosition(0);
    p.img = p.img_c;
    p.v = 1; // ��������
    p.dv = 0; // ����������
    p.s = 0; // ����
    p.x = 30;// ��������� ����������
    p.y = 250;
    p.dy = 0; // ������ ������������
    p.layer1 = 0; // ������������ 1
    p.layer2 = 1200; // ������������ 2
```

```
                repaint();
                mainTimer.restart();

            }
        }
    }
}
    public enum DifficultyLevel {
        CHILL(10000, 110),
        TRAVEL(30000, 190),
        DRAG_RACE(100000, 250);

        int endDistance;
        int maxSpeed;

        DifficultyLevel(int endDistance, int maxSpeed) {
            this.endDistance = endDistance;
            this.maxSpeed = maxSpeed;
        }
    }

    protected static DifficultyLevel currentDifficultyLevel = DifficultyLevel.CHILL;

    public static void setDifficultyLevel(DifficultyLevel level) {
        currentDifficultyLevel = level;
        endDist = level.endDistance;
        Playergame.MAX_V = level.maxSpeed;
        // ������������ MAX_V ��� ������
//        if (instance != null && instance.p != null) {
//            instance.p.MAX_V = level.maxSpeed;
//        }
    }


    @Override
    public void run() {
```

```
        while (true) {
            Random rand = new Random();
            try {
                Thread.sleep(rand.nextInt(2000));
                enemies.add(new Enemy(2000, rand.nextInt(510), rand.nextInt(60), this)); //
□□□□□□□□□□ "□□□□□□□"
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

}
    }
}
```