НАВЧАЛЬНО-НАУКОВИЙ КОМПЛЕКС

“ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ”

НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ “КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

**КУРСОВА РОБОТА**

3 дисципліни: Програмування

На тему: «Іподром»

Студентки 1 курсу групи KП-22

Спеціальність 121 Інженерія програмного забезпечення

Лешкович Олександри Олексіївни

Національно оцінка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів:\_\_\_\_ Оцінка ECTS \_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ 2023

**Зміст**

|  |  |
| --- | --- |
| Вступ | 3 |
| Розділ 1. Моделювання надання життя об’єктів у комп’ютерних іграх. | 4 |
| 1.1. Фізична та математична моделі. | 4 |
| 1.1.1. Фізичне моделювання. | 4 |
| 1.1.2. Математичне моделювання. | 7 |
| 1.2. Рух об’єктів по колу. Полярні координати. Кутова швидкість. | 8 |
| Розділ 2. Концепції та середовища, які були використані в розробці програмного забезпечення. | 10 |
| 2.1. Поліморфізм. | 11 |
| 2.2. Інкапсуляція. | 13 |
| 2.3. Динамічна пам’ять. | 15 |
| 2.4. Спадкування. | 16 |
| Висновок | 17 |
| Розділ 3. Розробка програмного забезпечення. | 18 |
| 3.1. Опис програмного коду. | 18 |
| 3.2. Огляд інтерфейсу програми. | 26 |
| Висновки | 28 |
| Список використаних джерел | 29 |
| Додаток А | 30 |

**Вступ**

Комп'ютерні ігри завжди були популярним видом розваги, здатним занурити гравця у фантастичний світ та надати йому можливість відчути себе героєм. Інтерес до створення ігор росте з кожним роком, і розробка власної гри є завданням, що вимагає великих зусиль та знань з різних областей програмування.

У цій курсовій роботі ми пропонуємо реалізувати програму під назвою "Іподром", яка втілює ідею одночасного руху декількох об'єктів з випадковою швидкістю. Головною метою гри є вибір гравцем одного з об'єктів та досягнення перемоги, отримуючи призові бали. Однак, на шляху гравця можуть виникнути непередбачені події, такі як неподолані перешкоди або травми, що роблять гру цікавішою та складнішою.

Для реалізації цієї програми ми використовуватимемо мову програмування C#. C# є потужним інструментом для розробки ігор, оскільки надає широкий спектр можливостей для створення графіки, обробки подій та реалізації логіки гри. Використовуючи цю мову програмування, ми зможемо створити гру, яка буде приваблювати користувачів своєю захоплюючою геймплейною механікою та візуальним оформленням.

У цій курсовій роботі ми будемо детально розглядати процес розробки гри "Іподром", починаючи з проектування ігрових об'єктів та їх взаємодії, а також розробки алгоритмів для випадкового руху об'єктів та виникнення подій. Ми також дослідимо можливості використання балів та призів для стимулювання гравців до досягнення кращих результатів.

Кінцевою метою цієї роботи є створення готової програми "Іподром", яка буде цікавою та захоплюючою для гравців. Вона буде демонструвати вміння використовувати мову програмування C# для створення ігор та реалізувати складні логічні механізми.

**Розділ 1. Моделювання надання життя об’єктів у комп’ютерних іграх.**

Меделювання об'єктів у комп'ютерних іграх є важливою складовою частиною розробки ігрового досвіду. Цей процес полягає в створенні віртуальних об'єктів, які мають свої властивості, поведінку та взаємодію з ігровим світом та гравцем.

Моделювання об'єктів передбачає використання різних технік інженерії програмного забезпечення та графічного дизайну. Розробники використовують 3D-моделювання, анімацію, текстури, фізичні двигуни та інші інструменти для створення реалістичних та живих об'єктів у грі.

Моделювання об'єктів також включає в себе розробку їхнього поведінки та взаємодії з іншими об'єктами у грі. Це може охоплювати програмування штучного інтелекту для керування неперсонажними персонажами, визначення фізики руху об'єктів, алгоритми штурму або ворожості, а також розробку взаємодій між об'єктами та гравцем.

Моделювання об'єктів у комп'ютерних іграх дозволяє створити віртуальний світ, який виглядає і працює так, ніби він реальний. Це дозволяє гравцям зануритися у гру, відчути її реалістичність та забезпечує глибокий іммерсивний досвід. Крім того, моделювання об'єктів дозволяє створити різноманітні сценарії, завдання та виклики, що збагачують геймплей та надають більше можливостей для взаємодії з грою.

У загальному контексті, моделювання об'єктів є важливим інструментом для розробників комп'ютерних ігор, оскільки воно допомагає створити незабутній та захоплюючий ігровий досвід для гравців.

* 1. **Фізична та математична моделі.** 
     1. **Фізичне моделювання.**

Фізичне моделювання[1] —

1) Створення [матеріальної моделі](https://www.wikiwand.com/uk/%D0%A4%D1%96%D0%B7%D0%B8%D1%87%D0%BD%D0%B0_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C), що має таку саму фізичну природу (такий самий фізичний зміст), як і дійсне явище, що вивчається на основі критеріїв геометричного, кінематичного й динамічного моделювання.

2) Відтворення на моделі й дослідження процесів, що якісно однакові з процесами у реальному об'єкті. Під час М.ф. процесу необхідно забезпечити геометричну, часову та фізичну подібності.

Фізичне моделювання є важливою складовою розробки комп'ютерних ігор і відіграє вирішальну роль у створенні реалістичної фізики та поведінки об'єктів у грі. Цей процес використовує математичні алгоритми та фізичні принципи для симуляції руху, столкнувшись з силами та іншими фізичними явищами.

Одним з основних аспектів фізичного моделювання є симуляція руху об'єктів. Використовуючи закони Ньютона, можна розрахувати швидкість, прискорення та силу, що діють на об'єкт. Це дозволяє створити реалістичну динаміку руху об'єктів, враховуючи такі фактори, як гравітація, тертя, вплив інших об'єктів та так далі.

Крім руху, фізичне моделювання також охоплює столкнувшись з об'єктами та взаємодію з ними. Розробники використовують колізії та детекцію столкнувшись, щоб визначити, коли та як об'єкти зіштовхуються один з одним. Це дозволяє створити реалістичні ефекти столкнувшись, такі як зміна траєкторії, розбиття, руйнування та інші[2].

Фізичне моделювання також може включати симуляцію рідини, вогню, диму, твердих тіл та інших фізичних явищ. Використовуючи відповідні алгоритми та розрахунки, розробники можуть створювати реалістичні та динамічні ефекти, які покращують іммерсивність гри та додають більше реалізму.

Фізичне моделювання важливо для створення гри, яка відтворює реальний світ або фантастичну реальність з віртуальними законами фізики. Воно допомагає гравцям відчути реалістичність у своїх діях та взаємодії з об'єктами, а також відкриває безліч можливостей для геймплею та пазлових ситуацій.

У розробці комп'ютерних ігор фізичне моделювання використовується з різним рівнем складності, залежно від потреб і особливостей самої гри. Від простих алгоритмів руху та столкнувшись з базовими формами до складних симуляцій реалістичної фізики, фізичне моделювання є потужним інструментом для покращення ігрового досвіду та взаємодії гравців з віртуальним світом.

У цій роботі можуть бути використані основні закони Ньютона для моделювання руху об'єктів. Ось декілька законів Ньютона, які можуть бути застосовані:

1 Перший закон Ньютона (Закон інерції): Об'єкт залишається у стані спокою або руху прямолінійним рівномірним рухом, якщо на нього не діють зовнішні сили. Цей закон може бути використаний для моделювання об'єктів, які зупиняються або продовжують рухатися без зовнішніх впливів.

2 Другий закон Ньютона (Закон про рух імпульсів): Сила, що діє на об'єкт, пропорційна швидкості зміни імпульсу цього об'єкта. Цей закон може бути використаний для обчислення прискорення об'єкту, залежно від сили, що на нього діє.

3 Третій закон Ньютона (Закон взаємодії): Кожна дія має протилежну реакцію. Цей закон може бути використаний для моделювання взаємодії між об'єктами, наприклад, при столкнувшись зі штучним інтелектом або іншими об'єктами на іподромі.

Ці закони Ньютона є базовими принципами фізики руху та дозволяють моделювати рух об'єктів з використанням сил, прискорення і імпульсу. Застосування цих законів допоможе створити більш реалістичну та динамічну поведінку об'єктів у програмі "Іподром".

* + 1. **Математичне моделювання.**

Математичне моделювання в цій роботі використовується для симуляції руху та взаємодії об'єктів на іподромі. Це означає використання математичних алгоритмів та формул для обчислення руху, столкнувшись з силами та іншими фізичними явищами[3].

У цьому контексті можна використовувати різні математичні концепції та методи. Ось декілька прикладів:

1 Кінематика: Математичні відношення для обчислення шляху, швидкості та прискорення об'єктів. За допомогою формул кінематики можна визначити, як об'єкти будуть рухатися на іподромі.

2 Колізії: Використання геометричних алгоритмів та формул для виявлення столкнувшись з об'єктами. Це дозволяє визначити, коли та як об'єкти перетинаються, і виконувати необхідні обчислення для їх взаємодії.

3 Фізичні сили: Використання законів фізики, таких як закони Ньютона, для обчислення сил, що діють на об'єкти. Це дозволяє враховувати гравітацію, тертя та інші впливи на рух об'єктів.

4 Імпульс та енергія: Використання розрахунків імпульсу та енергії для визначення взаємодії об'єктів під час столкнувшись. Це допомагає враховувати зміну швидкості та напрямку руху після зіткнення.

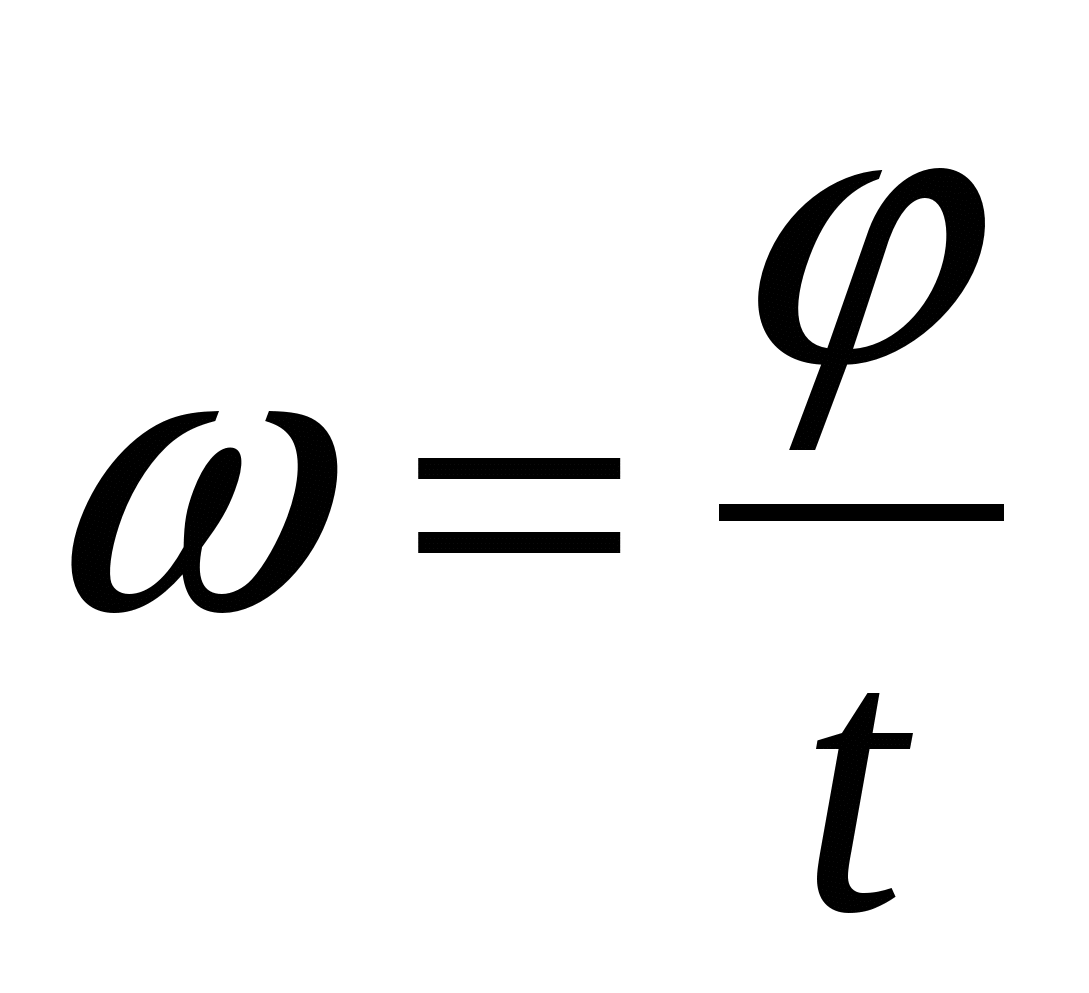
Математичне моделювання є важливим елементом розробки програми "Іподром", оскільки воно дозволяє точно відтворити фізичні властивості та поведінку об'єктів. Це створює реалістичну та захопливу геймплейну ситуацію для гравців та додає глибини до ігрового досвіду.

* 1. **Рух об’єктів по колу. Полярні координати. Кутова швидкість.**

У контексті даної роботи, програма "Іподром", рух об'єкту по колу в полярних координатах може бути використаний для моделювання траєкторії руху коней на іподромі. Кожен кінь може мати свою власну кутову швидкість (ω) та радіус (r), що впливають на його рух.

У програмі "Іподром", кутова швидкість використовується для визначення швидкості обертання об'єкту навколо центру кола. Це дозволяє кіню на іподромі рухатися вздовж кола, змінюючи своє напрямок та положення на основі цього параметра.

Кутова швидкість (ω) вимірюється у радіанах на одиницю часу. Якщо кінь має позитивну кутову швидкість, то він буде обертатися проти годинникової стрілки навколо центру кола. У випадку від'ємної кутової швидкості, кінь буде рухатися за годинниковою стрілкою навколо центру кола[4].



Де ð - Кут повороту, t - час повороту

Кутова швидкість може бути випадковою для кожного коня на іподромі або заданою в рамках логіки гри. Використовуючи це значення, програма "Іподром" розраховує новий кут кожного коня в кожному кадрі або визначеному проміжку часу. Залежно від кутової швидкості, кінь змінює своє положення на колі, дозволяючи йому рухатися по траєкторії кола.

Кутова швидкість впливає на швидкість руху коня навколо кола. Чим більше значення кутової швидкості, тим швидше об'єкт буде обертатися. Це дозволяє кіню на іподромі рухатися швидше або повільніше в залежності від цього параметра.

У програмі "Іподром" використовуються полярні координати для визначення положення об'єктів на іподромі, зокрема для визначення їх положення на колі.

Полярні координати складаються з двох основних компонентів: радіусу (r) та кута (θ). Радіус вказує на відстань від початку координат (центру кола) до точки, яка визначає положення об'єкта на колі. Кут (θ) визначає напрямок та орієнтацію об'єкта відносно центру кола[5].

У випадку програми "Іподром", полярні координати використовуються для представлення положення кожного коня на колі. Радіус вказує на відстань між центром кола та конем, тоді як кут вказує на напрямок, в якому конь рухається.

Застосовуючи ці параметри, програма "Іподром" може визначити положення кожного коня на колі в кожен момент часу, оновлюючи їх координати на основі їх кутової швидкості та радіуса. Це дозволяє створити реалістичну симуляцію руху коней, їх обертання навколо центру кола та взаємодію з іншими об'єктами на іподромі.

Такий підхід до моделювання руху об'єкту по колу в полярних координатах додає реалістичності та варіативності до геймплею програми "Іподром", забезпечуючи цікавий та унікальний досвід гравцям.

**Розділ 2. Концепції та середовища, які були використані в розробці програмного забезпечення.**

C# (C Sharp) - це об'єктно-орієнтована мова програмування, яка часто використовується для розробки різноманітних додатків, включаючи комп'ютерні ігри. C# є однією з основних мов програмування для платформи .NET, що робить її ідеальним вибором для реалізації програми "Іподром".

Visual Studio - це інтегроване середовище розробки (IDE), розроблене компанією Microsoft, яке надає потужні інструменти для розробки, налагодження та тестування програм на мові C# та інших мовах програмування. Visual Studio забезпечує зручний інтерфейс користувача, автоматичне завершення коду, відладку, підсвічування синтаксису та багато інших функцій, що спрощують процес розробки програми.

Використовуючи Visual Studio і мову програмування C#, ми зможемо легко створити програму "Іподром". Ми можемо створити новий проект у Visual Studio та вибрати шаблон для розробки десктопної програми на C#. Після цього ви можете розпочати написання коду програми, використовуючи мову C# та вбудовані бібліотеки .NET Framework для реалізації функціональності програми "Іподром".

Завдяки потужному поєднанню мови програмування C# та середовища Visual Studio, ми зможемо зробити програму "Іподром" більш модульною, додати взаємодію з користувачем, використати вбудовані засоби візуалізації та налагодження коду. Visual Studio також надає інтегровану систему керування версіями, що дозволяє легко відстежувати та управляти змінами у нашому проекті.

Отже, використання мови програмування C# та середовища Visual Studio забезпечує нам потужні інструменти для розробки програми "Іподром" зручно та ефективно.

**2.1. Поліморфізм.**

Поліморфізм є одним з основних принципів об'єктно-орієнтованого програмування, який дозволяє об'єктам різних класів виконувати однакові дії, використовуючи одну і ту саму інтерфейсну або базову класову структуру. Це означає, що об'єкти можуть вести себе по-різному, в залежності від свого конкретного типу, але однаково виконувати спільні операції або методи[6].

У контексті програми "Іподром" поліморфізм можна використати для реалізації різних видів об'єктів, таких як коні на іподромі. Наприклад, у програмі можуть бути різні класи коней, які успадковують спільний інтерфейсний клас або абстрактний клас. Цей спільний інтерфейсний клас містить загальні методи або властивості, які всі коні повинні реалізувати.

Завдяки поліморфізму, ми можемо створити колекцію об'єктів типу коня, але реальні об'єкти в цій колекції можуть бути різних підтипів класу коня. Наприклад, у вас можуть бути об'єкти класу "Трійка", "Гончак" і "Вершник", які успадковують спільний клас "Кінь". Завдяки поліморфізму, ми можемо викликати спільний метод, наприклад, "рухатися", для кожного об'єкта в колекції, і кожен об'єкт поведе себе по-різному, виконуючи свою унікальну реалізацію методу.

Це дає нам гнучкість і розширюваність у програмі "Іподром". Ми можемо легко додавати нові типи коней, створюючи нові підкласи, і вони будуть автоматично використовувати загальний інтерфейс для виконання спільних дій. Поліморфізм допомагає нам зробити код більш загальним та зручним для підтримки змін та розширення в програмі "Іподром".

Приклади поліморфізму:

В мові програмування C# поліморфізм реалізується за допомогою використання віртуальних методів та перевизначення методів.

• Віртуальні методи: Вони оголошуються в базовому класі з використанням ключового слова virtual і можуть бути перевизначені в похідних класах. Коли метод викликається через посилання на базовий клас, виконується реалізація методу з похідного класу.

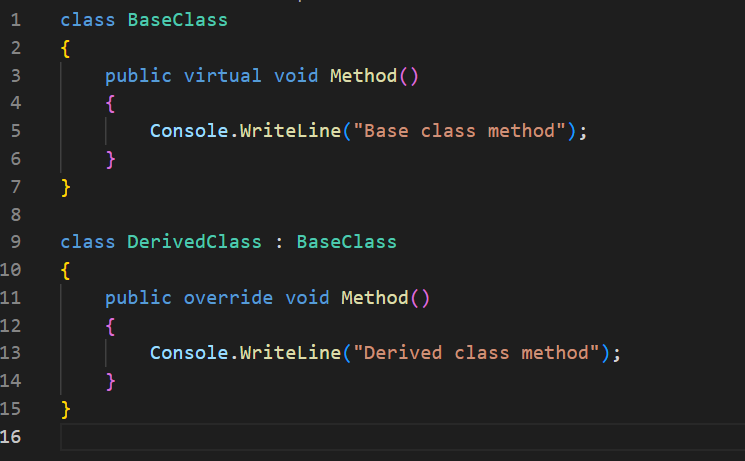


Рисунок 2.1. Віртуальні методи

• Перевизначення методів: У похідному класі метод, оголошений з використанням ключового слова override, заміщує віртуальний метод базового класу та надає нову реалізацію для нього.

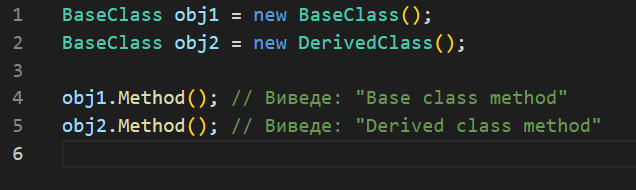


Рисунок 2.2. Приклад використання поліморфізму

У цьому прикладі метод Method() викликається через об'єкт obj1 базового класу і виконується його реалізація. Але коли метод викликається через об'єкт obj2 похідного класу, виконується реалізація методу з похідного класу.

**2.2. Інкапсуляція.**

Інкапсуляція є одним з основних принципів об'єктно-орієнтованого програмування, який дозволяє об'єднувати дані та методи, що з ними пов'язані, в єдину одиницю - об'єкт. Інкапсуляція надає можливість приховувати деталі реалізації об'єкта від зовнішнього світу і надає доступ до них лише через публічний інтерфейс[7].

У контексті програми "Іподром", інкапсуляція може бути використана для організації класів та даних, пов'язаних з об'єктами на іподромі. Наприклад, у вас може бути клас "Кінь", який містить приватні поля, такі як "ім'я", "швидкість" та "положення". Ці дані можуть бути доступні лише в межах самого класу "Кінь" через його методи та властивості.

Інкапсуляція дозволяє захистити дані об'єкта від неправильного використання та недозволених змін. Зовнішній код, що використовує об'єкти класу "Кінь", може отримувати доступ до цих даних лише через публічні методи, які надають контрольований доступ до них. Наприклад, можна мати публічний метод "отриматиІм'я", який повертає значення приватного поля "ім'я", але не дозволяє змінювати його напряму.

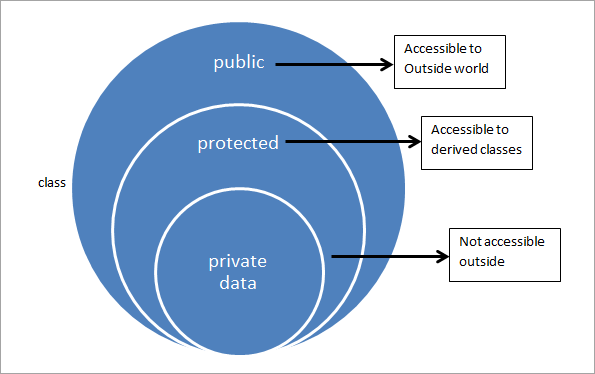


Рисунок 2.3. Приховування даних.

Крім того, інкапсуляція дозволяє змінювати внутрішню реалізацію класу "Кінь" без впливу на зовнішній код, що використовує цей клас. Наприклад, якщо ви вирішите змінити спосіб збереження положення коня на іподромі з полярних координат на декартові, вам потрібно буде змінити лише внутрішню реалізацію класу "Кінь", а зовнішній код, який використовує клас "Кінь", залишиться незмінним.

Інкапсуляція допомагає забезпечити чистоту коду, розділення відповідальностей та полегшує утримання та розширення програми "Іподром". Вона дозволяє контролювати доступ до даних та методів, забезпечуючи більшу безпеку та надійність програми.

**2.3. Динамічна пам’ять.**

Динамічна пам'ять - це область пам'яті в комп'ютері, яка використовується для виділення та звільнення пам'яті під час виконання програми. У контексті програмування, особливо в мові C#, динамічна пам'ять дозволяє програмі звертатися до пам'яті у більш гнучкий та динамічний спосіб[8].

В мові C#, для роботи з динамічною пам'яттю, використовується концепція посилань (references) і операторів new та delete (або garbage collector, який автоматично звільняє невикористовувану пам'ять).

Користування динамічною пам'яттю особливо корисне в ситуаціях, коли розмір або кількість об'єктів, які потрібно створити, визначаються під час виконання програми. Наприклад, у програмі "Іподром" можуть бути об'єкти коней, які створюються динамічно в залежності від вибору гравця чи генерування подій.

За допомогою динамічної пам'яті ви можете створювати нові об'єкти в режимі виконання, виділяючи для них необхідну пам'ять, та працювати з ними. Після використання об'єкти можуть бути звільнені з пам'яті, щоб уникнути витоку пам'яті та оптимізувати використання ресурсів.

Використання динамічної пам'яті дозволяє гнучко керувати пам'яттю в програмі "Іподром" та динамічно створювати та звільняти об'єкти за потребою. Це допомагає ефективно використовувати ресурси системи та покращує продуктивність програми. Однак, важливо пам'ятати про правильне використання та звільнення пам'яті, щоб уникнути проблем з витоком пам'яті або некоректною роботою програми.

**2.4. Спадкування.**

Спадкування є важливим принципом об'єктно-орієнтованого програмування, який дозволяє створювати ієрархію класів і перевикористовувати код та функціональність. При спадкуванні один клас, відомий як підклас або похідний клас, успадковує властивості та методи від іншого класу, відомого як базовий клас або батьківський клас[9].

У контексті програми "Іподром", спадкування може бути використане для організації класів, що представляють різні типи об'єктів на іподромі. Наприклад, можуть бути базовий клас "Тварина" і похідні класи "Кінь", "Собака" тощо. В базовому класі можуть бути визначені загальні властивості та методи, такі як "ім'я" чи "рухатися", які будуть успадковуватися всіма похідними класами. Крім того, похідні класи можуть додатково розширювати функціональність базового класу, додаючи нові методи або властивості, які є специфічними для конкретного типу тварини.

Спадкування дозволяє ефективно використовувати код, перевикористовуючи функціональність базового класу без необхідності повторного введення коду для кожного похідного класу. Воно сприяє поліпшенню структури та організації програми, спрощує розробку та підтримку коду.

Додатковою перевагою спадкування є можливість поліморфізму. Об'єкти класу-потомка можуть бути використані там, де вимагаються об'єкти базового класу, що сприяє гнучкості та розширюваності програми.

Проте, важливо враховувати, що спадкування пов'язує класи між собою, тому зміни в базовому класі можуть вплинути на всі похідні класи. Правильне проектування ієрархії класів та розумна організація спадкування допоможуть уникнути зайвої складності та непотрібних залежностей між класами[10].

**Висновок**

У даній частині курсової роботи ми розглянули мову програмування C# та середовище Visual Studio, які є потужними інструментами для розробки програмного забезпечення. Мова програмування C# дозволяє створювати структурований та об'єктно-орієнтований код, що дозволяє нам ефективно реалізувати програму "Іподром".

Visual Studio, у свою чергу, надає нам зручне інтегроване середовище розробки (IDE), яке спрощує процес написання, налагодження та тестування програми. Воно має багато корисних функцій, таких як автодоповнення коду, інструменти візуального проектування і налагодження, що полегшують розробку програмного забезпечення.

Крім того, ми ознайомились з основними концепціями об'єктно-орієнтованого програмування, такими як поліморфізм, інкапсуляція, спадкування та динамічна пам'ять. Ці концепції дозволяють нам створювати більш структурований, гнучкий та перевикористовуваний код, що є важливими аспектами розробки програмного забезпечення.

Загальною метою цієї частини роботи було ознайомлення з необхідними інструментами та концепціями, що допомагають нам ефективно розробляти програму "Іподром". Надані мова програмування C#, середовище Visual Studio та об'єктно-орієнтовані концепції виявляються потужними засобами для досягнення цієї мети.

**Розділ 3. Розробка програмного забезпечення.**

Розробити програму, яка реалізує одночасний рух декількох об'єктів з випадковою швидкістю. Можна передбачити випадкову появу інших подій (неподолана перешкода, травма і т.ін.). На початку гри гравець обирає об'єкт, та, якщо він виграє, отримує призові бали.

**3.1. Опис програмного коду.**

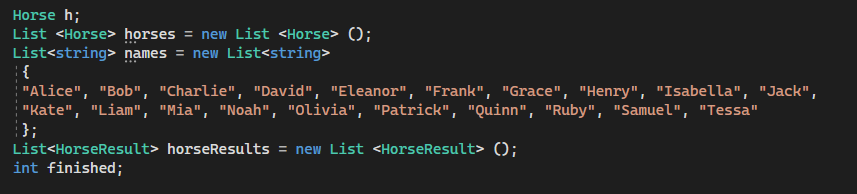
****

Рисунок 3.1. Фрагмент коду створює необхідні змінні та списки для подальшої роботи з об'єктами типу Horse.

Цей фрагмент коду програми "Іподром" починається з оголошення декількох змінних та створення необхідних списків.

• Змінна h використовується для створення об'єкту типу Horse, який представляє кіней на іподромі. Вона поки що не має присвоєної значення.

• Список horses створюється з використанням типу List<Horse>. Він призначений для зберігання об'єктів типу Horse. У даному випадку, список ініціалізується порожнім.

• Список names є списком рядків типу List<string>. Він містить список імен для кіней на іподромі. Імена заповнені заздалегідь заданими значеннями.

• Список horseResults є списком об'єктів типу HorseResult. Його призначення полягає у зберіганні результатів змагань кіней.

• Змінна finished є цілочисельною змінною та відображає кількість кіней, які закінчили змагання.

Цей фрагмент коду створює необхідні змінні та списки для подальшої роботи з об'єктами типу Horse, зберігання їхніх результатів та інформації про них. Далі в програмі можуть бути використані ці змінні та списки для додавання нових кіней, обробки результатів змагань та подібних операцій, пов'язаних з роботою іподрому.

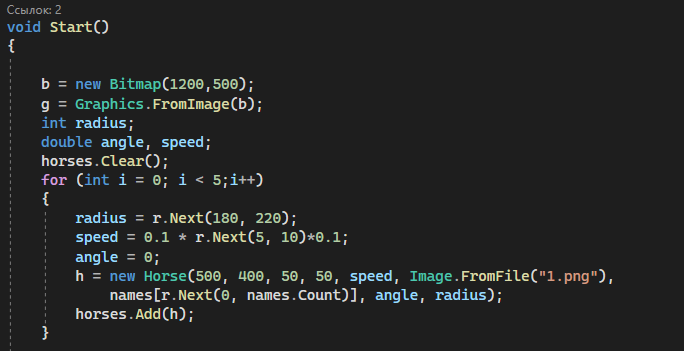


Рисунок 3.2. Фрагмент коду створює та ініціалізує об'єкти Horse з випадковими значеннями для радіусу

Функція Start() в даному фрагменті коду виконує певні дії для початку гри на іподромі. Основний функціонал цієї функції описаний нижче:

• Створюється новий об'єкт Bitmap з розмірами 1200 на 500 пікселів. Цей об'єкт Bitmap представляє зображення, на якому будуть відображатись об'єкти гри.

• Створюється об'єкт Graphics з цього зображення. Він дозволяє виконувати операції малювання та рендерингу на зображенні.

• Змінна radius отримує випадкове значення в діапазоні від 180 до 220. Це визначає радіус кола, по якому будуть рухатись кіні.

• Змінна speed отримує випадкове значення в діапазоні від 0.5 до 0.9. Це визначає швидкість руху кіней.

• Змінна angle ініціалізується значенням 0. Вона представляє кут, під яким кінь буде рухатись по колу.

• Створюється новий об'єкт типу Horse з використанням отриманих значень radius, speed, angle, а також інших параметрів, таких як координати та розміри.

• Об'єкт Horse додається до списку horses.

Цей фрагмент коду створює та ініціалізує об'єкти Horse з випадковими значеннями для радіусу, швидкості та інших параметрів. Це дозволяє створити декілька кіней з різними властивостями та початковими умовами для подальшого руху по колу на іподромі.

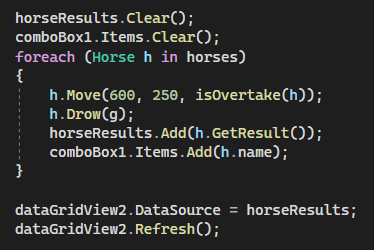


Рисунок 3.3. У цьому фрагменті коду розглядається обробка руху кіней на іподромі, оновлення візуалізації та оновлення даних в таблиці.

• horseResults.Clear() очищає список horseResults, що зберігає результати змагань кіней.

• comboBox1.Items.Clear() очищає вміст елементів у comboBox1. Це виконується для підготовки до заповнення комбінованого списку з іменами кіней.

• В циклі foreach для кожного об'єкта типу Horse у списку horses виконується наступне:

• Викликається метод Move, який відповідає за рух кіней. Метод отримує цільові координати (600, 250) та значення isOvertake(h), що вказує, чи потрібно виконувати обгін для даного кінька.

• Викликається метод Drow, який відповідає за візуалізацію кіней на графічному контексті g.

• Результати кіней додаються до списку horseResults.

• Імена кіней додаються до комбінованого списку comboBox1.

• dataGridView2.DataSource = horseResults прив'язує список horseResults до даних таблиці dataGridView2. Це дозволяє відображати результати змагань кіней у вигляді таблиці.

• dataGridView2.Refresh() оновлює відображення таблиці dataGridView2 з новими даними.

Цей фрагмент коду виконує процес оновлення руху кіней на іподромі, оновлення візуального представлення та оновлення даних таблиці з результатами змагань.

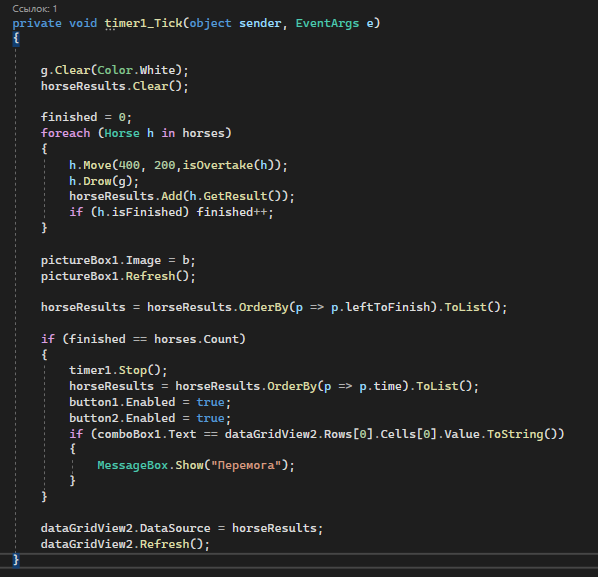


Рисунок 3.4. Цей фрагмент коду описує подію timer1\_Tick, яка виконується при кожному тіку таймера.

В цій події відбувається оновлення руху кіней, візуалізація, оновлення даних таблиці та перевірка на закінчення змагання.

• g.Clear(Color.White) очищає графічний контекст g шляхом заповнення його білим кольором. Це дозволяє очистити попередні візуальні елементи, щоб малювати новий стан.

• horseResults.Clear() очищає список horseResults, що зберігає результати змагань кіней.

• Змінна finished ініціалізується значенням 0. Вона використовується для підрахунку кількості закінчених змагань.

• В циклі foreach для кожного об'єкта типу Horse у списку horses виконується наступне:

• Викликається метод Move, який відповідає за рух кіней. Метод отримує цільові координати (400, 200) та значення isOvertake(h), що вказує, чи потрібно виконувати обгін для даного кінька.

• Викликається метод Drow, який відповідає за візуалізацію кіней на графічному контексті g.

• Результати кіней додаються до списку horseResults.

• Якщо кінь закінчив змагання (h.isFinished), збільшується лічильник закінчених змагань finished.

• pictureBox1.Image = b оновлює зображення на pictureBox1 з використанням зображення b, яке містить візуалізований стан кіней.

• pictureBox1.Refresh() оновлює відображення pictureBox1.

• horseResults = horseResults.OrderBy(p => p.leftToFinish).ToList() сортує список horseResults за залишком відстані до фінішу (leftToFinish) в порядку зростання.

• Умова if (finished == horses.Count) перевіряє, чи всі кіні закінчили змагання. Якщо так, виконується наступне:

• Зупиняється таймер (timer1.Stop()).

• Сортується список horseResults за часом (time) в порядку зростання.

• Активуються кнопки button1 і button2.

• Перевіряється, чи обраний кінь (comboBox1.Text) співпадає з першим рядком в стовпці "Name" (dataGridView2.Rows[0].Cells[0].Value.ToString()) таблиці dataGridView2. Якщо так, виводиться повідомлення "Перемога" (MessageBox.Show("Перемога")).

• dataGridView2.DataSource = horseResults прив'язує список horseResults до даних таблиці dataGridView2. Це оновлює відображення результатів змагань у таблиці.

• dataGridView2.Refresh() оновлює відображення таблиці dataGridView2 з новими даними.

Цей фрагмент коду виконує оновлення руху кіней, візуалізацію, оновлення даних таблиці та перевірку на закінчення змагання при кожному тіку таймера.

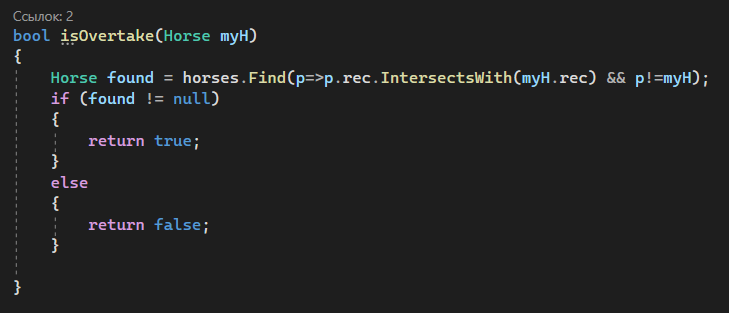


Рисунок 3.5. У цьому фрагменті коду визначена функція isOvertake, яка перевіряє, чи відбувається обгін між конкретним кінем myH та іншими кіньми у списку horses.

• Використовується метод Find для пошуку першого кінька found у списку horses, для якого виконується умова:

• Прямокутники rec (припущення: вони представлені як об'єкти типу Rectangle) кіньків myH та p перетинаються (IntersectsWith).

• Кінь p не є тим самим кіньком, що і myH (p != myH).

• Якщо знайдено такий кінь found, виконується наступне:

• Повертається значення true, що вказує на те, що відбувається обгін.

• Якщо ж жодного кінька не знайдено, виконується наступне:

• Повертається значення false, що вказує на відсутність обгону.

Ця функція допомагає визначати, чи відбувається обгін між конкретним кінем myH та іншими кіньми на іподромі.

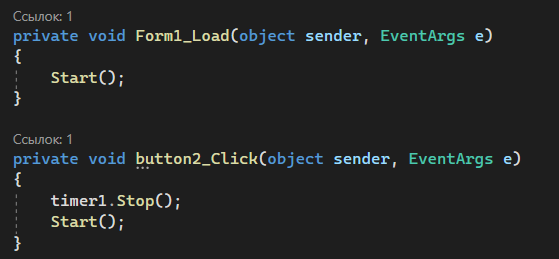


Рисунок 3.6. У цьому фрагменті коду описані дві події, пов'язані з формою (Form1).

• Form1\_Load: Ця подія виникає при завантаженні форми. У цьому випадку, коли форма завантажується, викликається метод Start(). Це означає, що при відкритті форми починається змагання на іподромі.

• button2\_Click: Ця подія виникає при натисканні на кнопку з ім'ям button2. У цьому випадку, коли кнопка натиснута, таймер timer1 зупиняється за допомогою timer1.Stop(), а потім викликається метод Start(). Це дозволяє зупинити поточне змагання та розпочати нове, коли користувач натискає кнопку.

Ці події забезпечують ініціалізацію та рестарт змагання на іподромі відповідно до потреб користувача.

**3.2. Огляд інтерфейсу програми.**

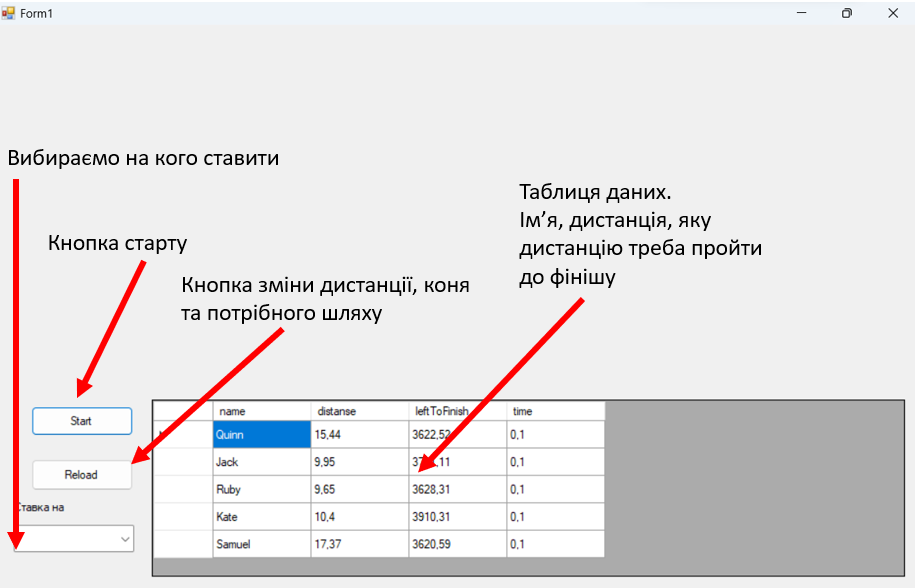


Рисунок 3.7. Огляд основних частин інтерфейсу програми.

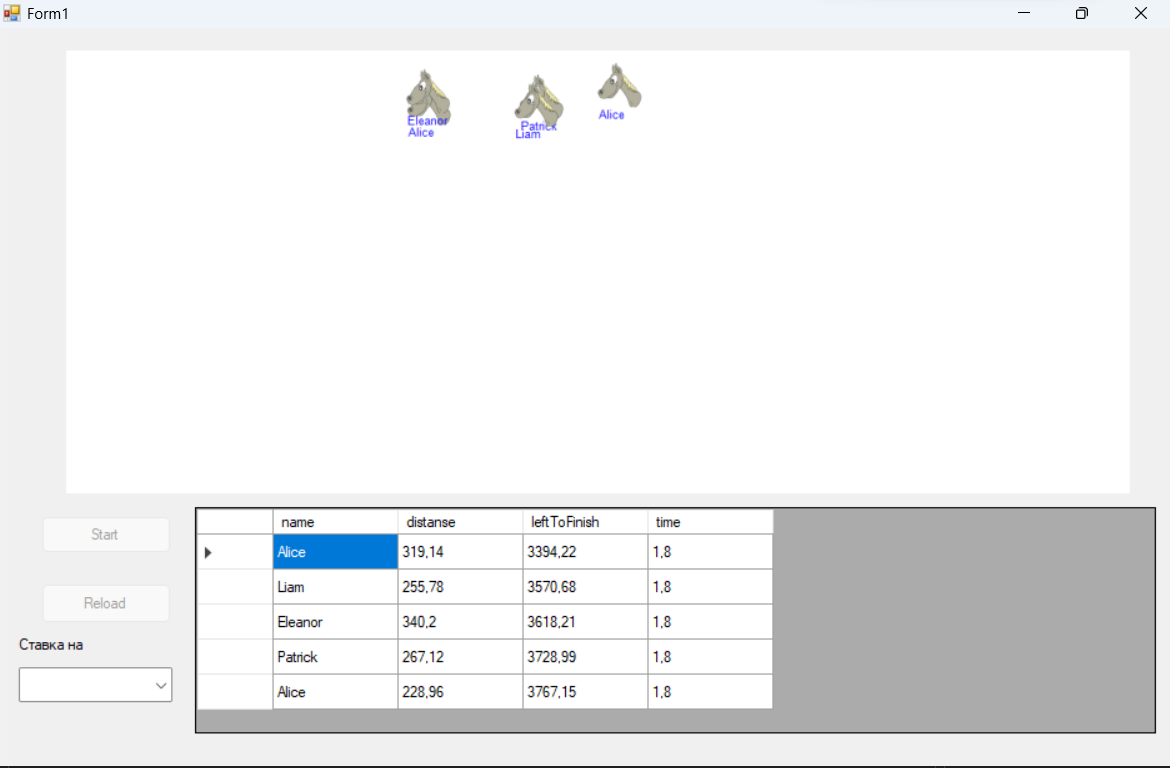


Рисунок 3.8. Початок гри.

Коли ми починаємо гру, ми маємо коня на якого ми зробили ставку. Коні починають рухатись по колу. У таблиці даних у реальному часі показуються зміни пройденої дистанції, залишку до фінішна, часу, який було затрачено.

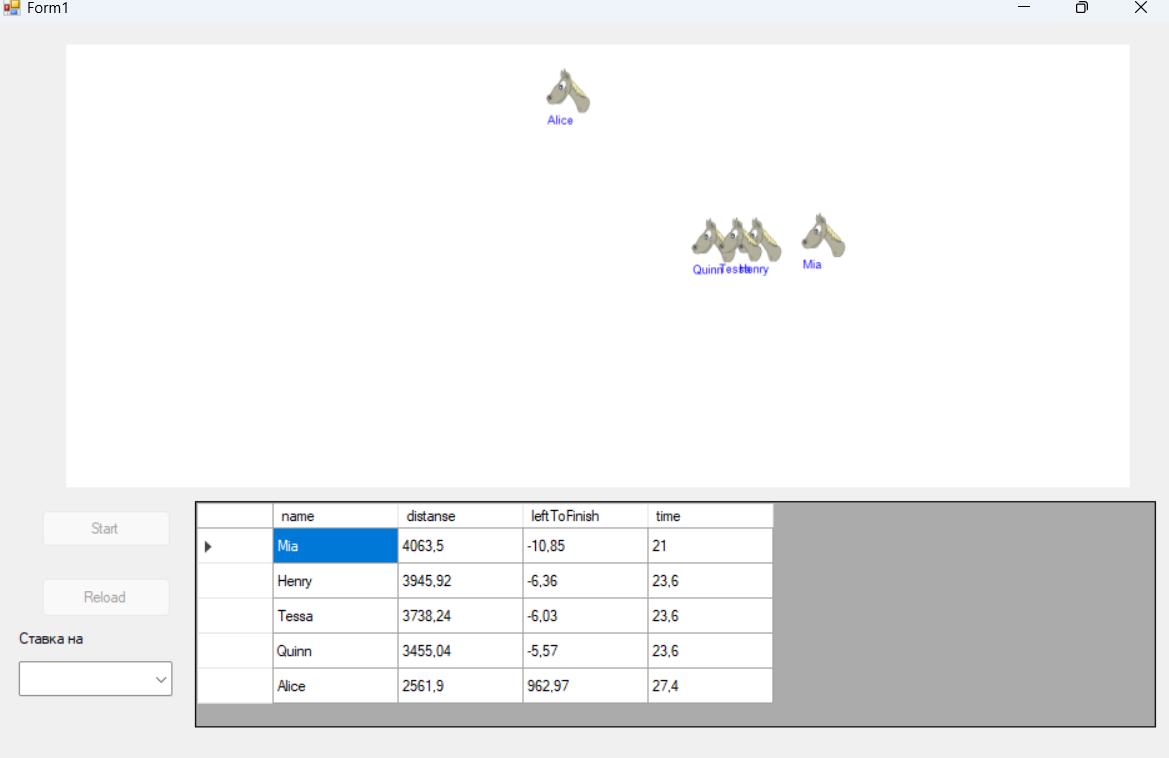


Рисунок 3.9. Кінець гри.

Кінець гри наступає коли всі коні пройшли свою дистанцію. Коли кінь пройде всю дистанцію, він зупиняється.

**Висновки**

У цій курсовій роботі була розроблена програма "Іподром", яка реалізує одночасний рух декількох об'єктів з випадковою швидкістю. Програма дозволяє відтворити ситуації, що виникають під час змагань на іподромі, включаючи неподолані перешкоди та можливі травми.

Мова програмування C# та середовище Visual Studio були використані для написання програми. C# є потужною та простою в освоєнні мовою програмування, яка надає широкі можливості для розробки різноманітних додатків. Visual Studio забезпечує зручне середовище для розробки, засоби налагодження та підтримку програміста під час написання коду.

У процесі розробки програми були використані різні концепції програмування, такі як поліморфізм, інкапсуляція, динамічна пам'ять та спадкування. Ці концепції дозволяють покращити структуру програми, зробити її більш модульною, розширюваною та легко зрозумілою.

Рух об'єктів по колу у програмі був моделюваний за допомогою полярних координат. Використовуючи кутову швидкість та радіус кола, об'єкти змагалися один з одним, спробуючи досягти фінішної прямої.

Курсова робота демонструє важливі аспекти розробки програмного забезпечення, включаючи використання мови програмування, роботу зі змінними, функціями, класами, графічним інтерфейсом користувача та різними концепціями програмування. Вона надає можливість практично застосувати отримані знання та навички для створення функціональної та цікавої програми.

В цілому, ця курсова робота дозволила поглибити розуміння розробки програмного забезпечення, а також набути практичний досвід у використанні мови програмування C#, середовища Visual Studio та використанні різних концепцій програмування.

**Список використаних джерел**

1 Eberly, D. (2012). Game Physics (2nd Edition). CRC Press.

2 Lengyel, E. (2012). Mathematics for 3D Game Programming and Computer Graphics (3rd Edition). CRC Press.

3 Baraff, D., & Witkin, A. (1998). Large Steps in Cloth Simulation. ACM SIGGRAPH Computer Graphics, 22(4), 43-54.

4 Sauer, T. (2013). Numerical Analysis (2nd Edition). Pearson.

5 Gewali, P., & Nourani, Y. (Eds.). (2018). Computational Science and Its Applications – ICCSA 2018. Springer.

6 Albahari, J., Albahari, B., & Albahari, E. (2012). C# 5.0 in a Nutshell: The Definitive Reference. O'Reilly Media.

7 Liberty, J., MacDonald, B., & Hoffman, M. (2015). Learning C# by Developing Games with Unity. Packt Publishing.

8 Lippert, E. (2019). C# Pocket Reference: Instant Help for C# Programmers. O'Reilly Media.

9 Schildt, H. (2017). C# 7.0: The Complete Reference. McGraw-Hill Education.

10 Troelsen, A., & Japikse, P. (2017). Pro C# 7: With .NET and .NET Core. Apress.

Додаток А

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Hippodrome

{

public partial class Form1 : Form

{

Horse h;

List <Horse> horses = new List <Horse> ();

List<string> names = new List<string>

{

"Alice", "Bob", "Charlie", "David", "Eleanor", "Frank", "Grace", "Henry", "Isabella", "Jack",

"Kate", "Liam", "Mia", "Noah", "Olivia", "Patrick", "Quinn", "Ruby", "Samuel", "Tessa"

};

List<HorseResult> horseResults = new List <HorseResult> ();

int finished;

public Form1()

{

InitializeComponent();

}

Bitmap b;

Graphics g;

Random r = new Random();

void Start()

{

b = new Bitmap(1200,500);

g = Graphics.FromImage(b);

int radius;

double angle, speed;

horses.Clear();

for (int i = 0; i < 5;i++)

{

radius = r.Next(180, 220);

speed = 0.1 \* r.Next(5, 10)\*0.1;

angle = 0;

h = new Horse(500, 400, 50, 50, speed, Image.FromFile("1.png"),

names[r.Next(0, names.Count)], angle, radius);

horses.Add(h);

}

horseResults.Clear();

comboBox1.Items.Clear();

foreach (Horse h in horses)

{

h.Move(600, 250, isOvertake(h));

h.Drow(g);

horseResults.Add(h.GetResult());

comboBox1.Items.Add(h.name);

}

dataGridView2.DataSource = horseResults;

dataGridView2.Refresh();

}

private void button1\_Click(object sender, EventArgs e)

{

button1.Enabled = false;

button2.Enabled = false;

timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

g.Clear(Color.White);

horseResults.Clear();

finished = 0;

foreach (Horse h in horses)

{

h.Move(400, 200,isOvertake(h));

h.Drow(g);

horseResults.Add(h.GetResult());

if (h.isFinished) finished++;

}

pictureBox1.Image = b;

pictureBox1.Refresh();

horseResults = horseResults.OrderBy(p => p.leftToFinish).ToList();

if (finished == horses.Count)

{

timer1.Stop();

horseResults = horseResults.OrderBy(p => p.time).ToList();

button1.Enabled = true;

button2.Enabled = true;

if (comboBox1.Text == dataGridView2.Rows[0].Cells[0].Value.ToString())

{

MessageBox.Show("Перемога");

}

}

dataGridView2.DataSource = horseResults;

dataGridView2.Refresh();

}

bool isOvertake(Horse myH)

{

Horse found = horses.Find(p=>p.rec.IntersectsWith(myH.rec) && p!=myH);

if (found != null)

{

return true;

}

else

{

return false;

}

}

private void Form1\_Load(object sender, EventArgs e)

{

Start();

}

private void button2\_Click(object sender, EventArgs e)

{

timer1.Stop();

Start();

}

}

}