КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки
на тему:

ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ В КОМП'ЮТЕРНИХ ІГРАХ

Виконала студентка 4-го курсу

Пономаренко Наталія Олександрівна

(підпис)

Науковий керівник:

професор, доктор фіз.-мат. наук

Пашко Анатолій Олексійович (підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики

«____» _____ 2023 р.,

протокол No ___

доктор фіз.-мат. наук, професор

Ю.В. Крак (підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 60 сторінок, 18 ілюстрацій, 20 джерела посилань, 8 додатків.

Об’єктом роботи є створення гри з генетичним алгоритмом. Предметом роботи є реалізація гри за допомогою генетичного алгоритму та його реалізацією за допомогою робототехніки.

Метою роботи є отримання в результаті авторської гри-аркади з цікавими правилами та анімованим оформленням, при цьому основна увага буде приділятися розробці штучного ігрового інтелекту.

Інструменти розроблення: інтегроване середовище розробки: Embarcadero Rad Studio 10 Berlin Delphi, Arduino IDE.

Результати роботи: створення гри типу аркада для мобільного пристрою з персонажем, що рухається між перешкодами до цілі за допомогою генетичного алгоритму. Також було запропоновано та показано, як простими засобами гра може бути переформатована у настільну розвагу.

Розроблене програмне забезпечення може використовуватись навчальними закладами середньої та вищої освіти. Також створений проект може бути цікавий в комерційних проектах, як продукт для розваг і в навчальних або розвиваючих цілях.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки:

Всі IT-сфери зробили великий стрибок у розвитку, при цьому така ж тенденція спостерігається і в ігровій індустрії. На початку були створені найпростіші ігри на зразок Pac-Man і вони дозволили забувати гравцям про реальність на тривалий час. Сучасні проекти по створенню ігор вже стали повноцінним хобі, настільки поширеним, що сьогодні кожен користувач мобільних пристроїв хоч раз спробував ігри зі схожими правилами. При цьому важливу роль в привабливості ігор має ігровий штучний інтелект.

Актуальність роботи:

Розробка гри за допомогою генетичного алгоритму та використання робототехніки є актуальною темою з двох основних причин.

По-перше, це відкриває нові можливості для створення унікальних та цікавих ігрових досвідів. Генетичний алгоритм дозволяє автоматизувати процес створення та оптимізації геймплею, персонажів та правил гри. Це дозволяє розробникам експериментувати з безліччю варіантів та швидше досягати оптимальних результатів. Додавання робототехніки дозволяє перенести гру у фізичний світ, створюючи інтерактивне середовище та додаткові можливості для взаємодії гравців.

По-друге, такий підхід є важливим для розвитку індустрії робототехніки та штучного інтелекту. Розробка гри на основі генетичного алгоритму допомагає вдосконалити та тестувати алгоритми штучного інтелекту, що можуть бути застосовані в різних областях, наприклад, управлінням роботами. Крім того, використання робототехніки дозволяє наочно продемонструвати можливості роботів та залучити більше людей до цього важливого напрямку технологічного розвитку.

Мета й завдання роботи:

Робота присвячена розробці гри для мобільного пристрою Android і створенню інтелектуального персонажу для неї з використанням генетичного алгоритму як способу знаходження оптимальних дій персонажу. В роботі послідовно розглянуто процес створення головного та інших ігрових персонажів та алгоритмів, за допомогою яких вони функціонують.

Для імітації штучного інтелекту буде використаний метод на основі генетичного алгоритму, що дозволяє надавати ігровому персонажу можливість самостійно приймати рішення щодо своєї траєкторії переміщення в просторі з перешкодами для досягнення положення спеціальної цілі, при цьому він змагається з персонажем, яким керує людина.

В якості гравця з інтелектом ми представимо персонажа, який знаходить мінімальний шлях до ближньої цілі. В той самий момент іншим гравцем керуватиме людина з таким самим принципом пересування та метою, однак вона сама в змозі вибирати ціль, до якої вона прямуватиме. В якості цікавого досліду можна здійснити порівняння правильності прийняття рішень цих двох різних персонажів для досягнення однакової мети.

Метою роботи є отримання в кінцевому результаті авторської гри-аркади з цікавими правилами та анімованим оформленням (в якості цікавості сюжетної лінії гри буде використано стиль мультфільму «Simon’s Cat»), при цьому основна увага буде приділятися розробці штучного ігрового інтелекту.

Об’єкт, методи й засоби розроблення:

Гра розроблена в середовищі Embarcadero Rad Studio 10 Berlin Delphi для мобільних пристроїв під управлянням Android. Гра може бути рекомендована як розвиваюча логіку та може бути цікава для широкої аудиторії. Також нами запропоновано, як використати алгоритм, застосований в грі, для створення настільної засобу розваги з використанням мініатюрного роботу; описано моменти реалізації системи для взаємодії з ним за допомогою передачі команд з

мобільного пристрою.

Можливі сфери застосування:

Створена гра може бути цікава в комерційних проектах, як продукт для розваг і в навчальних або розвиваючих цілях.

РОЗДІЛ 1: ІГРОВИЙ ШТУЧНИЙ ІНТЕЛЕКТ

1.1 Ігровий штучний інтелект

1.1.1 Визначення поняття "ігровий штучний інтелект"

Словосполучення "штучний інтелект" має вагомий і піднесений звук. Воно викликає уявлення про щось таємниче і недосяжне, над чим працюють справжні експерти програмування, які володіють секретними та майже містичними знаннями. Вони наділяють машину здатністю мислити подібно до людини [1].

Проте ігровий штучний інтелект - це просто набір правил і алгоритмів. Іноді складних, іноді не дуже. Зазвичай програма зводиться до простого алгоритму: комп'ютер точно знає, який хід є вигідним, але вводить в дії персонажа випадкові помилки.

Отже, ігровий інтелект – це набір програмних методик, які використовуються в комп'ютерних іграх для створення ілюзії інтелекту у поведінці персонажів, керованих комп'ютером. Ігровий інтелект, крім методів традиційного штучного інтелекту, також включає алгоритми теорії управління, робототехніки, комп'ютерної графіки та інформатики загалом [2]. Метою роботи ігрового інтелекту є розробка програмної системи, яка б забезпечувала створення реалістичної поведінки ігрових персонажів у грі. У іграх не потрібна така ж потужність, як у штучному інтелекті. Ігровий інтелект не потребує почуттів і самосвідомості, і йому не потрібно навчатися чогось поза межами ігрового процесу.

Справжня мета штучного інтелекту в іграх полягає в імітації розумної поведінки, забезпечення балансу між рівнем складності та викликами гри для гравців і в наданні гравцеві відчуття реалістичності та правдоподібного ефекту справедливості ворожих персонажів або опонентів, з іншого боку [3]. Це може включати розробку алгоритмів, які адаптуються до навичок гравця або використовують стратегії, які покращуються з часом.

Ігровий штучний інтелект може включати в себе різні алгоритми, такі як

правила, еволюційні алгоритми, машинне навчання та глибоке навчання, які використовуються для моделювання прийняття рішень, планування поведінки, розпізнавання образів та взаємодії з оточенням відеоігор.

Отже, ігровий штучний інтелект є важливим елементом розробки відеоігор, спрямованим на створення реалістичного геймплею, де комп'ютерні агенти виявляються здатними до інтелектуальної поведінки та пристосовуватися до змінних умов гри.

1.1.2 Огляд історії розвитку ігрового штучного інтелекту.

Розвиток ігрового штучного інтелекту пройшов великий шлях від початку свого існування до сучасного стану. Починаючи з ранніх експериментів у 1950-1960-х роках до досягнень останніх десятиліть, прогрес в цій галузі сприяв створенню все більш складних та реалістичних ігрових досвідів.

У 1950-х роках вчені, такі як Алан Тюрінг та Клод Шеннон, вперше почали досліджувати питання штучного інтелекту і створювати перші програми, які демонстрували інтелектуальну поведінку. В цей період ігровий штучний інтелект був обмежений через обмежену потужність обчислювальної техніки.

У 1970-1980-х роках розвиток ігрового штучного інтелекту отримав новий поштовх завдяки зростанню потужності комп'ютерів. Було створено прості алгоритми для керування ворожими персонажами в аркадних іграх, таких як Pong та Space Invaders.

В 1990-2000-х роках з'явилися ігрові індустрії, спеціалізовані в розробці ігрового штучного інтелекту. Були розроблені складніші алгоритми, що дозволяли ігровим агентам приймати розумні рішення в режимі реального часу. Наприклад, шаховий комп'ютер Deep Blue переміг у чемпіона світу Гаррі Каспарова у 1997 році.

В сучасний період, починаючи з 2010-х років, ігровий штучний інтелект розвивається з швидкістю світла. Великі досягнення в галузі глибокого навчання та нейронних мереж стали основою для створення потужних ігрових інтелектів. Проекти, такі як AlphaGo від DeepMind, демонструють вражаючі здібності в

перемозі над професійними гравцями в складних стратегічних іграх.

Зараз ігровий штучний інтелект втілюється в реалістичних ігрових середовищах, забезпечуючи вражаюче занурення в ігровий процес та інтелектуальний виклик для гравців. Інтеграція штучного інтелекту з віртуальною реальністю та розширеною реальністю відкриває нові горизонти для ігрового досвіду і створення більш реалістичних та емоційно насичених ігр.

Все це свідчить про неперервний розвиток ігрового штучного інтелекту, який стає все більш важливою складовою частиною відеоігрової індустрії, вдосконалюючи геймплей та створюючи захоплюючі та реалістичні враження для гравців.

1.2 Архітектура ігрового штучного інтелекту:

1.2.1. Моделі поведінки агентів у відеоіграх.

Моделі поведінки агентів у відеоіграх використовуються для створення реалістичного інтелектуального поведінки некерованих персонажів або штучного інтелекту в гральному середовищі. Ці моделі зазвичай розробляються з метою забезпечення більшого ступеня реалізму, виклику викликів та задоволення для гравців.

Типові моделі поведінки агентів у відеоіграх:

- Розкладання на підцілі: Ця модель визначає поведінку агента на основі певної послідовності підцілей. Агент поступово розкладає загальну мету на менші підцілі і приймає рішення на основі поточної ситуації і стану гри.
- Кінематика: Ця модель використовує фізичні закони і кінематику для визначення руху агента. Вона базується на розрахунку траєкторії агента, його швидкості, прискорення і маневреності.
- Машинне навчання: Застосування методів машинного навчання, таких як нейронні мережі, генетичні алгоритми або підсилення, дозволяє агентам вчитися з досвіду та самостійно покращувати свою поведінку у відповідь на різні ситуації в грі.
- Скрипти та правила: Цей підхід використовує попередньо визначені

скрипти або правила для керування поведінкою агента. Він вимагає вручну створених наборів правил, що визначають, як агент повинен реагувати на різні події або ситуації.

- Вирішення проблеми: Ця модель вимагає розрахунку оптимальних рішень для завдань агенту на основі аналізу доступних варіантів і їх оцінки за певними критеріями.

Ці моделі можуть комбінуватися або доповнюватися одна одну для досягнення бажаної поведінки агентів у відеоіграх. Від реалістичних противників до союзників з інтелектом, моделі поведінки агентів грають важливу роль у створенні захоплюючого грального досвіду.

1.2.2. Архітектури прийняття рішення у відеоіграх.

У відеоіграх існує кілька різних архітектур прийняття рішень, які використовуються для моделювання поведінки агентів і реалізації інтелектуальних систем. Ось кілька з них:

- Дерево прийняття рішень (Decision Trees): Це модель, яка представляється у вигляді дерева, де кожен вузол відповідає певному рішенню, а кожне ребро - можливому стану гри або події. Вона дозволяє агентові приймати рішення на основі послідовного аналізу умов та варіантів.

- Машинне навчання та нейронні мережі: Використання методів машинного навчання, таких як нейронні мережі, дозволяє агентам вчитися з досвіду та приймати рішення на основі аналізу великої кількості даних. Ці моделі можуть бути навчені для визначення оптимальних стратегій і вирішення конкретних завдань у грі.

- Підсилення (Reinforcement Learning): Це підхід до машинного навчання, де агент взаємодіє з оточенням і навчається на основі винагород та покарань. Він намагається знайти оптимальну стратегію, яка максимізує сумарну винагороду на протязі гри.

- Генетичні алгоритми: Цей підхід базується на еволюційних принципах,

де генетичні алгоритми використовуються для емуляції процесу природного відбору. Агенти представлені у вигляді популяції, яка еволюціонує шляхом комбінування, мутації та відбору найкращих особин.

- Скіп-грами та марковські процеси прийняття рішень: Ці моделі використовуються для моделювання невизначеності та залежностей між діями та станами гри. Вони можуть допомогти агенту приймати рішення на основі аналізу ймовірностей та очікуваних винагород.

Ці архітектури можуть використовуватися окремо або комбінуватися разом для досягнення бажаних результатів у відеоіграх та моделюванні поведінки агентів. Конкретний вибір архітектури залежить від типу гри, завдання, обмежень та інших факторів.

1.2.3. Використання станів та правил для прийняття рішень ігровими агентами.

Використання станів та правил є одним з підходів до прийняття рішень ігровими агентами. В цьому підході агенти оцінюють поточний стан гри і застосовують попередньо визначені правила для визначення своїх дій. Основні етапи такого підходу включають:

- Визначення станів гри: Гра може бути представлена у вигляді набору станів, які відображають поточний стан гри. Це можуть бути параметри, такі як положення агента, розташування противників, наявність ресурсів тощо.

- Визначення правил: За допомогою правил визначається, як агент повинен реагувати на різні стани гри. Правила можуть бути визначені вручну або шляхом аналізу та вивчення експертного досвіду. Наприклад, правило може гласити: "Якщо противник знаходиться у визначеному радіусі, агент атакує його."

- Оцінка станів та вибір дій: Агент оцінює поточний стан гри відповідно до заданих правил. Він обирає дію або стратегію, яка найкраще відповідає поточному стану. Це може бути визначено шляхом порівняння оцінок різних можливих дій.

- Виконання дій: Після прийняття рішення агент виконує вибрану дію у грі. Це може включати рух, атаку, збір ресурсів тощо.

Цей підхід може бути ефективним для простих ігрових ситуацій, де можна заздалегідь визначити набір правил на основі експертного знання. Однак, в складних іграх з великою кількістю можливих станів та варіантів дій цей підхід може стати обмеженим. У таких випадках використання інших підходів, таких як машинне навчання або генетичні алгоритми, може бути більш ефективним.

1.3 Персонаж з ігровим інтелектом

1.3.1 Зір персонажу

Якщо наш персонаж збирається приймати досить виважені рішення, йому необхідно знати, що відбувається навколо.

Віртуальні світи, в яких відбувається дія більшості ігор, мають величезну перевагу над реальним світом з точки зору ігрового інтелекту і його сприйняття. На відміну від реального світу, нам відомо кількість буквально всього, що є у віртуальному світі: десь серед ресурсів гри є список, де перераховується все, що існує в грі. Ви можете шукати за цим списком, вказавши потрібний запит, і миттєво отримати інформацію, яку ваш персонаж зможе використовувати, щоб приймати більш обґрунтовані рішення. При цьому можна або зупинитися на першому ж об’єкті, який буде представляти інтерес для вашого персонажу, або отримати список всіх об’єктів в межах заданої дальності, щоб агент зміг прийняти оптимальне рішення щодо навколишнього світу. Такий підхід непогано працює для простих ігор.

1.3.2 Навігація персонажу

Прийнявши рішення, інтелектуальному персонажеві потрібно зрозуміти, як рухатися з точки А в точку Б. Для цього можна використовувати різні підходи, вибравши оптимальний, залежно від характеру гри і від потрібного рівня продуктивності.

Алгоритм, умовно званий «зіткнутися і повернути», є одним з найпростіших способів формування маршруту руху об’єкта:

- рухатися в напрямку мети.

- якщо сталося зіткнення зі перешкодою, повернутися в напрямку, при якому можна опинитися найближче до цілі. Якщо жоден з доступних для вибору варіантів не має очевидних переваг, вибір робиться випадковим чином.

Такий підхід непогано працює для нескладних ігор, тому часто застосовується.

Якщо ж персонаж в грі повинен діяти не настільки безглуздо, то необхідно використовувати більш складні математичні моделі отримання траєкторії руху персонажу. Маючи бажання зробити персонажа з натяком на інтелект, можна спробувати застосовувати для цього найпростіші пошукові методи оптимізації з теорії ігрового інтелекту. До таких відноситься генетичний алгоритм.

1.4 Аркада як жанр комп'ютерних ігор

Взагалі, аркади – це монетні ігрові автомати, які найчастіше встановлюються в публічних закладах, в ресторанах, барах, а найчастіше в залах ігрових автоматів. І лише з недавнього часу термін «аркада» почали використовувати по відношенню до комп'ютерних ігор [6].

Штучний інтелект аркадних ігор, як правило, представляє собою простих персонажів з правилами поведінки, які не залежать від дії гравця. Незважаючи на те, що ігровий інтелект є простим, його шліфування – складне завдання, так як розробник повинен балансувати між двома факторами: захопленням гравця і монетизацією.

Суть класичних аркад пояснити досить складно. Зазвичай головною метою є проходження рівня за максимально короткий проміжок часу, збір всіх бонусів на рівні і отримання максимальної кількості очок. Сюди ж можна віднести таку гру як Pac-Man [7].

Ціль гри дуже проста – гравець знаходиться в лабіринті, наповненому

«їжею» (зображеної у вигляді точок), і йому потрібно з'їсти їх всі, щоб

пройти на наступний рівень. Завдання ускладнюють чотири привиди, які переслідують Pac-Man. Якщо Pac-Man зустрінеться з одним з привидів, він втрачає життя і повертається на початок, так само як з'їдені ним точки. Крім простої втечі від привидів, єдиний захист Пекмена – це чотири гранули-енерджайзери, розташовані в кутах лабіринту. Якщо з'їсти одну – у привидів вмикається режим переляку і вони відступатють, а на ранніх рівнях Pac-Man може з'їсти кого-небудь з них, щоб отримати бонусні бали. З'їдений привид не видаляє його повністю, а повертає на початкове положення, щоб знову почати переслідування. Крім поїдання точок і привидів, існує ще одна можливість отримати бонусні бали - фруктики, які з'являються на кожному рівні ближче до середини лабіринту. Перший фруктик з'являється, коли Pac-Man з'їв 70 точок в лабіринті, другий - коли 170 [8, 9].

В грі Pac-Man штучний інтелект привидів був тривіальний. У противників Пекмена було лише два стани: режим спокою та переслідування. У першому випадку привиди працювали за найпростішим алгоритмом: рухалися по прямій траєкторії і приймали рішення тільки на розвилках. Фактично кидався віртуальний жереб, який випадково визначав подальший маршрут привидів.

В режимі переслідування інтелект привидів, як і раніше, не міг похвалитися далекоглядністю, але при цьому вони бачили гравця по прямій траєкторії. Тобто в алгоритмі дій не враховувалися множинні розвилки лабіринту і примари вибирали маршрут до гравця випадковим чином. Саме ця недосконаліть інтелекту привидів зробило їх поведінку найчастіше помилковою, але тим самим більш реалістичною та непередбачуваною [9].

Рисунок 1 – Відомий вигляд гри Pac-Man

В поведінці привидів різного кольору проте існували незначні відмінності, що дозволяли зробити процес гри більш різноманітним (Додаток А).

Привиди справжнього «штучного інтелекту» не мають, хоча може здатися, що це не так. Вони просто слідують алгоритмам, що створює відчуття їх «розумності». В результаті виявляється, що в лабіринті є Pac-Man і чотири унікальних ворога, які переслідують його. Це робить гру цікавою, захоплюючою і веселою [10].

Навмисні помилки в іграх зі «штучним інтелектом» – це не програмна помилка, а заздалегідь придумана можливість і один з найпоширеніших прийомів розробників. Таким чином у гравця буде більше шанців виграти.

РОЗДІЛ 2: ГЕНЕТИЧНИЙ АЛГОРИТМ

2.1 Генетичний алгоритм

2.1.1 Поняття генетичного алгоритму

Генетичні алгоритми – це багатофункціональний інструмент для вирішення складних завдань. Генетичні алгоритми застосовуються в оптимізації, штучному інтелекті, інженерії та інших областях.

Цей алгоритм є евристичним алгоритмом оптимізації, який інспірований процесами природного відбору і генетики. Він використовує принципи еволюції для пошуку оптимального рішення у просторі можливих рішень.

Також генетичні алгоритми – це один з напрямків досліджень в області штучного інтелекту, що займається створенням спрощених моделей еволюції живих організмів для вирішення завдань оптимізації [4].

2.1.2. Ідея генетичного алгоритму

Основна ідея генетичного алгоритму полягає в створенні популяції особин (індивідів), кожна з яких представляється у вигляді хромосоми. Будь- яка хромосома кодує можливе рішення даної оптимізаційної задачі. Для пошуку кращих рішень необхідне тільки значення цільової функції, або функції пристосованості. Значення функції пристосованості особини показує, наскільки добре підходить особина, описана даною хромосомою, для вирішення завдання. Хромосома складається з кінцевого числа генів, представляючи генотип об’єкту, тобто сукупність його спадкових ознак. Процес еволюційного пошуку ведеться тільки на рівні генотипу. До популяції застосовуються основні біологічні оператори: схрещування, мутації, інверсії та інші.

В процесі еволюції діє відомий принцип «виживає сильніший». Популяція постійно оновлюється за допомогою генерації нових особин і знищення старих, де кожна нова популяція краща і залежить тільки від попередньої [5].

2.2 Опис та схема алгоритму

2.2.1 Схема роботи алгоритму

Загальна схема роботи генетичного алгоритму (Додаток Б):

- створення початкової популяції;
- обчислення функції пристосованості для осіб популяції (оцінювання);
- повторювання до виконання критерію зупинки алгоритму;
- вибір індивідів із поточної популяції (селекція);
- схрещення або/та мутація;
- обчислення функції пристосованості для всіх осіб;
- формування нового покоління.

З нашої зору перспективним є використання генетичного алгоритму при розробці ігор в тих випадках, коли поведінка персонажу (наприклад його траєкторія руху) може бути легко інтерпретована в термінах цього алгоритму.

2.2.2 Операції в генетичному алгоритмі

Операції в генетичному алгоритмі включають схрещування (кросовер), мутацію, відбір та інші додаткові операції. Розпишемо кожну з цих операцій:

1 Схрещування (кросовер):

Схрещування є процесом комбінування генетичної інформації двох батьківських рішень для створення нащадку. В процесі схрещування обираються певні точки або розрізи в генотипах батьківських особин, і генетична інформація між цими точками обмінюється. Результатом схрещування є нове рішення, яке має комбінацію генетичних властивостей батьківських рішень.

2 Мутація:

Мутація полягає у випадковій зміні генетичної інформації в окремих генотипах рішень. Це дозволяє вносити випадкові зміни, що сприяють

різноманітності популяції та дослідженню нових областей пошуку рішень. Мутація може впливати на один або кілька генів в генотипі залежно від ймовірності мутації.

3 Відбір:

Відбір полягає в виборі певних рішень з популяції для участі у схрещуванні та наступних поколіннях. Ймовірність вибору рішення залежить від його пристосованості або фітнес-функції. Рішення з вищою пристосованістю мають більшу ймовірність бути вибраними, але іноді низько пристосовані рішення також можуть мати шанс бути вибраними, щоб зберегти різноманітність популяції.

4 Інші операції:

Крім основних операцій, генетичний алгоритм може включати додаткові операції, які поліпшують його ефективність або розширюють його можливості. Наприклад:

- -Елітарний відбір: Кращі рішення з кожного покоління без змін включаються у наступне покоління, щоб зберегти найкращі рішення протягом еволюції.
- -Локальний пошук: Застосування локальних оптимізаційних методів для покращення рішень, що отримані за допомогою генетичного алгоритму.
- -Міграція: Обмін інформацією між популяціями для розширення пошуку рішень у ширшому просторі.
- -Адаптивні параметри: Зміна параметрів алгоритму протягом процесу еволюції на основі внутрішньо популяційних або зовнішніх факторів.

Ці операції у поєднанні з функцією пристосованості дозволяють генетичному алгоритму систематично експлуатувати простір рішень та знаходити оптимальні або прийнятні рішення в залежності від поставленої задачі.

2.3 Вплив параметрів на роботу генетичного алгоритму:

2.3.1 Розмір популяції:

Розмір популяції визначає кількість рішень, які утворюють початкову популяцію. Більша популяція може сприяти збільшенню різноманітності та швидкому знаходженню оптимального рішення, але вимагає більшого обсягу обчислювальних ресурсів.

2.3.2 Ймовірність схрещування і мутації:

Ймовірність схрещування визначає ймовірність того, що два батьківські рішення будуть обрані для схрещування, а ймовірність мутації визначає ймовірність виникнення мутації в окремому генотипі. Зміна цих параметрів може впливати на збільшення або зменшення різноманітності в популяції, а також на швидкість збіжності алгоритму.

2.3.3 Кількість ітерацій:

Кількість ітерацій або поколінь визначає кількість еволюційних кроків, які виконує генетичний алгоритм. Більша кількість ітерацій може покращити точність і збіжність алгоритму, але може також збільшити час виконання.

2.3.4 Функція пристосованості:

Функція пристосованості оцінює якість рішень у популяції. Вибір або створення відповідної функції пристосованості впливає на розв'язувану задачу та орієнтацію генетичного алгоритму. Правильне визначення функції пристосованості може покращити здатність алгоритму знаходити оптимальні рішення.

2.3.5 Інші параметри:

Крім основних параметрів, існують інші додаткові параметри, які можуть впливати на роботу генетичного алгоритму. Наприклад, можуть бути встановлені параметри для елітарного відбору, локального пошуку, міграції або адаптивних змін параметрів. Кожен з цих параметрів може мати вплив на ефективність, швидкість збіжності та здатність генетичного алгоритму знаходити оптимальні рішення в конкретній задачі.

Всі ці параметри потрібно належним чином налаштовувати та оптимізувати для конкретної задачі з метою досягнення найкращих результатів.

2.4 Застосування генетичних алгоритмів

Генетичні алгоритми широко застосовуються в багатьох галузях і задачах, де потрібно знайти оптимальні рішення або наближені рішення до складних проблем. Основна перевага генетичних алгоритмів полягає в їх здатності працювати з великими просторами рішень та знаходити відповіді в умовах обмеженої інформації. Деякі застосування генетичних алгоритмів включають:

- Оптимізація параметрів: Генетичні алгоритми можуть бути використані для оптимізації параметрів складних систем, таких як моделі, алгоритми машинного навчання або фінансові стратегії. Шляхом еволюції популяції рішень генетичний алгоритм знаходить наближені або оптимальні значення параметрів для досягнення кращої продуктивності або найкращих результатів.

- Проектування систем: Генетичні алгоритми можуть бути використані для проектування складних систем, таких як оптимальне розміщення мереж телекомунікацій, проектування електричних схем або структури багато потокових програм. Генетичні алгоритми допомагають знайти найкращі конфігурації та розподіл ресурсів для досягнення заданих критеріїв ефективності.

- Розкладання задач: Генетичні алгоритми можуть бути використані для розкладання складних задач на менші під задачі. Це особливо корисно у великих проектах або виробничих системах, де необхідно оптимізувати розподіл ресурсів, графіки виконання або послідовність дій.

- Проектування імітаційних моделей: Генетичні алгоритми можуть бути використані для створення імітаційних моделей та симуляційних систем. Вони допомагають генерувати еволюційні системи з певними правилами, що дозволяють досліджувати та аналізувати поведінку системи в різних умовах.

- Розробка штучного інтелекту: Генетичні алгоритми є одним із компонентів в розробці штучного інтелекту. Вони можуть бути використані для навчання та еволюції нейронних мереж, генетичного програмування та розв'язання різних задач машинного навчання.

Це лише кілька прикладів застосування генетичних алгоритмів. Вони знаходять своє застосування в багатьох галузях, де необхідно знайти оптимальні або прийнятні рішення в складних проблемах.

2.5 Застосування генетичних алгоритмів в комп’ютерних іграх

Генетичні алгоритми можуть бути застосовані в комп'ютерних іграх для вирішення різноманітних завдань. Ось кілька прикладів застосування генетичних алгоритмів в комп'ютерних іграх:

- Еволюція поведінки ворогів: генетичний алгоритм може використовуватись для еволюції поведінки ворогів у грі. Починаючи з початкового набору поведінкових правил для ворогів, генетичний алгоритм може здійснювати випадкові мутації і комбінування, щоб створити нові набори правил. Ці нові правила тестуються на ефективність у грі, і ті, що виявляються більш успішними, виживають та розмножуються, створюючи нащадків з покращеною поведінкою. По мірі прогресу еволюції, вороги стають все більш інтелектуальними і виконують більш складні стратегії.

- Генерація рівнів: генетичних алгоритмів може бути використаний для генерації нових рівнів у комп'ютерних іграх. Починаючи зі стартового набору рівнів, генетичний алгоритм може змінювати їх шляхом мутацій і комбінацій, створюючи нові рівні. Ці нові рівні оцінюються на основі критеріїв, таких як рівень складності, цікавість та різноманітність. Найкращі рівні вибираються для подальшого використання, або вони можуть бути поєднані, щоб створити ще більш складні та захоплюючі рівні.

- Підлаштування параметрів гри: генетичний алгоритм може

використовуватись для автоматичного підлаштування параметрів гри з метою поліпшення її геймплею. Наприклад, генетичний алгоритм може еволюціонувати набір параметрів, таких як швидкість руху гравця, максимальне здоров'я або сила ворогів, щоб знайти оптимальні значення, які забезпечують найкращий геймплей. Генетичний алгоритм може проводити численні ітерації, змінюючи та оцінюючи параметри, поки не буде досягнуто оптимального налаштування.

Генетичні алгоритми можуть бути використані для вирішення різних завдань, що стосуються ігрового процесу, штучного інтелекту та геймдизайну.

Ось кілька конкретних прикладів ігор, де використовуються генетичні алгоритми:

1. **Spore: Spore** - це симуляційна гра, розроблена Will Wright, творцем The Sims. У грі Spore генетичні алгоритми використовуються для еволюції ігрових істот. Гравці створюють своїх унікальних істот, вибираючи їх фізичні характеристики, а потім спостерігають, як вони еволюціонують та адаптуються до свого середовища.
2. **Creatures: Creatures** - це серія комп'ютерних ігор, де гравці вирощують та виховують віртуальних створінь, які мають свої власні генетичні коди. Генетичні алгоритми використовуються для емуляції генетичного спадку та еволюції створінь. Гравці можуть впливати на процес еволюції, вибираючи та схрещуючи створінь з найкращими характеристиками.
3. **Black & White: Black & White** - це стратегічна гра, розроблена Peter Molyneux. У грі гравці виступають в ролі бога та керують світом. Генетичні алгоритми використовуються для еволюції та розвитку тварин у грі. Гравці можуть взаємодіяти з тваринами, навчати їх та формувати їх поведінку, використовуючи генетичні алгоритми.
4. **NeuroEvolution of Augmenting Topologies** (NEAT) у грі **Mario: NEAT** - це метод генетичного програмування, який може бути використаний для навчання комп'ютерних агентів грати у відеоігри. В одному з проектів було використано NEAT для тренування агента, який може грати в оригінальну гру Mario. Генетичні

алгоритми дозволили агентові еволюціонувати та поліпшувати свої навички у грі протягом багатьох поколінь.

5. **Galactic Arms Race: Galactic Arms Race** - це науково-фантастична гра стратегії, де гравці створюють та еволюціонують власний флот космічних кораблів. Генетичні алгоритми використовуються для генерації нових кораблів та оцінки їх ефективності у боях. Гравці можуть вибирати найкращі кораблі для використання в своїх флотах, щоб поліпшити свої шанси на перемогу.

6. **Darwinbots: Darwinbots** - це імітаційна гра, в якій гравці створюють та еволюціонують віртуальних роботів, які змагаються один з одним у штучному середовищі. Генетичні алгоритми використовуються для визначення генетичного коду роботів та їх еволюції. Гравці можуть спостерігати, як їх роботи навчаються та адаптуються до змінюваних умов гри.

7. **Fable: Fable** - це серія рольових ігор, розроблених Peter Molyneux. У грі Fable генетичні алгоритми використовуються для еволюції та змінення вигляду персонажів. Гравці можуть спарювати персонажів із різними генетичними характеристиками, що призводить до народження нащадків з унікальними зовнішніми ознаками.

8. **Genetic Gladiators: Genetic Gladiators** - це мобільна гра, де гравці створюють команду мутантів із різними генетичними характеристиками та змагаються з іншими гравцями. Генетичні алгоритми використовуються для комбінування генів мутантів та створення нових комбінацій. Гравці можуть експериментувати з різними генетичними складами, щоб створити сильну та унікальну команду бійців.

Це лише декілька прикладів ігор, де використовуються генетичні алгоритми. Застосування генетичних алгоритмів у комп'ютерних іграх досить широке і може бути використане для різних аспектів.

Використання генетичних алгоритмів в комп'ютерних іграх виявляється корисним і цікавим підходом, що додає глибину та різноманітність до геймплею. Генетичні алгоритми дозволяють симулювати процеси еволюції, адаптації та

вибору в ігрових середовищах, що дозволяє створювати унікальні та цікаві ігрові елементи.

Ці алгоритми можуть бути застосовані для еволюції істот, генерації рівнів, балансування геймплею, оптимізації параметрів і багато іншого. Вони дозволяють створювати унікальних та непередбачуваних персонажів, створюючи враження живого світу. Крім того, вони дозволяють гравцям взаємодіяти з ігровим середовищем на новому рівні, пропонуючи непередбачувані виклики та експерименти.

Хоча поки що немає надзвичайно популярних або культових ігор, де генетичні алгоритми були основною механікою, їх використання в іграх продовжує розширюватися і еволюціонувати. Застосування генетичних алгоритмів у комп'ютерних іграх відкриває нові можливості для творчості, інновацій та експериментів, допомагаючи створювати унікальні та захоплюючі ігрові враження.

РОЗДІЛ 3: РОЗРОБКА ВЛАСНОГО ІГРОВОГО ШТУЧНОГО ІНТЕЛЕКТУ НА ОСНОВІ ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Постановка задачі

Найчастіше в аркадних іграх персонажі лише імітують розумну поведінку, і прийняття рішень повністю базується на випадкових рішеннях. З нашої точки зору набагато цікавіше було б грати проти персонажу, який дійсно має інтелект. Ми пропонуємо в якості розуму персонажу використати правила поведінки природи, тобто еволюційні правила, які можна змоделювати за допомогою генетичного алгоритму.

Нами пропонуються наступні правила гри. Припустимо, що в нас є два гравці, один з яких керується людиною (герой), а інший – за допомогою генетичного алгоритму (робот). Персонажі розміщені в двовимірному просторі. В цьому ж просторі також розміщенні два типи об’єктів: перешкоди, які потрібно обходити так, щоб гравці не знаходилися в певному радіусі цієї прешкоди, та цілей.

Фізика гри: в грі один проти одного грають два персонажі. Обидва персонажі мають однакові можливості руху:

1. персонажі мають вибір робити кроки тільки вправо чи вліво;
2. не можна робити хід, якщо він лежить в деякому радіусі від перешкоди;
3. той, хто знайде більше цілей, стане переможцем в раунді;
4. можливий варіант, коли обидва персонажі потраплять в глухий кут одночасно. В такому випадку буде нічия.

Персонаж, який керується генетичним алгоритмом, ходить до найближчої доступної цілі, і так кожен раз від цілі до цілі.

Персонаж, яким керує людина (герой), здатен сам вибирати ціль, до якої

йти. Також цей персонаж може робити тільки 3 повороти за хід, після цього одразу виконується хід персонажа, яким керує генетичний алгоритм.

Персонажам ми надаємо можливість мати вибір ходу лише вправо або вліво. Таким чином, вони, не маючи можливості розвернутись, можуть потрапити в глухий кут (рис. 2). Персонаж для цього повинен мати таке положення, при якому будь-який наступний крок переміщує його в зону недоступності.

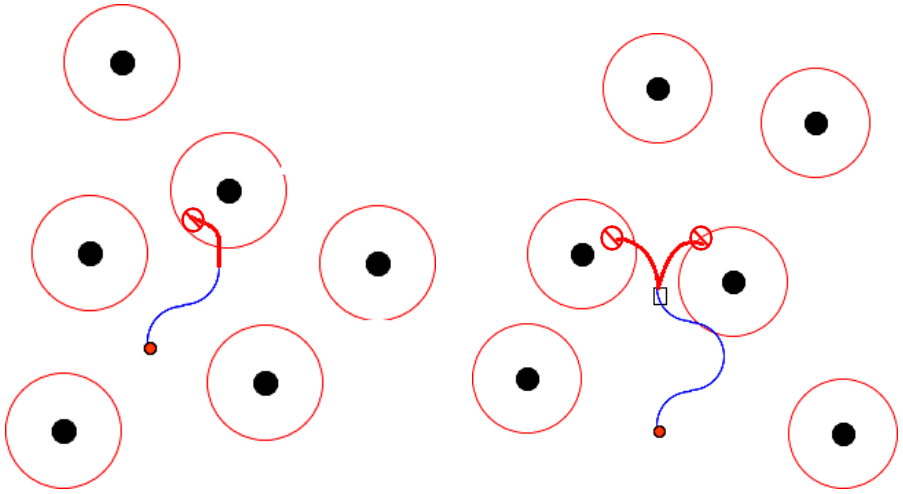


Рисунок 2 – Схема що пояснює попадання в глухий кут (пастку): синій шлях – можливий для переміщення, червоний – не можливий для переміщення.

3.2 Алгоритм руху ігрового персонажу та правила гри

Для персонажу зі «штучним інтелектом» (роботу) необхідний розв’язок задачі знаходження оптимальної послідовності команд для переміщення до цілі з малою свободою рухів в просторі з перешкодами.

В нашій інтерпретації нами пропонується вважати хромосомами пройдений шлях, а генами – команди рухатись або вліво, або вправо (рис. 3).

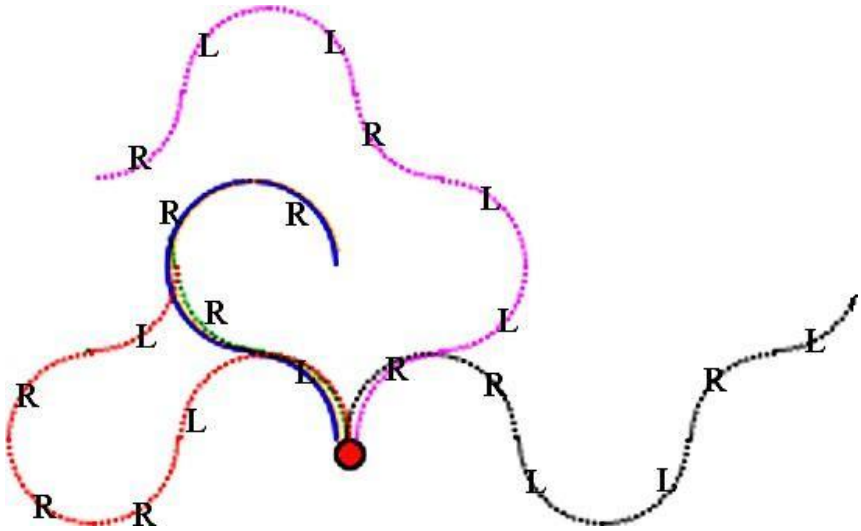


Рисунок 3 – Представлення у вигляді хромосоми послідовності команд для шляху.

Червоною точкою є початкове положення.

Алгоритм знаходження оптимального шляху:

1) задаємо початкову популяцію з декількох шляхів у вигляді масиву з випадкових команд.

2) робимо розрахунок для кожного шляху (особини) згідно команд і з врахуванням точкових перешкод, і знаходимо відстань від очікуваного кінцевого положення до цілі.

3) сортуємо шляхи за параметрами мінімальної відстані до цілі та кількості необхідних команд.

4) робимо схрещування найкращих хромосом між собою і отримуємо потомство з деякою кількістю мутованих генів, що замінюють собою найгірші шляхи.

5) повертаємось до пункту 2) і повторюємо цикл певну кількість раз.

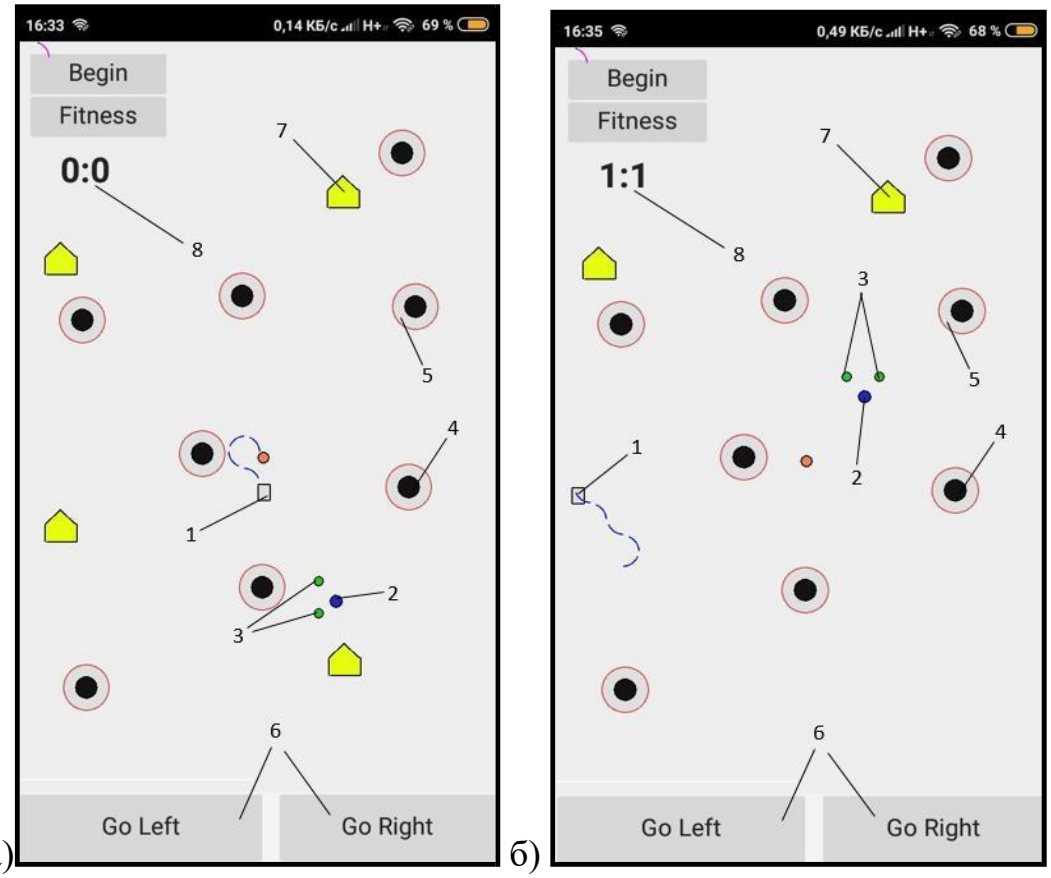
Програма в кінцевому результаті бере набір команд з найкращої хромосоми, які потім виконуються роботом.

3.3 Програмна реалізація алгоритмів гри

Додаток у вигляді мобільної гри нами створювався у Embarcadero Rad Studio 10 Berlin Delphi. Вибір цього середовища був обумовлений потужними інструментами для візуальної розробки, можливістю використання мови програмування Object Pascal та великою бібліотекою документації і прикладів робіт з обчислювальними, мультимедійними та телекомунікаційними можливостями мобільних пристроїв.

Окремі фрагменти нашого коду приведено у додатках В-Д.

На рис.4 показано «голе» схематичне представлення гри у вигляді програми без графічного оформлення. Зі схеми можна зрозуміти, що в процесі гри робот дійшов до цілі. В той час як герой теж дійшов до цілі. Отже в даному раунді, показаному на схемах, вийшла нічия.



а) б)

Рисунок 4 – Схематичне представлення програми у вигляді функціональних елементів (без графічних надбудов) а – початок гри, б – в

процесі гри.

1 – положення робота; 2 – положення героя; 3 – можливий поворот героя 4 – перешкоди; 5 – зона недоступності; 6 – кнопки для повороту вліво чи вправо для героя; 7 – ціль; 8 – лічильник рахунку для персонажів

3.4 Графічне оформлення та кінцевий вигляд мобільної гри

Для естетичного сприйняття важливо якісно візуально оформити функціональні елементи гри та зробити цікаву анімацію. Ми представили гру в чорно-білому стилі, також відомому як стиль мультфільму «Simon's Cat».

На рис. 5 представлено графічне оформлення функціональних елементів гри.

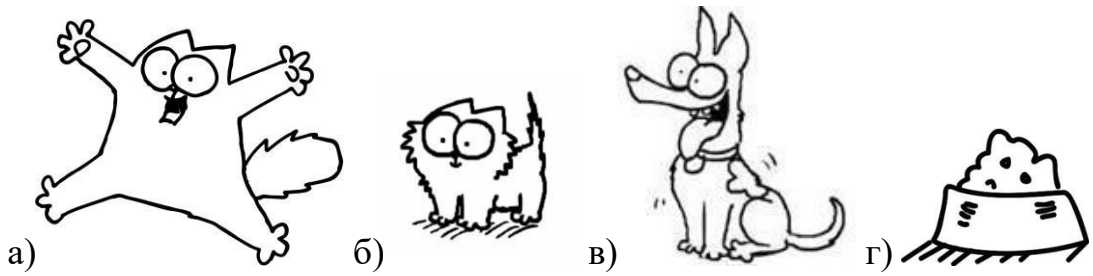


Рисунок 5 – Вигляд графічного оформлення персонажів та елементів гри: а) – герой; б) – робот; в) – перешкода; г) – ціль.

Нами використані файли типу png з прозорим фоном. Для їхнього створення застосовувалась обробка картинок в програмі Adobe Photoshop CC 2019. При зміні зображень для анімації використовувався компонент TTimer.

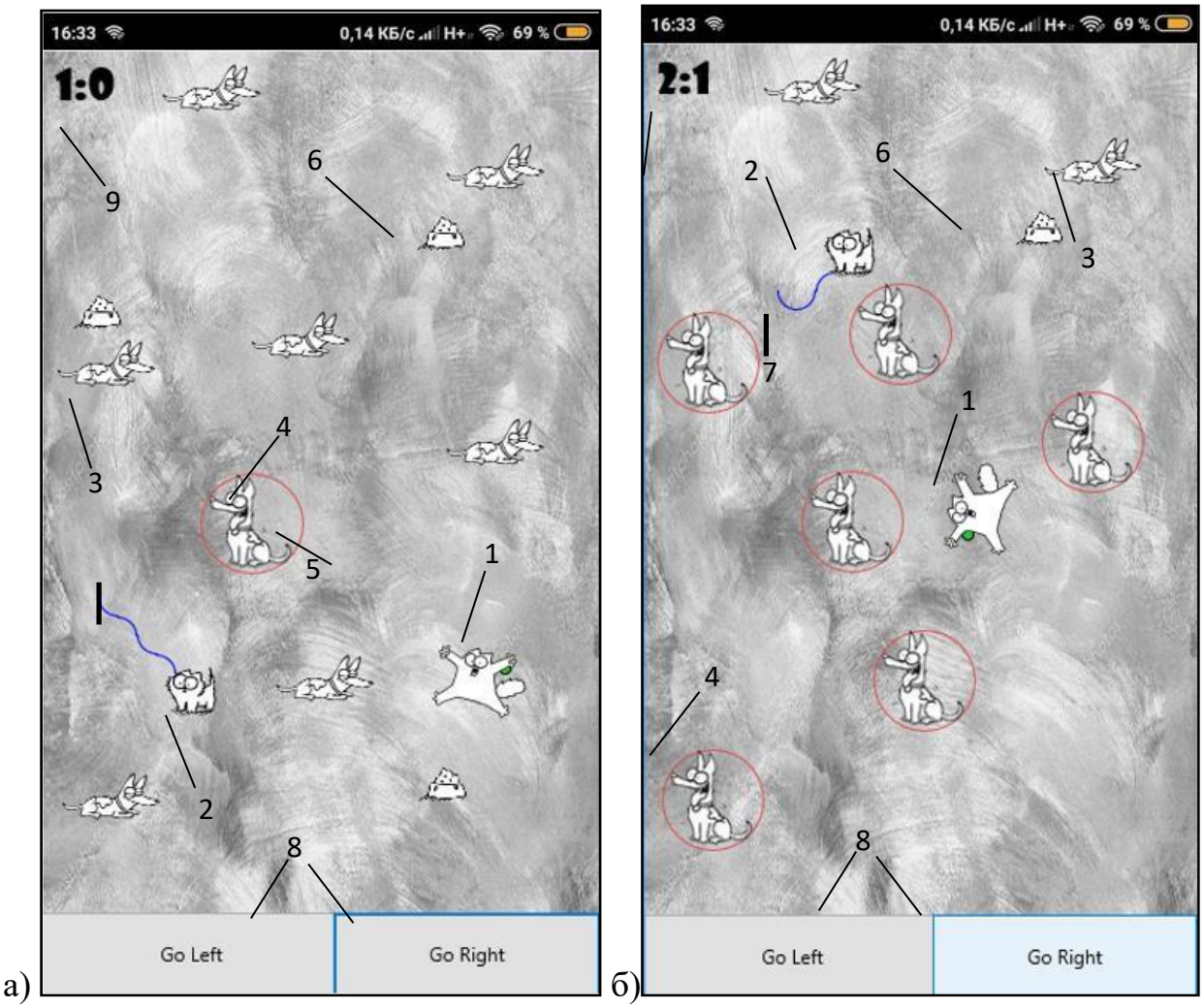
В процесі гри зображення героя (рис. 5а) змінює кут повороту в залежності від вибраної команди (повороту вліво чи вправо) а також від напрямку руху.

Для надання більшої ефектності також використовувались і звичайні компоненти, такі як TArc, TCircle, TLabel та TRectangle.

Якщо будь-який з персонажів опиняється в деякому радіусі від перешкоди, то ця перешкода (рис. 5в) стає активною (змінює картинку) і навколо неї з'являється коло, що показує зону недоступності.

На рис. 6 зображений остаточний вигляд гри. Роздільна здатність та

розміри графічних компонентів гри підбирались для найкращого відображення у мобільному додатку.



а) б)

Рисунок 6 – Кінцевий варіант вигляду гри: 1 – герой; 2 – робот; 3 – неактивна перешкода; 4 – активна перешкода; 5 – зона недоступності; 6 – ціль; 7 – шлях робота; 8 – кнопки для повороту героя вправо чи вліво; 9 – лічильник.

Щодо робота (рис. 5б), то для анімації його зображення, в процесі ходу запропоновано і реалізовано ефект метушливого бігу навколо його початкового положення, що створює імітацію поведінки грайливого котика. Цей ефект створено проміжним показом варіантів шляхів, знайдених генетичним алгоритмом до моменту знаходження оптимального.

Підходи анімації дозволяють зробити гру більш ефектною та привабливою для гравців.

Запис процесу гри в реальному часі можна подивитись за посиланням: https://www.dropbox.com/sh/h0i5ch4swptke90/AAAYObf2cD0HhioFaWYWMxkpa?dl=0

Гра може бути розширена на декілька різних по складності рівнів при збільшенні або зменшенні кількості перешкод, варіюванням кількості хромосом та ітерацій в персонажі-роботі, що відповідає за «розумність» поведінки цього персонажу.

РОЗДІЛ 4: РЕАЛІЗАЦІЯ ГРИ У ВИГЛЯДІ НАСТІЛЬНОГО ЗАСОБУ РОЗВАГ

4.1Основна ідея реалізації настільної гри

Гра з наведеними вище персонажами (розділ 3), правилами та алгоритмом руху може бути реалізована у вигляді настільного засобу для розваг за допомогою простих дешевих механізмів у вигляді брістлботів або віброботів.

Принцип відтворення гри в реальності: на всі елементи гри наводиться камера мобільного телефону (рис. 7) із створеною нами програмою, яка спочатку аналізує положення всіх елементів та розташовує їх на полі гри, відповідно їх координатам та орієнтації.

Програма, за таким самими алгоритмом, що і в грі, розраховує шлях робота повз перешкоди до цілі і надсилає розраховані команди роботу. Робот відтворює рухи по четвертях кола в реальності, імітуючи поведінку віртуального персонажу.

В якості цілей та перешкод використовуються світлодіоди різного кольору (білий, червоний, синій та зелений) та тенісні кульки на них (у якості розсіювача світла). На роботі також встановлені два світлодіоди різних кольорів для простішого визначати його положення та орієнтації.

Далі приведемо основні деталі реалізації блоків програми керування системою. В програмі нами враховані відносні відстані між об’єктами різного кольору (білий – для цілі, синій та червоний – для крайніх точок робота, зелений – для перешкоди) та їх масштабування відповідно до видимих розмірів робота (це зроблено для того, щоб надати можливість знімати камерою з довільною відстані). Також ми робимо перетворення системи координат згідно з матрицею повороту [11, 12], для того, щоб забезпечити будь-яке позиціювання телефону відносно робота.

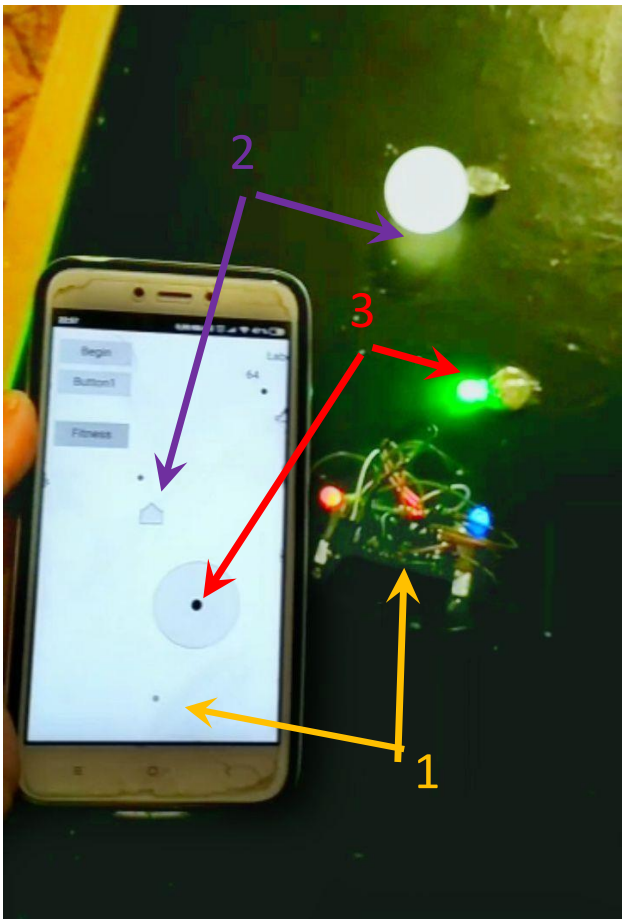


Рисунок 7 – Ілюстрація, що пояснює загальний принцип запропонованого нами перенесення гри у реальність: гравець наводить камеру смартфону на робота, ціль та перешкоди і робот здійснює кроки до цілі, не дотикаюсь до перешкод. 1 – робот; 2 – ціль; 3 – перешкода.

4.2 Брістлботи та віброботи

Вібраційні роботи – це прості роботи, які використовують крихітний двигун з вібратором; вони є предметом багатьох проектів Maker / DIY, а також деяких відомих комерційних іграшок [16].

На початку 2000-х років з’явилась можливість розробляти мініатюрні роботи, рушійною силою яких стали вібромоторчики, що у великій кількості з’явились при масовому розповсюдженні мобільних телефонів.

За основною конструкцією таких роботів прийнято ділити на брістлботи або віброботи.

Брістлбот – це простий робот із жорстким корпусом, де нижня поверхня являє собою щітку або іншу щетинисту поверхню (рис. 8) [17]. Форма та напрям щетинок орієнтована таким чином, що існує загальний нахил до вертикалі і при вібрації двигунів це дає бажаний напрямок вперед. Робот керується вібрацією свого тіла. Дія цих коливань через масу щетинок полягає в тому, щоб поступово рухати робота в переважному напрямку щетинок. Механізм руху робота нагадує принцип пересування джгутикових бактерій.

Рисунок 8 – Найпростіша реалізація брістлбота (робот-тарган)

Вібробот – це схожий робот, але базується він на використанні пружних дротяних ніжок, а не щіточок (рис. 9). Вони стали традиційною темою в роботах бразильського дизайнера Чико Бікало.

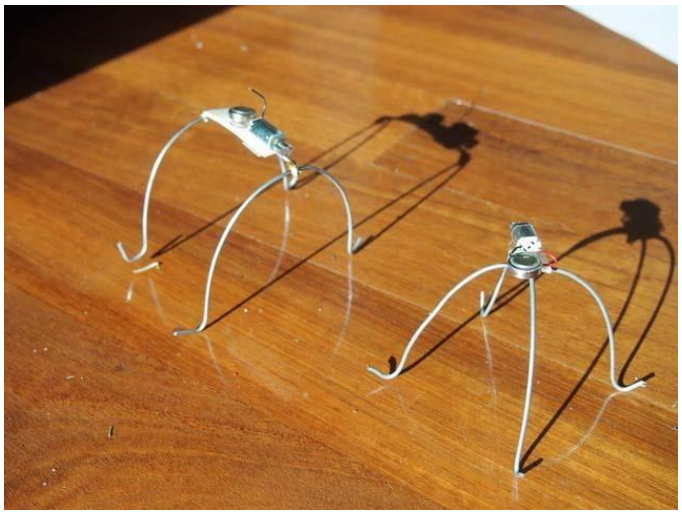


Рисунок 9 – Найпростіша реалізація вібробота (робот-павук)

За можливості максимально зменшити розміри роботів такі конструкції знайшли поширення в науковій області. Вони часто зустрічаються в проектах з ройових робототехнічних систем. Існували кілька різних конструкцій (роботи Robot Kilobots, I-Swarm), які створювали великі групи керованих віброботів (рис. 10)[16].

а)  б) 

Рисунок 10 – а – вібробот MiCRoN project's; б – роботи Kilobots

Останнім часом все більше зустрічаються проекти з використанням одночасно двох моторчиків та інфрачервоним управлінням. Контролюючи швидкість двигуна за допомогою ширини імпульсу, можна зробити так, що

робот рухається вперед і/або повертає. На рис. 11 показаний стандартний Bristlebot, який має два двигуни, зміщені від центральної осі [5].

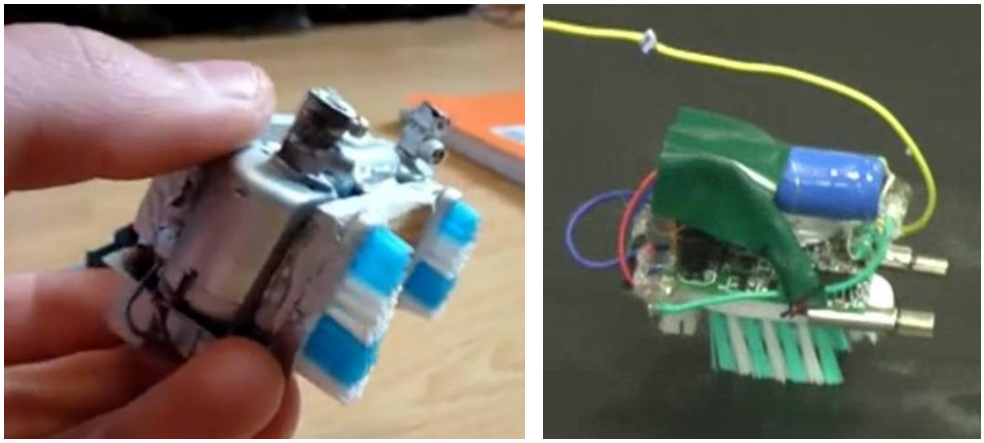


Рисунок 11 – Брістлбот з двома двигунами

Також з двома моторчиками можна створити вібробота, який матиме приблизно такі самі характеристики. На рис. 12 зображеного одного з таких роботів [18].

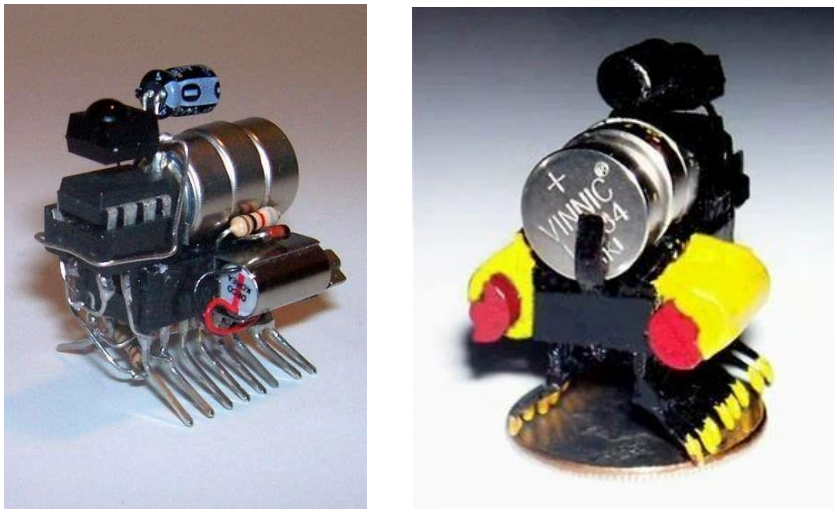


Рисунок 12 – Вібробот з двома двигунами

Апаратне забезпечення роботів досить просте: невелика друкована плата, два вібраторні двигуни, мікроконтролер, інфрачервоний фотодіод і дуже маленький акумулятор. Програмне забезпечення для такого типу робота також

не дуже складне і це дозволяє йому стати чудовим проектом DIY та потенційно хорошим продуктом масового ринку.

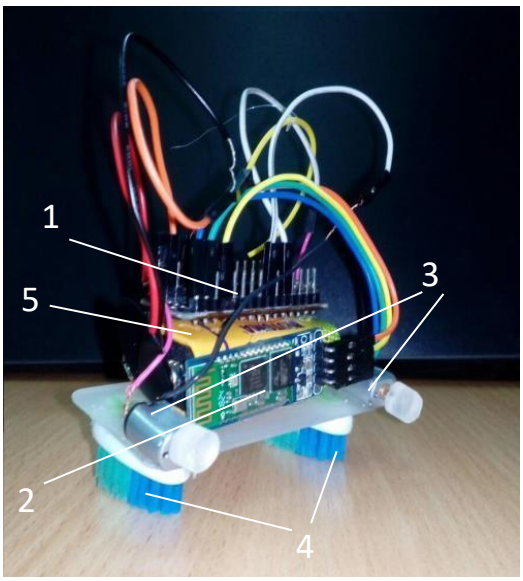
Отже цей робот є бюджетним та доступним у створенні, простим у використані та розумінні. Цей тип роботів є відмінним варіантом для використання у нашій грі.

4.3 Структура та логіка брістлбота

Для реалізації мого проекту було використано платформу Android в якості програмного конструктора та Arduino в якості апаратного.

На рис. 13 представлено створену модель, яка використовувалась при дослідженні. Ми використали конструкцію з двома моторчиками та керування через Bluetooth. Нами було помічено значну чутливість керованості робота від форми та орієнтації щіточок. Щіточки повинні бути спеціально підрізані таким чином, щоб робот був нахилений під певним кутом (рис. 13б), який дозволяє перетворювати вібраційну енергію в поступальну з одночасним поворотом робота відносно іншої щіточки. Також рухи робота чутливі до коефіцієнтів тертя поверхонь, на яких він пересувається.

В додатку Е представлено скетч, який використовувався для управління роботом з мобільного пристрою.

а) 

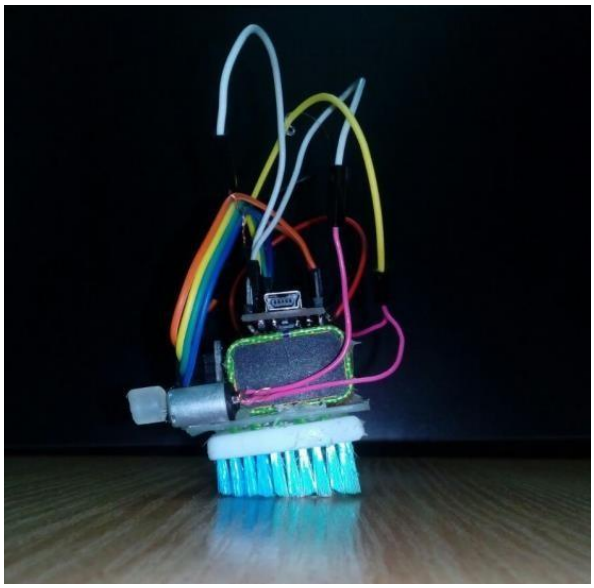
б) 

Рисунок 13 – Будова робота: а) загальний вигляд; б) вигляд з боку. 1- плата ARDUINO NANO; 2- Bluetooth module HC-06; 3- вібромоторчики; 4- щіточки для пересування; 5- батарейка «Крона».

При тестуванні такої моделі робота ми зіткнулись з проблемою, що даний робот важко керується за допомогою ручного керування, тому ми вирішили вдосконалити його інтелектуальним управлінням.

Виявлено, що робот пересувається лише по дугах кола в різні сторони і повертаючись на 90^o, то створюються умови для створення алгоритму дістатися до цілі з мінімальним набором команд і при цьому обминати перешкоди, які трапляються в нього на шляху. Для виконання поставленої задачі на допомогу прийшов генетичний алгоритм, який ідеально інтерпретується до заданої задачі.

Далі запропоноване авторське удосконалення керування роботом з вібраційним принципом руху за допомогою генетичного алгоритму в задачі знаходження оптимального шляху до певного предмету з обходженням інших предметів.

4.4 Система управління та постановка задачі

Основною метою є розв’язок задачі, знаходження оптимальної послідовності команд для переміщення робота до цілі з малою свободою рухів в просторі з перешкодами.

Для цього будемо використовувати можливості сучасних мобільних пристроїв – камеру телефона, Bluetooth або інфрачервоний зв’язок та його обчислювальні потужності.

Телефон направляється на поле з роботом, перешкодами та ціллю, програмою, яка аналізує зображення, визначаються координати всіх названих об’єктів на полі і далі вони передаються до розрахункової частини програми.

Для того, щоб спростити аналіз візуальної інформації, я додатково забезпечила робота двома світлодіодами різного кольору: синій та червоний. Це

дало мені змогу визначати орієнтацію робота та відстань до нього відносно android пристрою, оскільки аналізуємо вже сильно контрастуючи за кольором елементи, які легко знаходяться на неяскравому фоні.

Для цього я використовувала найпростіші методи комп'ютерного зору, а саме виділення певного відтінку кольору серед всіх інших за допомогою формули:

$$(R - R_0)^2 + (G - G_0)^2 + (B - B_0)^2 < \delta$$

Операція проводилась у RGB у просторі кольорів окремо для зелених, синіх, червоних та білих точок (рис.14).

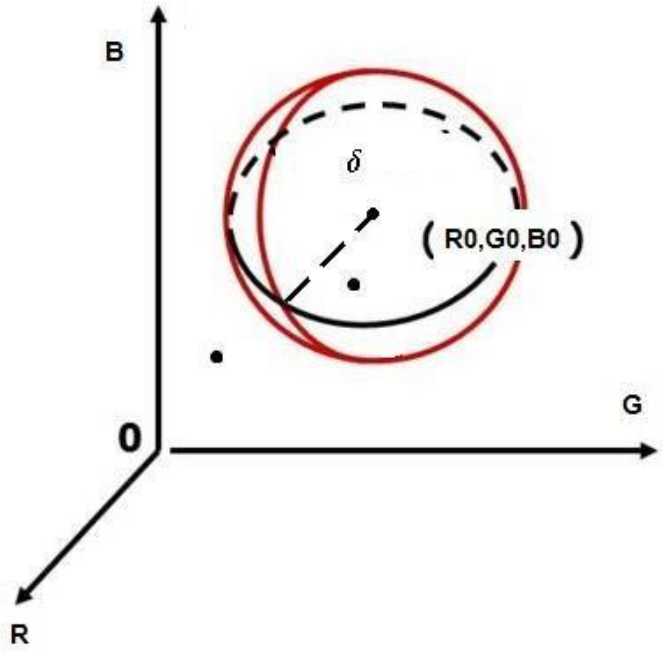


Рисунок 14 – Належність певної кольорової точки до визначеного кольору (R_0, G_0, B_0) в просторі кольорів зображення.

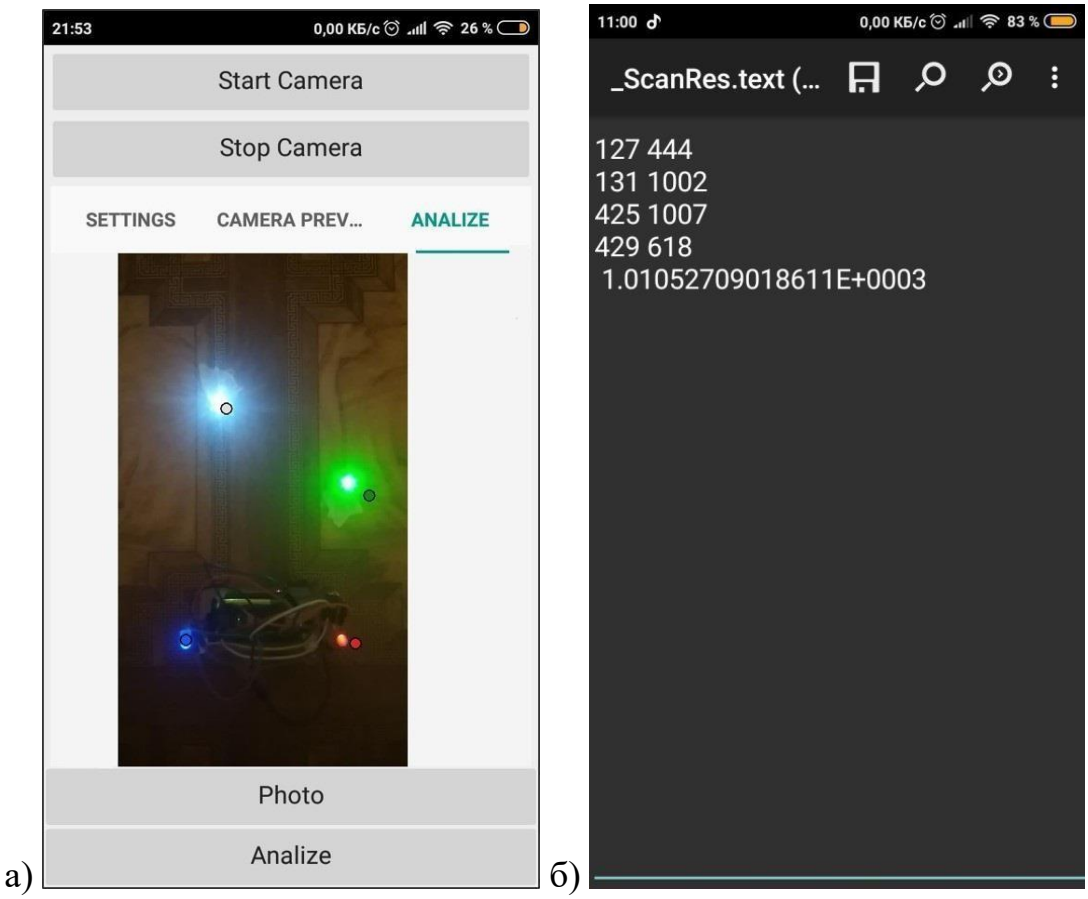
Після виділення області знаходились координати центру області обраного кольору (x_c, y_c), що відповідала положенню світлодіодів робота, цілі, та перешкоди:

$$x_c = \frac{\sum_{x=0}^{w-1} \quad \sum_{y=0}^{h-1} \quad xI(x,y)}{S}$$

$$y_c = \frac{\sum_{x=0}^{w-1} \quad \sum_{y=0}^{h-1} \quad yI(x,y)}{S}$$

Результат роботи такого методу знаходження орієнтації та положення об’єктів показано на рисунку 15.

Програмна оболонка створювалася в Embarcadero Rad Studio 10 Berlin Delphi за допомогою мови програмування Object Pascal з використанням семплів для роботи з камерою [20].

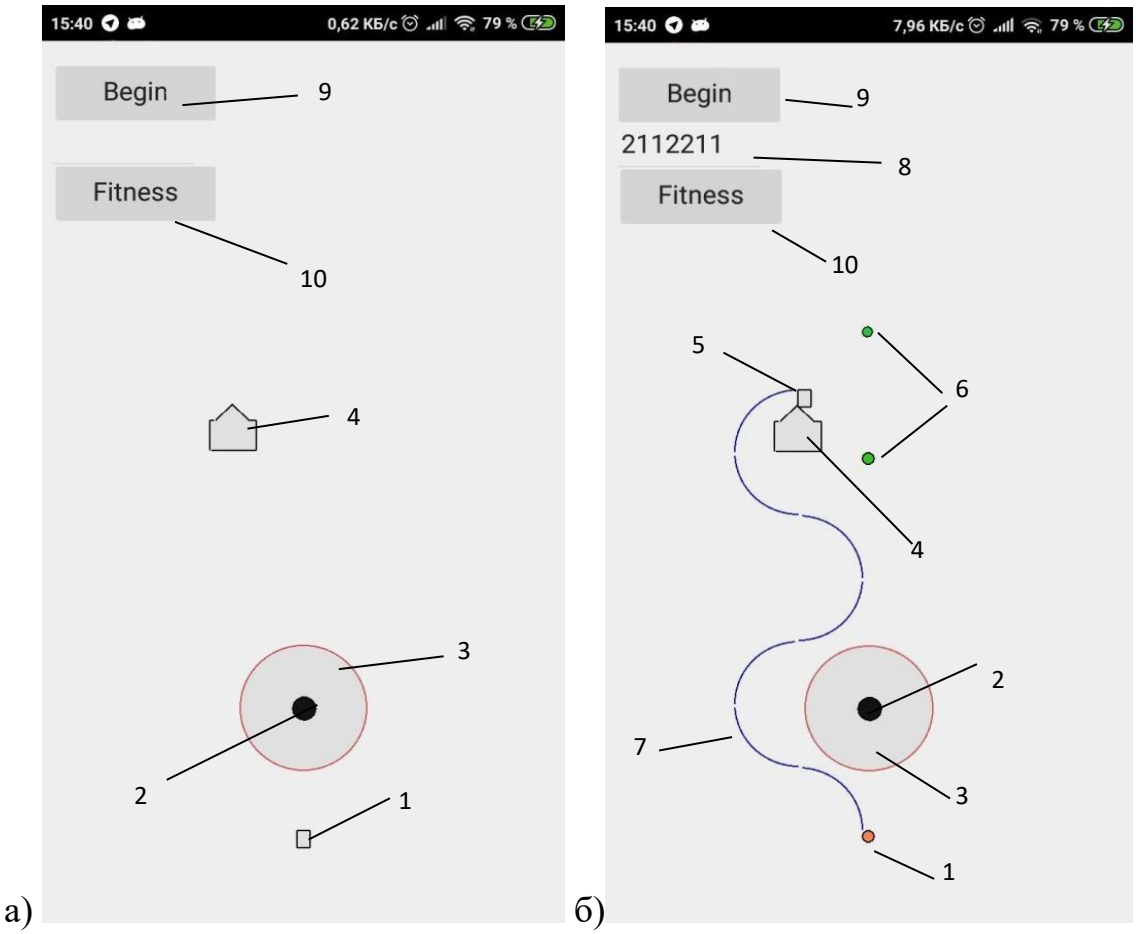


а) б)

Рисунок 15 – Результат роботи такого методу знаходження орієнтації та положення об’єктів. Точками (а) позначено знайдені об’єкти: синім – лівий край робота, червоним – правий край робота, білим – ціль, зеленим – перешкоду. б) – координати, що відповідають згори до низу відповідно координатам білої, синьої, червоної та зеленої точок.

Після знаходження відносних координат всіх елементів, вони передаються до блоку програми з алгоритмом обрахунку набору кроків до цілі оптимальним шляхом.

В нашій програмній інтерпретації генетичного алгоритму хромосомами є пройдений шлях, а генами команда рухатись або вліво, або вправо, а популяцією є набір хромосом з різним набором генів. Виконуючи стандартні правила генетичного алгоритму, можна отримати найкращий варіант для переміщення. Результат роботи програми показано на рисунку 16.



а) б)

Рисунок 16 – Схема пояснення роботи інтелектуального обходу перешкоди при рухові до об’єкту. 1 – початкове положення робота; 2 – перешкода; 3 – зона недоступності; 4 – мета; 5 – кінцеве положення робота; 6 – можливе положення робота після наступного кроку; 7 – шлях робота; 8 – набір команд, які потрібно надіслати роботу; 9 – кнопка для зчитування інформації про положення об’єктів, які були отримані блоком аналізу зображення з

камери; 10 – кнопка для початку алгоритму з визначенням оптимального шляху.

Програма в кінцевому результаті бере набір команд з найкращої хромосоми і повинна надіслати їх роботу.

На рисунку 17 показано кадри покрокового виконання роботом набору команд, які дозволяють пройти шлях до цілі не торкнувшись перешкоди.

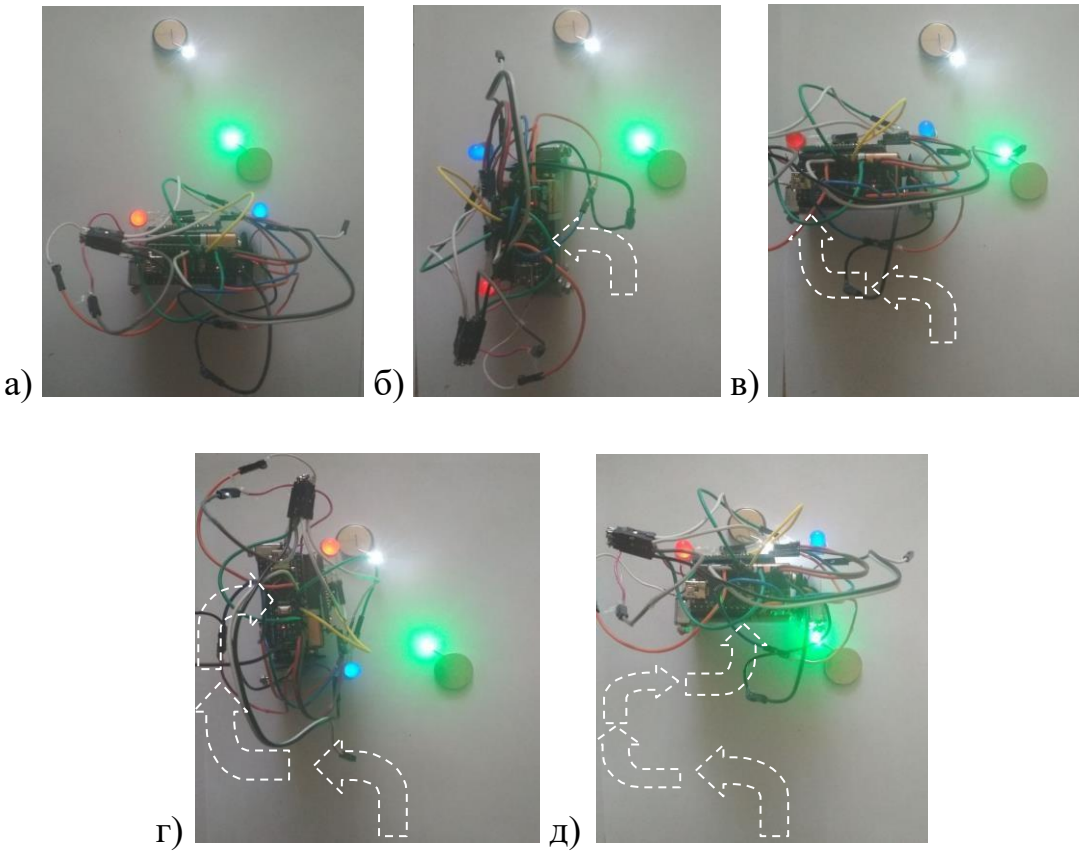


Рисунок 17 – Кадри з поступовим виконанням команд з Android пристрою віброботом; зелена точка відповідає за перешкоду, біла – за ціль; а – початкове положення робота; б-г – рух робота покроково до цілі з огинанням перешкоди; д – кінцеве положення.

Якщо художньо оформити роботів у вигляді іграшок, то їх рухи можуть ефектно виглядати та бути цікавими як настільний засіб розваги.

4.5 Перехід до аналогових компонентів.

Оцінивши вартість робота, я дійшла до висновку, що 75% від загальної вартості робота становлять тільки плата Arduino та Bluetooth module. Саме тому я вирішила замінити ці компоненти схеми на деяку аналогову схему з інфрачервоним приймачем, тобто перетворити робота на beam-робота.

Слово BEAM є абревіатурою від Biology, Electronics, Aesthetics, Mechanics. Це термін, що позначає принцип побудови роботів, що використовує прості аналогові ланцюги (наприклад, компаратори) замість мікропроцесорів з метою досягти незвично простого (в порівнянні з традиційними пересувними роботами) дизайну, який жертвує гнучкістю заради надійності і ефективності виконання певного завдання [21].

Цим можна набагато зменшити собівартість робота. На рис. 14 подано схему такого beam-робота. На додатку Є подано принципову схему.

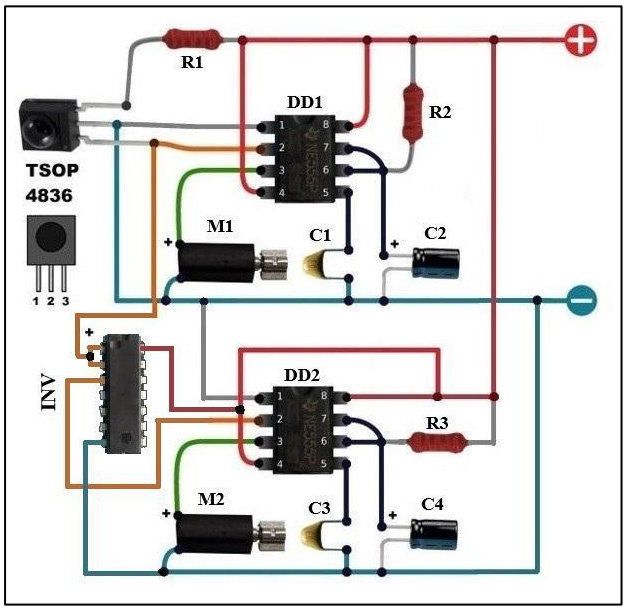


Рисунок 18 – Аналогова схема для вібробота з двома моторчиками, що керується інфрачервоним датчиком. TSOP – інфрачервоний приймач; DD1,DD2 – програмований таймер і осцилятор; M1, M2 – вібромоторчики; INV – інвертор; R1, R2, R3 – резистори; C1, C3 – конденсатори керамічні; C2, C4 – електролітичні конденсатори.

В розробленій схемі є два вібромоторчики. Коли на схему не подається інфрачервоне випромінювання, то працює перший вібромоторчик, а коли навпаки подається, то перший вібромоторчик припиняє працювати і починає працювати інший. Інфрачервоне світло будемо передавати за допомогою мобільного телефону, в більшості яких є інфрачервоний передавач.

Дана схема дасть можливість значно зменшити собівартість робота, що дасть змогу зменшити й вартість вихідного продукту, що зробить його більш доступним для користування.

ВИСНОВКИ

Мною було запропоновано ідею створення гри типу аркада для мобільного пристрою з персонажем, що рухається між перешкодами до цілі за допомогою генетичного алгоритму.

На відміну від більшості простих аркад була здійснена спроба створити не імітацію розумної поведінки, а реальний розум, хоч і примітивний.

Для розробки розумної поведінки в грі був застосований метод на основі генетичного алгоритму, тому що правила цього алгоритму ідеально інтерпретувалися для законів руху, якими був обмежений персонаж. Це дозволило надати ігровому персонажу з ігровим інтелектом можливість конкурувати з героєм, яким керує користувач. Додаткові анімаційні нюанси при розробці дозволили зробити гру більш ефектною.

В якості цікавості сюжетної лінії гри було використано стиль мультфільму «Simon’s Cat».

Гра розроблена в середовищі Embarcadero Rad Studio 10 Berlin Delphi для мобільних пристроїв під управлянням Android. Гра може бути рекомендована як розвиваюча логічне мислення.

Також було запропоновано та показано, як простими засобами гра може бути переформатована у настільну розвагу. Для цього використовуються мініатюрний робот та Android пристрій, що керує ним за допомогою візуальної інформації через камеру.

Вся робота виконана мною, я самостійно програмувала алгоритми, створювала алгоритми обробки зображень, вивчала можливості програмування для мобільних пристроїв, розроблювала дизайн та вибирала, які компоненти

використовувати, створювала загальну концепцію завдань та конструювала брістлботів.

В якості цікавого досліду можна здійснити порівняння прийняття рішень двох різних персонажів (робота та героя) для досягнення однакової мети.

Створена гра може бути цікава в комерційних проектах, як продукт для розваг і в навчальних або розвиваючих цілях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кращий штучний інтелект в іграх, або Чому II - це підробка [Електронний ресурс]. – Режим доступу: URL: - https://www.igromania.ua/article/29712/Luchshiy_iskusstvennyy_intellekt_v_igrah_ili_Pochemu_II-yeto_poddelka.html

2. Ігровий штучний інтелект [Електронний ресурс]. – Режим доступу: URL: - https://ua.wikipedia.org/wiki/Игровой_искусственный_интеллект

3. Створення штучного інтелекту для ігор - від проектування до оптимізації[Електронний ресурс]. – Режим доступу: URL: - https://habr.com/company/intel/blog/265679/

4. Генетичні алгоритми [Електронний ресурс]. – Режим доступу: URL: - http://www.aiportal.ua/articles/genetic-algorithms/genetic-algorithms.html

5. Класичний генетичний алгоритм. Частина IV. Схрещування, мутація, створення популяції [Електронний ресурс]. – Режим доступу: URL: - http://www.aiportal.ua/articles/genetic-algorithms/classic-alg-part4.html/

6. Аркада (гра) [Електронний ресурс]. – Режим доступу: URL: - https://dic.academic.ua/dic.nsf/wiki/159579

7. Pac-Man [Електронний ресурс]. – Режим доступу: URL: - https://ua.wikipedia.org/wiki/ Pac-Man#Игровой_процесс

8. Алгоритм поведінки привидів у грі Pac-Man [Електронний ресурс]. – Режим доступу: URL: - https://habr.com/post/109406

9. Історія створення Pac-Man [Електронний ресурс]. – Режим доступу: URL: - https://habr.com/company/ruvds/blog/430648/

10. Bristlebot [Електронний ресурс]. – Режим доступу: URL: – https://en.wikipedia.org/wiki/Bristlebot

11. Обертання фігури в 3-х мірному просторі [Електронний ресурс]. – Режим доступу: URL: – http://grafika.me/node/82

12. Матриця повороту [Електронний ресурс]. – Режим доступу: URL: - https://ua.wikipedia.org/wiki/Матрица_поворота

13. FMX.CameraComponent Sample - RAD Studio Code Examples [Електронний ресурс]. – Режим доступу: URL: – http://docwiki.embarcadero.com/CodeExamples/Tokyo/en/FMX.CameraComponent_Sample

14. C2design's Cross Platform Projects [IoT] Bluetooth AC Control Android app with Delphi XE8 and Arduino [Електронний ресурс]. – Режим доступу: http://c2design5sh.blogspot.com/2015/08/BTAcc.html

15. I-Swarm Micro Robots Realized -- Impressive Full-System Integration [Електронний ресурс]. – Режим доступу: URL: – http://www.hizook.com/blog/2009/08/29/i-swarm-micro-robots-realized-impressive-full-system-integration

16. Bristlebot [Електронний ресурс]. – Режим доступу: URL: – https://en.wikipedia.org/wiki/Bristlebot

17. Infrared Remote Controlled (RC) Steerable Vibrobot Created by Naghi Sotoudeh [Електронний ресурс]. – Режим доступу: URL: - http://www.hizook.com/blog/2011/09/08/infrared-remote-controlled-rc-steerable-vibrobot-created-naghi-sotoudeh

18. SOCBOT - the Next Generation Vibrobot [Електронний ресурс]. – Режим доступу: URL: - https://www.instructables.com/id/SOCBOT-the-next-generation-vibrobot/

19. FMX.CameraComponent Sample - RAD Studio Code Examples [Електронний ресурс]. – Режим доступу: URL: - http://docwiki.embarcadero.com/CodeExamples/Tokyo/en/FMX.CameraComponent_Sample

20. BEAM robotics [Електронний ресурс]. – Режим доступу: URL: - https://en.wikipedia.org/wiki/BEAM_robotics

ДОДАТКИ

Додаток А

Особливості поведінки привидів в грі Pac-Man

Червоний привид: його першого потрібно розглядати як загрозу, так як він прокладає найкоротший шлях до Пекмена майже моментально. Його ім'я

«Блінкі» і гра описує його особистість як «тінь». Блінкі майже завжди безпосередньо переслідує Пекмена, до тих пір, поки через недалекоглядність прийняття рішень він не вибере неефективний шлях. Він має одну особливість, якої немає у інших привидів - в двох певних координатах на кожному рівні (в залежності від решти точок) його швидкість збільшується на 5%. Термін зміни швидкості варіюється в залежності від рівня і відбувається все раніше і раніше зі збільшенням прогресу гравця.

Рожевий привид: Його ім'я «Пінкі» і його особистість описується як

«швидкий». Це значна відмінність від Японського опису особистості - machibuse, яке перекладається як «сидить в засідці». Пінкі не рухається швидше за інших привидів, а система пошуку мети намагається перемістити його туди, куди Пекмен направляється, а не там, де він зараз. Він працював, коли Пекмен прямував наліво, вниз або вправо. Але коли Пакмен прямував угору, внаслідок помилки переповнення в коді цільова клітина Пінкі встановлювалася на чотири клітини перед Пакменом і чотири зліва від нього. Одним з важливих наслідків методу вибору мети Пінкі те, що Пекмен часто міг виграти у нього в «курячої» грі (коли два водія йдуть на таран і один з них в останній момент повертає).

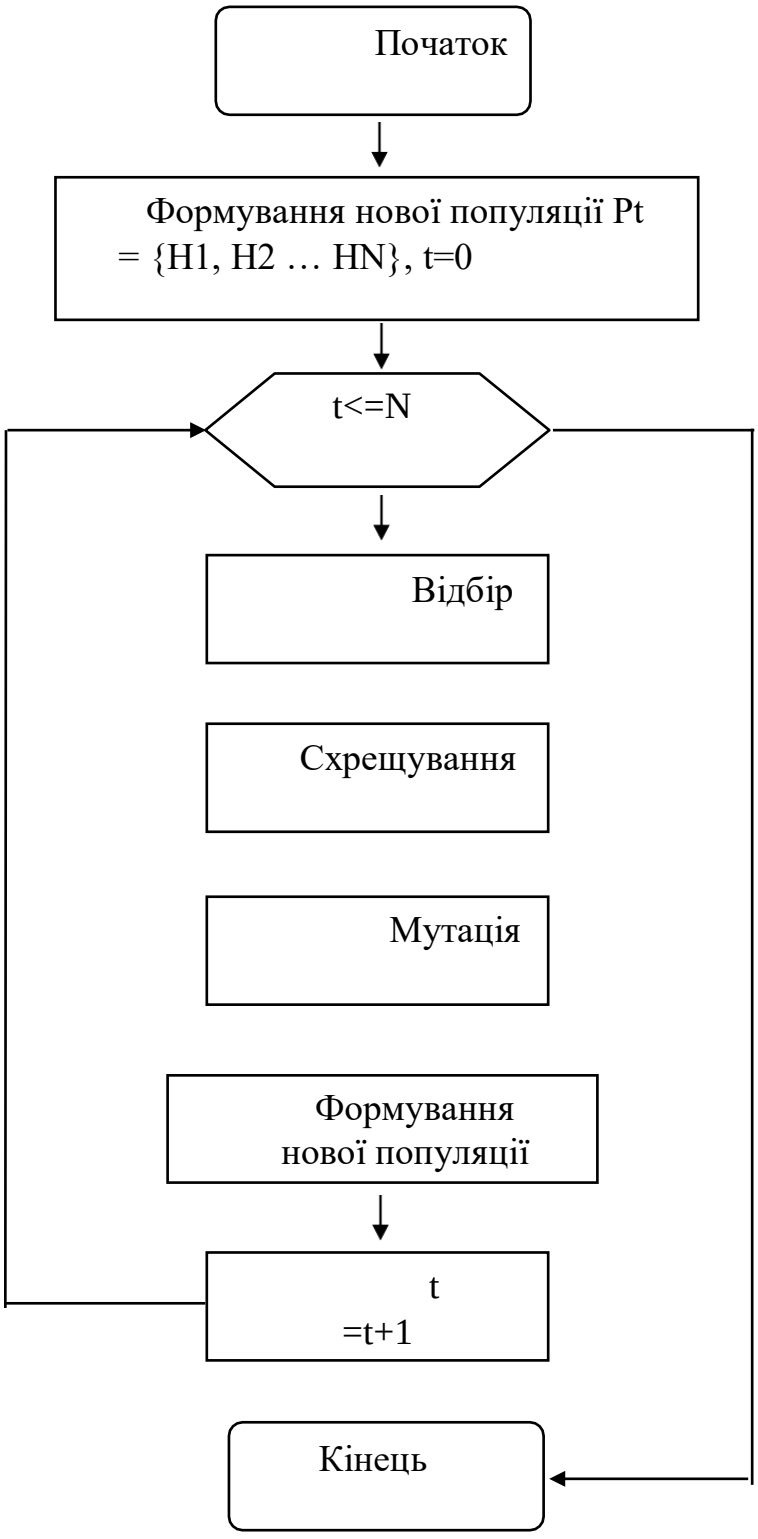
Блакитний привид: Привид на ім'я Інкі залишається в будиночку на короткий час на першому рівні і не кидається в погоню поки Пекмен не з'їсть як мінімум 30 точок. Англійською його описують як «сором'язливий»,

«боязкий». Поведінку Інкі складно передбачити, тому що це єдиний привид, який використовує в своїй гонитві не тільки положення Пекмена. Інкі використовує положення і напрямок як Пекмена, так і Блінкі (червоного привида). Таким чином мета Інки може бути де завгодно, поки Блінкі не буде близько з пекменом, але якщо Блінкі вдалося зблизитися, Інкі зробить те ж саме.

Помаранчевий привид: «Клайд», останній привид, який залишається в будинку найдовше, і не виходить поки як мінімум третина точок не буде з'їдена. Англійською його особистість описується як «тюремник». Особливість Клайда - це два режими, які перемикаються в залежності від його віддаленості від Пекмена. Кожен раз коли Клайд повинен обчислити свою цільову клітку, він спочатку обчислює відстань до Пекмена. Якщо вона більша ніж 8 клітин, то він діє як Блінкі, тобто його метою є сам Пекмен. Однак як тільки його відстань до Пекмена стає менше восьми клітин, його цільова клітина встановлюється там же, де вона була б в режимі розбігання, неподалік від лівого нижнього кута лабіринту. Поєднання цих двох методів і дає те, що Клайд постійно змінює напрямок то до пекмену, то в іншому напрямку.

Додаток Б

Схематичне представлення кроків генетичного алгоритму



Додаток В

Процедура відбору шляху згідно мінімальності відстані до цілі і кількості команд

```
procedure fitness;
var j,i:integer;
  k1:real;
  k2:myarray;
begin
  for j:=1 to (m-1) do
     for i:=1 to (m-j) do
       begin
         if distance[i]>distance[i+1] then
           begin
             k1:=distance[i];
             distance[i]:=distance[i+1];
             distance[i+1]:=k1;

             k2:=population[i];
             population[i]:=population[i+1];
             population[i+1]:=k2;

            k2:=popsize[i];
             popsize[i]:=popsize[i+1];
             popsize[i+1]:=k2;

           end;
       end;


    for j:=1 to (3-1) do
     for i:=1 to (3-j) do
       begin
         if nnnn[i]>nnnn[i+1] then
           begin
             k1:=nnnn[i];
             nnnn[i]:=nnnn[i+1];
             nnnn[i+1]:=k1;

             k2:=population[i];
             population[i]:=population[i+1];
             population[i+1]:=k2;

            k2:=popsize[i];
             popsize[i]:=popsize[i+1];
             popsize[i+1]:=k2;

             k1:=distance[i];
             distance[i]:=distance[i+1];
             distance[i+1]:=k1;
           end;
       end;
end;
```

Додаток Г

Процедура схрещення

```
procedure Cross(n1,n2:integer) ;
var  i,r: Integer;
begin
 randomize;
   if random<0.5 then
     begin
       way1:=population[n1];
       size1:=popsize[n1];
       way2:=population[n2];
        size2:=popsize[n2];
     end
   else
     begin
       way2:=population[n1];
        size2:=popsize[n1];
       way1:=population[n2];
        size1:=popsize[n2];
     end;
 for i := 1 to n do
  begin
    if odd(i) then begin way3[i]:=way2[i]; size3[i]:=size2[i]; end
    else begin way3[i]:=way1[i];   size3[i]:=size1[i]; end;
  end;
 r:=random(n div 2)+(n div 2)+1;
 if random<0.5 then way3[r]:='R' else way3[r]:='L';
 if random<0.5 then size3[r]:='0' else size3[r]:='R';
  r:=random(n div 2)+(n div 2)+1;
 if random<0.5 then way3[r]:='R' else way3[r]:='L';
 if random<0.5 then size3[r]:='0' else size3[r]:='R';
```

```
   r:=random(n div 2)+(n div 2)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
  if random<0.5 then size3[r]:='0' else size3[r]:='R';
    r:=random(n div 2)+(n div 2)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
  if random<0.5 then size3[r]:='0' else size3[r]:='R';
   r:=random(n div 2)+(n div 2)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
  if random<0.5 then size3[r]:='0' else size3[r]:='R';
  r:=random(n)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
   if random<0.5 then size3[r]:='0' else size3[r]:='R';
     r:=random(n)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
    if random<0.5 then size3[r]:='0' else size3[r]:='R';
     r:=random(n)+1;
  if random<0.5 then way3[r]:='R' else way3[r]:='L';
    if random<0.5 then size3[r]:='0' else size3[r]:='R';
end;
```

Додаток Ґ

Процедура створення нової популяції

```
procedure TForm1.Button5Click(Sender: TObject);
begin
k:=0;
for k := 1 to kkkkk do
begin
application.ProcessMessages;
  R:=25;
 for j:=1 to m do   begin
 hhh:=0;
 findplace(j);
 end;
 fitness;
 memo1.Text:='';
 for j:=1 to m do
 Memo1.Lines.Add(population[j]+'#
'+floattostr(distance[j]));
  cross(1,2);
 population[4]:=way3;
 popsize[4]:=size3;
 cross(1,2);
 population[5]:=way3;
 popsize[5]:=size3;
 cross(1,3);
 population[6]:=way3;
 popsize[6]:=size3;
 memo1.Text:='';
 for j:=1 to m do     begin
 Memo1.Lines.Add(population[j]+'#
'+floattostr(distance[j]));
  Memo1.Lines.Add(popsize[j]+'#
'+floattostr(nnnn[j]));
 end;
 label1.Text:=inttostr(k);
end;
k:=kkkkk;
hhh:=1;
findplace(2);
findplace(3);
findplace(1);
 x_current:=x_end;
 y_current:=y_end;
 dir_current:=direct;
 for i:=1 to n do
     begin
       if random<0.5 then way[i]:='L' else way[i]:='R';
       if random<0.5 then size[i]:='0' else size[i]:='R';
     end;
end;
```

Додаток Д

Процедура знаходження пристосованості хромосоми

```
procedure FindPlace(i:integer);
var
  j,kk: Integer;
begin
  direct:=dir_current;
  y0:=y_current;
  x0:=x_current;
  y_end:=y0;
  x_end:=x0;
  way:=population[i];
  size:=popsize[i];
  nnnn[i]:=0;
  for j:=1 to n do
    begin
      com1:=way[j];
      x0:=x_end;
      y0:=y_end;
     if (size[j]='0')  then  begin
      R:=0;
       end
       else begin R:=25;     end;
   if R<>0 then begin
      if (direct='D') then
        begin
          if com1='L' then
            begin
              x_end:=x0+R;
              y_end:=y0+R;
              direct1:='R';
            end;
          if com1='R' then
            begin
              x_end:=x0-R;
              y_end:=y0+R;
              direct1:='L';
            end;
        end;
      if (direct='U') then
        begin
          if com1='L' then
            begin
              x_end:=x0-R;
              y_end:=y0-R;
              direct1:='L';
            end;
          if com1='R' then
            begin
              x_end:=x0+R;
              y_end:=y0-R;
              direct1:='R';
            end;
        end;
      if (direct='L') then
        begin
          if com1='L' then
            begin
              x_end:=x0-R;
              y_end:=y0+R;
              direct1:='D';
            end;
          if com1='R' then
            begin
              x_end:=x0-R;
              y_end:=y0-R;
```

```
          direct1:='U';
        end;
      end;
    if (direct='R') then
      begin
        if com1='L' then
          begin
            x_end:=x0+R;
            y_end:=y0-R;
            direct1:='U';
          end;
        if com1='R' then
          begin
            x_end:=x0+R;
            y_end:=y0+R;
            direct1:='D';
          end;
      end;
      form1.Rectangle1.Position.x:=x0;
      form1.Rectangle1.Position.y:=y0;
      allow:=0;
      for kk := 1 to 8 do
      begin
      dblock:=sqrt(sqr(x_end-block[kk,1])+sqr(y_end-
block[kk,2]));
      if dblock>40 then    allow:=allow+1;
      end;
    if allow/8>=1 then
        begin
          drawe(direct,com1,R,j,i);
          direct:=direct1;
          nnnn[i]:=nnnn[i]+1;
        end
      else
        begin
          x_end:=x0;
          y_end:=y0;
          end;
        end;
      end;
    end;
  distance[i]:=sqrt(sqr(x_end-
form1.circle1.Position.X)+sqr(y_end-
form1.circle1.Position.Y));
  form1.rectangle1.Position.X:=x_end;
  form1.rectangle1.Position.Y:=y_end;
end;
```

Додаток Е

Скетч для плати для управління роботом з мобільного пристрою.

```
int val;
int MOTOR1 = 6;
int MOTOR2 =8;
void setup()
{
  Serial.begin(9600);
  pinMode(MOTOR1, OUTPUT);
  digitalWrite(MOTOR1, LOW);
  pinMode(MOTOR2, OUTPUT);
  digitalWrite(MOTOR2, LOW);
}
void loop()
{
 if (Serial.available())
  {
    val=Serial.read();
    // При символі "1" включаємо перший моторчик
    if (val == '1')
    {
      digitalWrite(MOTOR1, HIGH);
     // Моторчтк працює визначений час, а потім відключається
      delay(3150);
      digitalWrite(MOTOR1, LOW);
    }

    // Аналогічно для другого моторчику
    // При символі "0" вмикаємо другий моторчик
    if ( val == '2')
    {
      digitalWrite(MOTOR2, HIGH);
     // Моторчик працює визначений час, а потім відключається
      delay(3250);
      digitalWrite(MOTOR2, LOW);
    }
  }
}
```

Додаток Є

Принципова схема робота на аналогових компонентах