Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Інформаційних управляючих систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Розробка модуля «Оптимізація поведінки неігрових персонажів» інформаційної ігрової системи

(тема)

Виконав:
здобувач 4 року навчання,
групи ІТУ-21-2
Денис СКРИНИЧЕНКО
(власне ім'я, прізвище)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні технології управління
(повна назва освітньої програми)

Керівник: доц. каф. ІУС Олена ДОЛЯ
(посада, власне ім'я, прізвище)

Допускається до захисту

Зав. кафедри ІУС _____ Костянтин ПЕТРОВ
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні технології управління
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

" 19 " травня 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Скриниченку Денису Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка модуля «Оптимізація поведінки неігрових персонажів» інформаційної ігрової системи

затверджена наказом по університету від " 19 " травня 2025 р. № 370Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії " 15 " червня 2025 р.

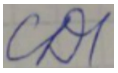
3. Вихідні дані до роботи Інформаційна ігрова система у середовищі розробки Unity, інформація про існуючі методи оптимізації поведінки неігрових персонажів, вимоги до модуля «Оптимізація поведінки неігрових персонажів»

4. Перелік питань, що потрібно опрацювати у роботі Огляд і аналіз існуючих методів оптимізації поведінки неігрових персонажів, опис методу навчання з підкріпленням, опис архітектури модуля оптимізації поведінки неігрового персонажа, створення діаграми потоків даних, діаграми компонентів, діаграми класів, діаграми послідовностей для графічного представлення архітектури модуля, діаграми декомпозиції процесу навчання з підкріпленням, розробка й обґрунтування елементів програмної забезпечуючої системи

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд і аналіз сучасного стану розглянутої проблеми та існуючих методів і засобів вирішення задач кваліфікаційної роботи	19.05.2025 - 20.05.2025	Виконано
2	Формулювання завдання оптимізації поведінки неігрового персонажа	21.05.2025 - 22.05.2025	Виконано
3	Опис архітектури модуля оптимізації поведінки неігрового персонажа	23.05.2025 - 25.05.2025	Виконано
4	Розробка й обгрунтування елементів програмної забезпечуючої системи	26.05.2025 - 01.06.2025	Виконано
5	Розробка User Experience (UX) та User Interface (UI) рішень	02.06.2025 - 03.06.2025	Виконано
6	Тестування та оцінка надійності функціонування неігрового персонажа	04.06.2025 - 05.06.2025	Виконано
7	Оформлення пояснювальної записки	06.05.2025 - 10.06.2025	Виконано
8	Захист кваліфікаційної роботи	18.06.2025	Виконано

Дата видачі завдання 19 травня 2025 р.

Здобувач  _____
(підпис)

Керівник роботи _____ доц. каф. ІУС Олена ДОЛЯ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 63 с., 17 рис., 0 табл., 1 дод., 8 джерел.

АГЕНТ, ІНФОРМАЦІЙНА ІГРОВА СИСТЕМА, МАШИННЕ НАВЧАННЯ, НАВЧАННЯ З ПІДКРІПЛЕННЯМ, НЕІГРОВИЙ ПЕРСОНАЖ, ОПТИМІЗАЦІЯ ПОВЕДІНКИ, UNITY.

Сучасні ігрові рушії є потужними інструментами для моделювання світів зі складною фізикою та складною взаємодією між агентами з різними можливостями. В наш час відбувається дуже багато експериментів і спроб знаходження нових способів взаємодії між гравцем і неігровими персонажами.

Об’єктом дослідження кваліфікаційної роботи є застосування методів машинного навчання для оптимізації поведінки неігрових персонажів інформаційних ігрових систем. Основним методом дослідження є використання навчання з підкріпленням за допомогою бібліотеки ML-Agents ігрового рушія Unity.

Метою кваліфікаційної роботи є розробка модуля «Оптимізація поведінки неігрових персонажів» інформаційної ігрової системи жанру «три в ряд», дослідження застосування методу навчання з підкріпленням для оптимізації поведінки неігрових персонажів в інформаційних ігрових системах, порівняння його з іншими існуючими методами оптимізації поведінки.

У результаті виконання кваліфікаційної роботи, успішно створено і проведено тренування неігрового персонажа за допомогою навчання з підкріпленням, який досяг середніх результатів. На основі отриманих даних і досвіду, гарною рекомендацією вважається застосування не одного методу машинного навчання, а поїднання декількох методів, а також проведення тренування агента на сучасному комп’ютері.

ABSTRACT

Master’s thesis: 63 pages, 17 figures, 0 tables,1 appendix, 8 sources.

AGENT, BEHAVIOR OPTIMIZATION, GAME INFORMATION SYSTEM, MACHINE LEARNING, NON-PLAYABLE CHARACTER, REINFORCEMENT LEARNING, UNITY.

Modern game engines are powerful tools for modeling worlds with complex physics and complex interactions between agents with different capabilities. Nowadays, there is a lot of experimentation and attempts to find new ways of interacting between the player and non-player characters.

The object of the research of the qualification work is the application of machine learning methods to optimize the behavior of non-game characters in information game systems. The main research method is the application of reinforcement learning using the ML-Agents library in the Unity game engine.

The purpose of the qualification work is to develop the module «Optimization of the behavior of non-game characters» of the information game system of the «match three» genre, to study the application of the reinforcement learning method to optimize the behavior of non-playable characters in information game systems, and to compare it with other existing methods of behavior optimization.

As a result of the qualification work, a non-playable character was successfully created and trained using reinforcement learning, which achieved average results. Based on the data and experience obtained, a good recommendation is to use not one machine learning method, but several methods, as well as to train the agent on a modern computer.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ДСТУ – державний стандарт України

ІС – інформаційна система

ІТ – інформаційна технологія

ПЗ – програмне забезпечення

ML – machine learning

NPC – non-playable character

RL – reinforcement learning

UI – user interface

UX – user experience

ВСТУП

Неігрові персонажі відіграють важливу роль у відеоіграх, створюючи динамічне ігрове середовище, в якому може бути багато різних дружніх, нейтральних чи ворожих до гравця персонажів. Оптимізація їхньої поведінки необхідна для досягнення правильних і стабільних результатів виконання коду, задуманих розробниками, але такі персонажі зазвичай є передбачуваними та мають обмежений набір дій. Це відбувається тому, що стандартні методи оптимізації поведінки не можуть підготувати неігрових персонажів до всіх варіантів подій у грі, які неможливо заздалегідь прописати в коді. Тому, вони часто поводяться нереалістично чи повторюють свої дії, а деякі задачі не можуть виконати взагалі.

З появою методів машинного навчання з’явилися нові підходи до покращення поведінки неігрових персонажів. Зокрема, використання навчання з підкріпленням дозволяє створювати адаптивних персонажів, які можуть самостійно приймати правильні рішення на основі власного досвіду, що робить їх більш універсальними та непередбачуваними. Багато світових ігрових студій почали експериментувати з такими методами, що дозволяє підвищити реалістичність ігрового процесу та залученість гравців [1].

Застосування методів машинного навчання в ігровій індустрії є актуальним напрямом досліджень, оскільки він дозволяє значно покращити якість взаємодії між гравцем і неігровими персонажами. Це дає змогу створювати більш складних і реалістичних персонажів у будь-яких жанрах відеоігор. У майбутньому це може призвести до створення повністю автономних агентів навіть у складних реалістичних іграх.

Метою розробки модуля «Оптимізація поведінки неігрових персонажів» є дослідження застосування методу навчання з підкріпленням для оптимізації поведінки неігрових персонажів в інформаційних ігрових системах, порівняння його з іншими існуючими методами оптимізації поведінки.

1 ОПИС ТА АНАЛІЗ СТРУКТУРНИХ І ФУНКЦІОНАЛЬНИХ ОСОБЛИВОСТЕЙ КОМПАНІЇ З РОЗРОБКИ МОБІЛЬНИХ ІГОР

1.1 Опис особливостей функціонування об’єкту автоматизації

Об’єктом дослідження виступає компанія з розробки мобільних ігор, яка спеціалізується на розробці ігор різних жанрів, зокрема «Три в ряд». Компанія складається з 30 працівників, які працюють над розробкою продукту в центральному офісі у Києві, а публікацією та просуванням готових ігор на глобальному ринку займається партнер – міжнародна компанія з видавництва ігор.

Розробка ігор в даній компанії починається з етапу планування дизайну гри, ключових механік, створення сюжету і персонажів, та вибору інструментів розробки. Далі, створюється 2D і 3D графіка та анімації для всіх візуальних елементів гри. Програмування починається із побудови основної архітектури проєкту, далі реалізуються основні ігрові механіки, потім – поведінка NPC, створення різних рівнів, та інше. Постійно відбувається тестування, знаходження і усунення помилок в коді, оптимізація та рефакторинг коду, поки не буде досягнуто фінальної версії гри. Після публікації ігор, компанія ще може їх оновлювати, додаючи щось нове, при відповідному запиті від компанії-видавця.

Об’єктом автоматизації є процес оптимізації поведінки неігрового персонажа (далі – NPC). На прикладі гри у жанрі «Три в ряд», у даній роботі досліджується застосування методу навчання з підкріпленням для оптимізації поведінки NPC. Ефективне застосування даного методу дозволить NPC використовувати ефективні та непередбачувані стратегії, що зробить його більш складним та цікавим суперником, та підвищіть популярність гри серед користувачів.

1.2 Опис організаційної структури

Організаційна структура компанії з розробки мобільних ігор містить такі ключові посади:

– головний виконавчий директор: відповідає за загальне управління компанією, стратегічний розвиток та прийняття ключових бізнес-рішень;

– технічний директор: контролює технічний напрямок розробки, відповідає за вибір технологій, архітектуру програмних рішень та ефективність команди розробників;

– артдиректор: керує художнім відділом, відповідає за створення візуального стилю гри, якість графічних елементів та їхню відповідність ігровій концепції;

– продюсер: керують створенням персонажів, ігрового світу та сюжету гри, обирають ігрові механіки та як вони працюватимуть, аналізують конкуруючі продукти на ринку;

– керівник з розвитку бізнесу: знаходить видавця для гри, контролює гроші, займається пошуком партнерств, інвесторів та стратегічних можливостей для розширення компанії.

Схема організаційної структури компанії-розробника мобільних ігор представлена на рисунку 1.1.

Рисунок 1.1 – Схема організаційної структури компанії-розробника мобільних ігор

1.3 Опис методу навчання з підкріпленням

Для навчання оптимізації поведінки NPC в рамках даної роботи використовується метод навчання з підкріпленням (англ. reinforcement learning, далі – RL). Цей метод дозволяє створити автономного агента, який може приймати рішення та діяти відповідно до свого середовища без залежності від прямих інструкцій в програмному коді. Він покращує свої дії на основі системи винагород, протягом всього часу навчання, яке може тривати багато годин, в залежності від складності завдань.

Основні етапи навчання включають:

– спостереження (observation): агент отримує певну інформацію про поточний стан ігрового середовища (поля). В коді зазначається яку саме інформацію отримує агент;

– прийняття рішення (decision): на основі отриманих даних агент сам

обирає оптимальну дію (обирається випадково, якщо це перший цикл);

– дія (action): агент виконує обрану дію. Це команди в коді, написані програмістом, які можуть відрізнятися в залежності від того, яку дію обрав агент. Дія певним чином змінює стан ігрового середовища (поля);

– призначення винагороди (reward): агент отримує позитивну чи негативну винагороду залежно від ефективності свого ходу. Яку саме винагороду і за якої умови отримує агент зазначається в коді;

– оновлення політики (policy update): на основі отриманих даних агент покращує свою стратегію гри та оновлює файл із моделлю агента [3].

Візуальна модель процесу навчання агента методом навчання з підкріпленням зображена на рисунках 1.2 та 1.3.

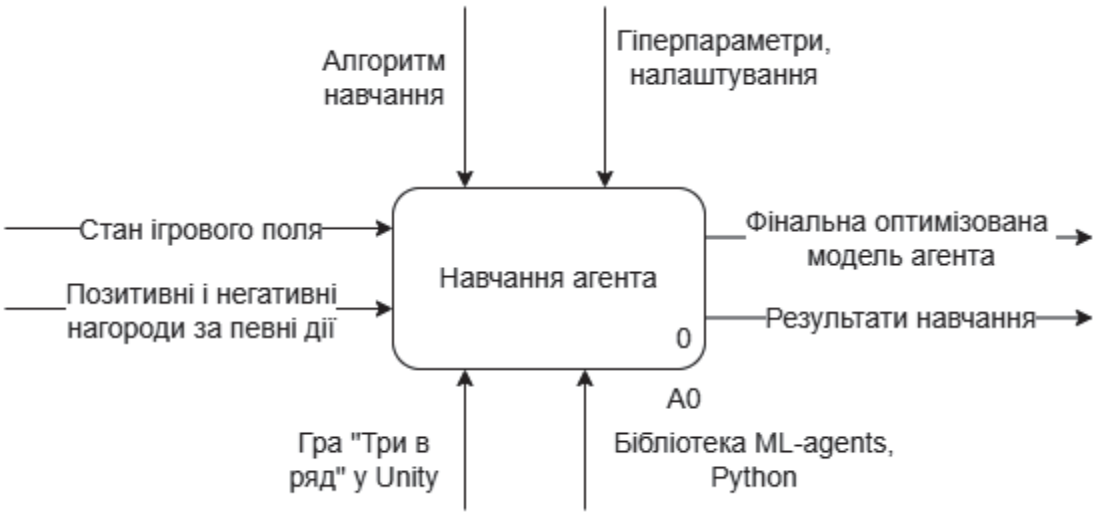


Рисунок 1.2 – Контекстна діаграма процесу навчання агента методом навчання з підкріпленням

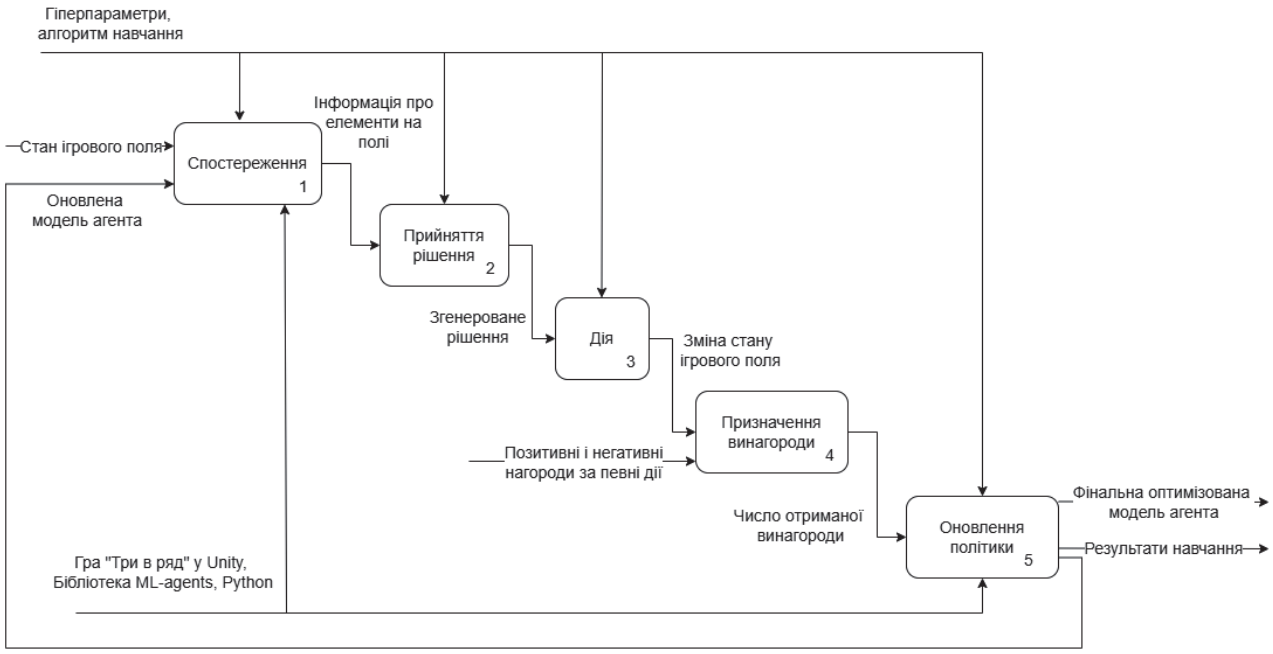


Рисунок 1.3 – Діаграма декомпозиції першого рівня процесу навчання агента методом навчання з підкріпленням

Для використання методу RL в ігровому рушії Unity використовується бібліотека Machine Learning (далі – ML) Agents, яка є відкритим офіційним проектом від Unity. Бібліотека активно підтримується та оновлюється, має документацію та спільноту розробників. ML-Agents дозволяє створювати і навчати автономних агентів в Unity різними методами машинного навчання, а також налаштовувати різні параметри агентів, наприклад тип дії для алгоритму (цілі чи дробові числа та їх діапазон), гіперпараметри (параметри для керування навчанням моделі). Навчених агентів можна використовувати для багатьох цілей, навіть для суперництва одних агентів з іншими, або ж для автоматизованого тестування ігор [4].

2 ОГЛЯД І АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ОПТИМІЗАЦІЇ ПОВЕДІНКИ НЕІГРОВИХ ПЕРСОНАЖІВ

Найпростіший метод оптимізації поведінки NPC – це дерево прийняття рішень (decision tree). Ця структура використовується для обчислення значення цільової змінної на основі декількох змінних на вході. Тобто, персонаж може обирати свою наступну дію на основі деяких даних із середовища гри, а реалізується це за допомогою if-else структур у програмному коді. Цей метод є найбільш популярним, бо підходить для більшості простих задач.

Набір станів (state machine) – це модель обчислень, яка описує систему як набір станів і переходів між ними. Тобто, персонаж може мати тільки один активний стан (набір дій), а за певних умов буде переходити на інший. Такий підхід добре підходить, коли персонаж простий і не має великої кількості різних станів. З великими наборами станів стає важко додавати щось нове та проводити налагодження [5]. Наприклад, NPC може перемикатись від стану «патрулювання», до стану «переслідування гравця», а потім до «атака гравця», під час яких виконує чіткі інструкції з програмного коду.

Дерево поведінки (behaviour tree) ітеративно проходить через внутрішні вузли, поки не досягне дії. Кожен раз дерево повторно оцінюється, щоб досягти дії. Цей підхід досить складний, але більш гнучкий у перемиканні між діями та забезпечує більшу модульність і масштабованість. Тобто, його краще використовувати для складних NPC, які мають багато різних станів чи умов для їх зміни [5].

Бібліотека ML-Agents дозволяє використовувати не тільки метод навчання з підкріпленням. Іншими популярними методами машинного навчання є імітаційне навчання (imitation learning), гра проти себе (self-play), навчання за планом (curriculum learning), Ієрархічне RL (hierarchical RL), застосування датчика камери (Camera sensor) та ін.

Імітаційне навчання (imitation learning) дозволяє агенту навчатися шляхом

наслідування поведінки гравця. Тобто, після певних налаштувань, розробник запускає гру і вручну керує агентом, виконуючи необхідні дії в ігровому середовищі. Ці дії автоматично записуються і зберігаються у файл у проєкті гри. Зібрані дані використовуються для початкового формування політики агента, значно прискорюючи процес навчання, бо агент одразу починає виконувати правильні дії. Далі, агент вдосконалює свою політику за допомогою стандартного RL, щоб його дії стали більш універсальними і стабільними.

Метод гри проти себе (self-play) передбачає, що агент навчається, граючи проти своєї ж моделі, або проти іншого агента. Цей підхід зазвичай треба використовувати, якщо у контексті певної гри неможливо запустити тренування лише одного агента. Наприклад, якщо NPC має відповідати на різні дії гравця, а не працювати самому, то слід створювати self-play, в якому гравець замінюється іншим агентом. Тренування агента із гравцем теж можна проводити, але це буде дуже довго і неефективно, тож метод гри проти себе важливий ще й тому, що сильно пришвидшує процес тренування, адже агенти виконують дії миттєво, без довгого обмірковування.

Під час використання навчання за планом (curriculum learning) агент спочатку вчиться ефективно виконувати прості дії, а потім поступово переходить до тренування з більш складними завданнями. Цей метод досить простий, але в певних ігрових середовищах, де його можна зручно застосувати, є гарним доповненням до стандартного RL.

Ієрархічне RL дозволяє агенту вивчати кілька рівнів політик, кожен з яких відповідає різним рівням абстракції. Тобто, У HRL завдання розбиваються на підзадачі, і ці підзадачі можна додатково декомпонувати, якщо необхідно. Кожен рівень ієрархії зосереджений на вирішенні конкретного аспекту загального завдання, полегшуючи для агента вивчення та оптимізацію політик на різних рівнях абстракції. Цей ієрархічний підхід не тільки спрощує процес навчання, але й покращує масштабованість і ефективність алгоритмів RL [6].

Компонент датчик камери (Camera sensor) з бібліотеки ML-Agents – це датчик, який використовує камеру з ігрового середовища в Unity для створення

візуальних спостережень для агента. Тобто, в режимі реального часу, зображення з камери перетворюються у зменшену копію зображення необхідного розміру. Далі, агент використовує це зображення як вхідні дані (спостереження), замість звичайних числових значень, та вчиться розпізнавати закономірності в пікселях і на основі цього приймати правильні дії.

Всі ці та інші методи машинного навчання також можна поєднувати між собою. Наприклад, розробники RL агента для гри-стратегії в реальному часі «StarCraft 2» зазначили, що використали імітаційне навчання, щоб навчити агента різним стратегіям для ефективного початку гри, а також метод self-play, під час якого єдиною ціллю другого агента було викрити слабкі сторони основного агента. Таким чином, основний агент перестав виконувати лише одну стратегію, а став грати більш різноманітно і став краще відповідати на різні стратегії противника [7].

3 ОПИС ВИМОГ ДО ОБ’ЄКТА РОЗРОБКИ. ФОРМУЛЮВАННЯ ЗАВДАННЯ РОЗРОБКИ

3.1 Опис функціональних вимог до об'єкта розробки

Розробка модуля «Оптимізація поведінки неігрового персонажа» у грі «Три в ряд» передбачає створення агента за допомогою бібліотеки ML-Agents у проєкті гри ігрового рушія Unity. Агента необхідно навчати за допомогою методу навчання з підкріпленням, а безпосередньо у грі використовувати натреновану модель поведінки для гри проти гравця, виконуючи ходи по черзі. Коли ходи NPC та гравця закінчуються, визначається переможець за допомогою набраних очок та кількості зібраних фруктів певного кольору. Основні функціональні вимоги включають:

– правильне налаштування спостережень, щоб агент отримував достовірну інформацію про поточний стан ігрового поля (розмір ігрового поля, відповідні значення для різних об’єктів на полі);

– правильна обробка спостережень, щоб агент правильно визначав доступні ходи, які призведуть до створення комбінацій;

– виклик функції прийняття рішення у правильний момент, тільки коли всі фрукти на ігровому полі перестали рухатись та опинились на своїх місцях;

– повноцінне виконання дії, яке внесе зміни у ігрове поле та рахунок (score) агента, а також призначить йому певну нагороду;

– використання згенерованого файлу навченої моделі у Unity, та коректне налаштування режиму гри проти NPC, в якому кожен з учасників має всього 30 ходів, а після використання 15 з них, хід переходить опоненту.

3.2 Опис нефункціональних вимог до об'єкта розробки

Агент мусить тренуватися самостійно, та після використання всіх ходів ігрове поле необхідно створювати заново, всі змінні повертати до початкових значень. Тобто, модуль не повинен викликати жодних помилок, щоб тренування проходило стабільно декілька годин.

Гра повинна працювати зі стабільним показником частоти кадрів більше 60, та без підвищеного використання CPU/GPU.

Версію проєкту гри для тренування слід експортувати у форматі серверної збірки (server build), щоб навчання можна було проводити одразу на декількох сутностях гри, для прискорення процесу. Також, необхідне завантаження відповідних бібліотек для навчання агента: python pip, pytorch, ML-Agents. Під час навчання, використання оперативної пам’яті кожною сутністю гри не має перевищувати 700 МБ.

3.3 Обґрунтування мети і критеріїв ефективності об'єкта розробки

Метою розробки модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи жанру «Три в ряд» в Unity є навчання NPC виконувати оптимальні ходи (збирати фрукти певного типу, шукати комбінації з великою кількістю фруктів, використовувати спеціальні об’єкти на полі), щоб продемонструвати можливість створення унікального і непередбачуваного суперника для гравця.

Критерії ефективності модуля «Оптимізація поведінки неігрового персонажа»: суттєвий ріст в отриманні агентом винагород під час навчання, та відсоток перемог проти гравця при тестуванні готової моделі має бути більше 40 відсотків, а агент має виконувати переважно ефективні ходи.

4 ОПИС АРХІТЕКТУРИ МОДУЛЯ «ОПТИМІЗАЦІЯ ПОВЕДІНКИ НЕІГРОВОГО ПЕРСОНАЖА»

Об’єктом розробки і дослідження є модуль «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи жанру «Три в ряд». Розробка відбувається за допомогою ігрового рушія Unity, мовою програмування C#. Розробка модуля поділяється на 2 етапи: навчання агента та використання готової моделі у грі. В цьому розділі представлені діаграми, що відображають роботу модуля під час фінального етапу – використання готової моделі у грі, де гравець і NPC виконують ходи по черзі. Під час навчання агента, хід гравця завжди пропускається, а після використання всіх ходів агента, гра перезапускається, щоб навчання тривало автоматично.

Контекстна діаграма та діаграма декомпозиції першого рівня модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи зображені відповідно на рисунках 4.1 та 4.2.

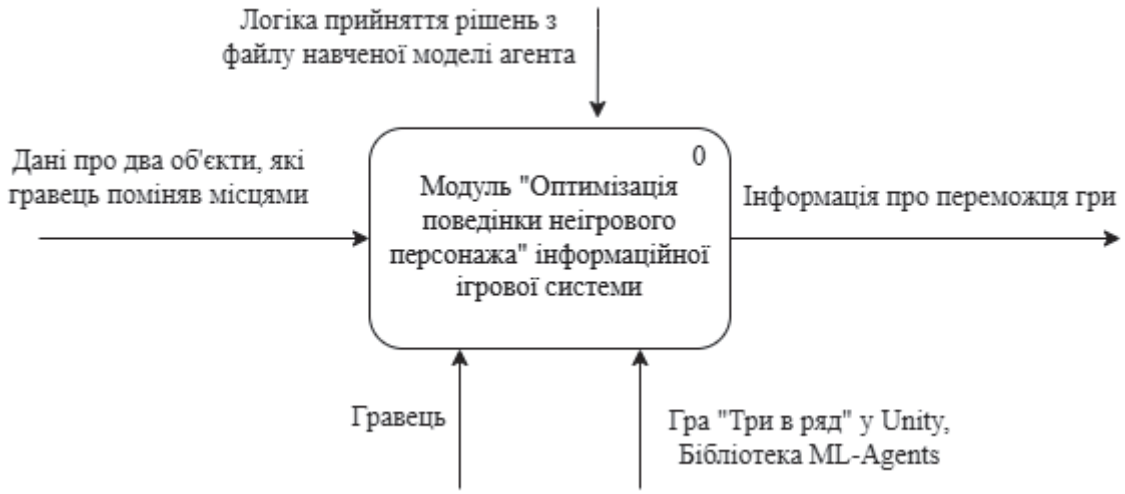


Рисунок 4.1 – Контекстна діаграма модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

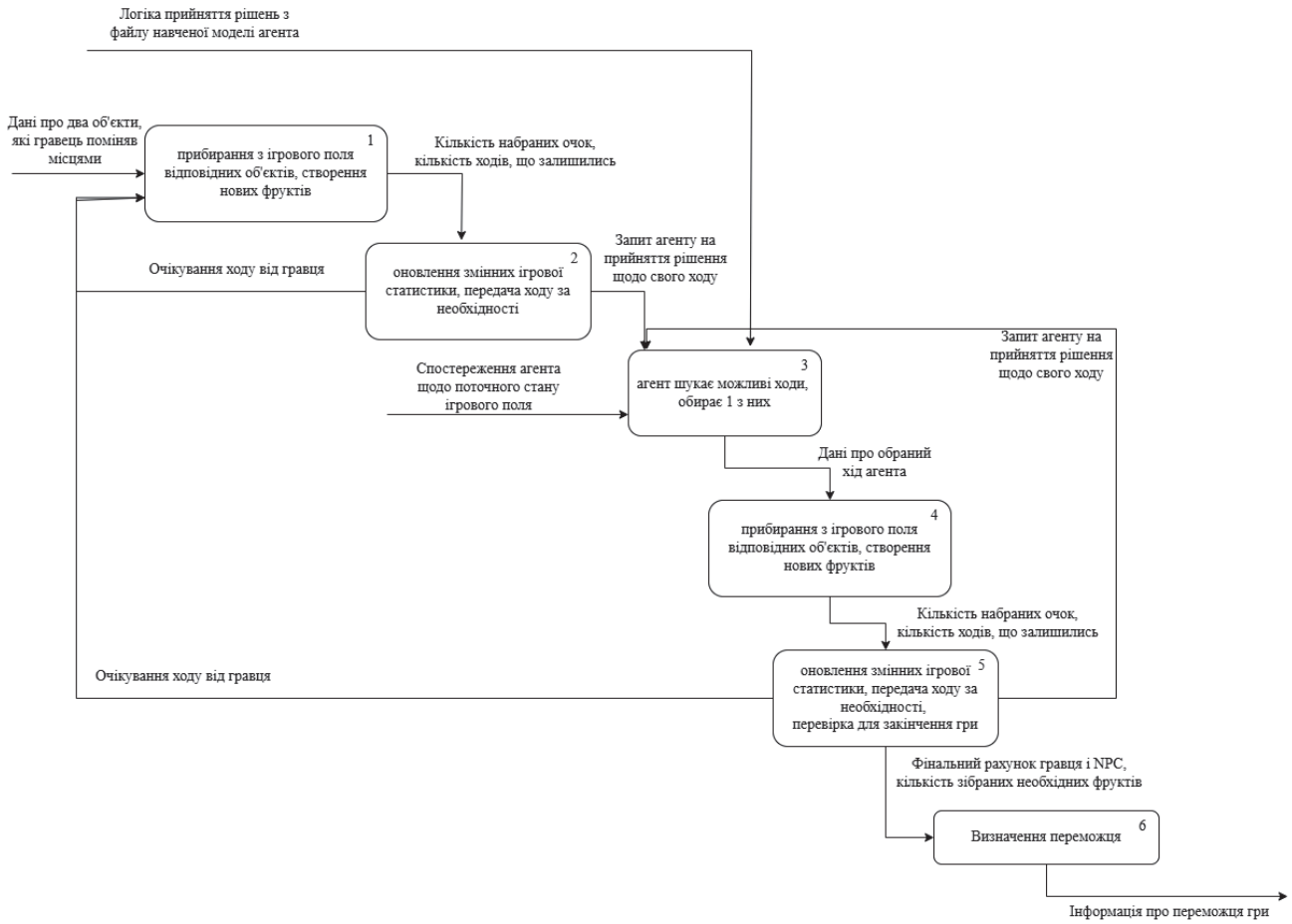


Рисунок 4.2 – Діаграма декомпозиції першого рівня модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

Гравець, гра «Три в ряд» у Unity, та бібліотека ML-Agents безпосередньо використовуються на всіх етапах, зазначених на рисунку 4.2. Очікування ходу гравця – етап, коли гравець обмірковує свій хід, та рухає певний об’єкт на ігровому полі. Також, відбувається перевірка щодо того, чи було створено комбінацію з трьох чи більше однакових фруктів, та наступні етапи запускаються тільки після такого правильного ходу. NPC завжди виконує тільки правильні ходи, які створюють комбінації, а також миттєво приймає рішення щодо свого ходу. Як і було зазначено у вимогах до об’єкта розробки, гравець виконує 15 ходів, далі NPC виконує свої 15 ходів, потім знову відбувається по 15 ходів від кожного учасника, та гра закінчується визначенням переможця.

Діаграма потоків даних модуля «Оптимізація поведінки неігрового персонажа» для інформаційної ігрової системи «Три в ряд» представлена на рисунку 4.3. На діаграмі зображено лише основні потоки даних, які відповідають за почергове виконання ходів гравцем і агентом. Обробка результатів зміни фруктів місцями відбувається однаково під час ходу гравця, та під час ходу NPC.

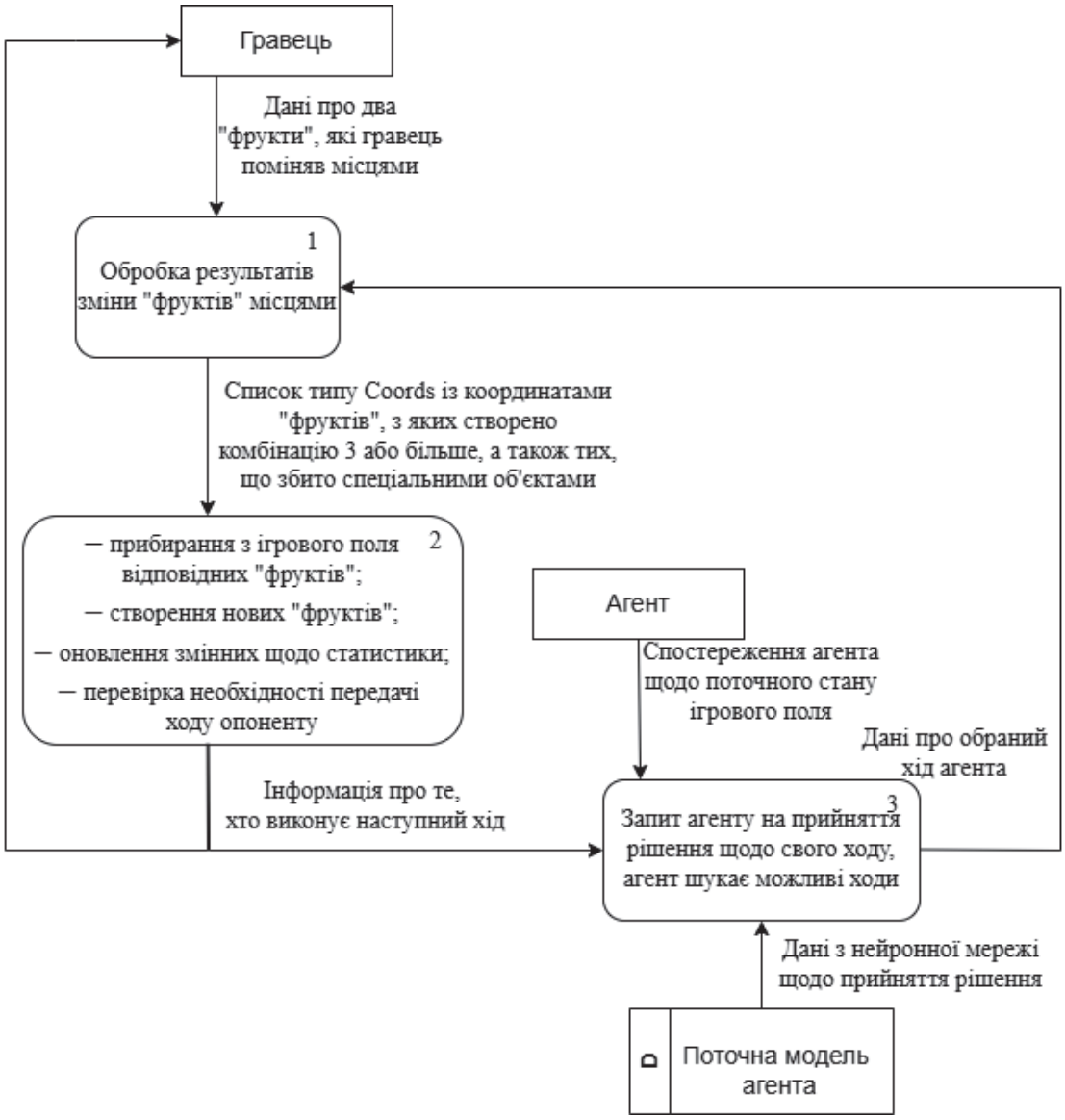


Рисунок 4.3 – Діаграма потоків даних модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

Діаграма компонентів демонструє основні зовнішні зв’язки між різними класами (рисунок 4.4).

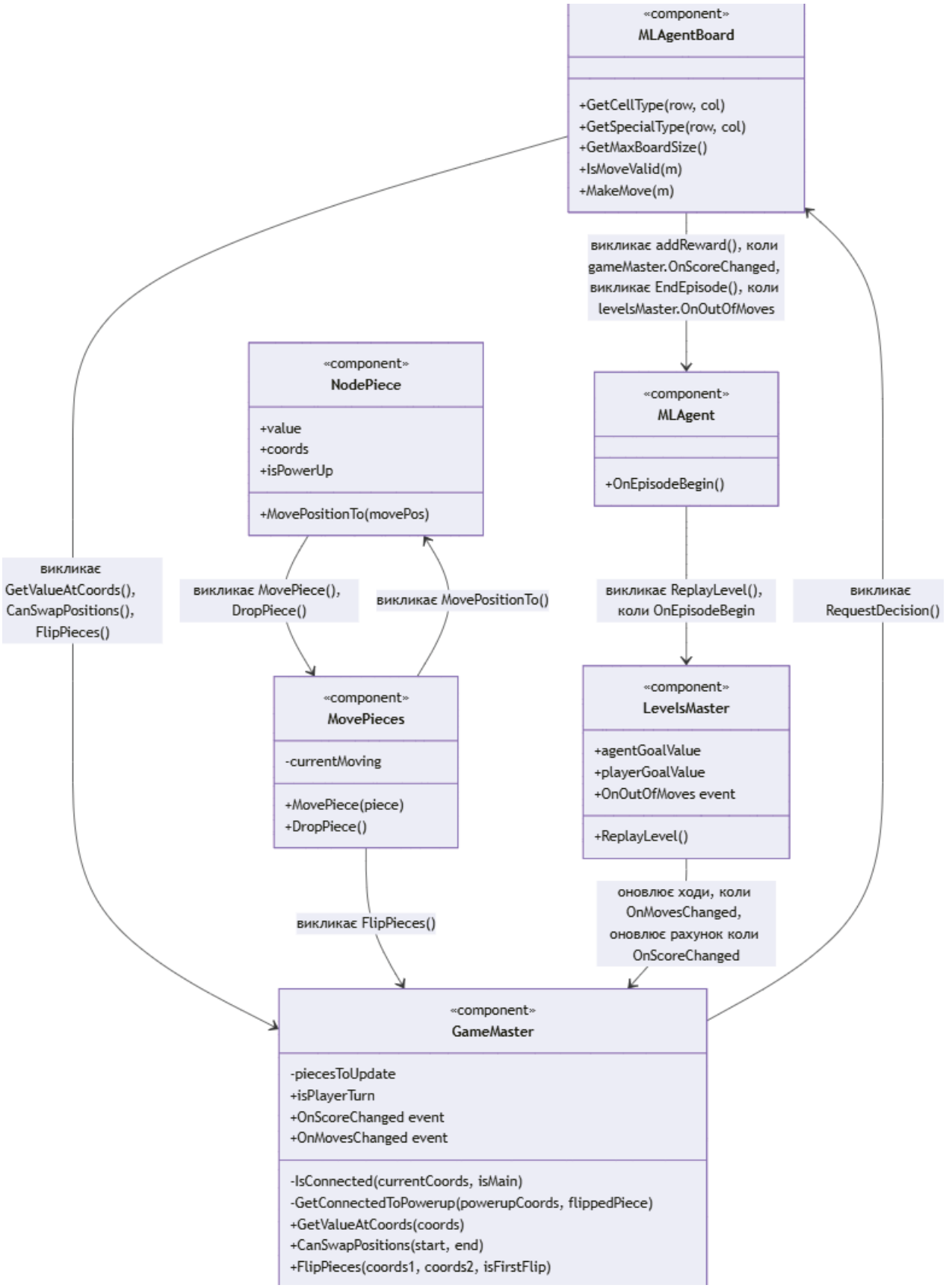


Рисунок 4.4 – Діаграма компонентів модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

Компонент NodePiece – це клас для різних фруктів чи спеціальних об’єктів, які розміщуються випадковим чином на ігровому полі, та з яких користувачу необхідно створювати комбінації з трьох в ряд, або більше. Поле value відповідає за тип фрукта (від 1 до 5). Якщо створено комбінацію із 4 чи більше фруктів, то створюється спеціальний об’єкт, а поле isPowerUp дорівнюватиме true, поле value в такому випадку матиме значення від 6 до 9. Поле coords – це структура, яка відповідає за позицію фрукта на ігровому полі.

Компонент MovePieces необхідний для правильного руху фруктів, під час натискання, утримування та відпускання клавіші миші. Коли клавіша миші відпускається, а координати фрукти вже не ті, що були раніше, викликається функція класу GameMaster FlipPieces, яка міняє фрукти місцями. Це відбувається, навіть якщо комбінації не станеться, в такому випадку фрукти повторно поміняються місцями.

Клас GameMaster, після отримання даних про два фрукти, які змінилися місцями, перевіряє чи утворено в результаті цього якусь комбінацію функцією IsConnected, або GetConnectedToPowerup. Якщо було створено якусь комбінацію, то запускаються event OnScoreChanged та OnMovesChanged, які відповідно оновлюють рахунок та кількість ходів користувача (в залежності від змінної isPlayerTurn, яка вказує на те чий зараз хід). Фрукти, та інші об’єкти, що рухаються в результаті створення комбінацій, додаються у список piecesToUpdate, а при зупинці видаляються з нього. Таким чином, ефективно викликається функція RequestDecision, тільки тоді, коли piecesToUpdate порожній, щоб агент не робив свої ходи занадто швидко.

Після надсилання запиту на прийняття рішення RequestDecision, агент одразу шукає всі можливі ходи на поточному ігровому полі, за допомогою функції IsMoveValid, яка в свою чергу викликає функцію CanSwapPositions із класу GameMaster, бо вся інформація про ігрове поле знаходиться там. Після цього, агент виконує функцію MakeMove, щоб виконати обраний хід на ігровому полі. MakeMove викликає універсальну функцію FlipPieces із класу GameMaster, тож хід відбувається так само, як у випадку, коли його робить гравець. Тому, у

класі GameMaster знову спрацьовує event OnScoreChanged, на який підписаний MLAgentBoard, та під час цього агенту додається нагорода за виконаний хід AddReward. Коли ж спрацьовує event OnOutOfMoves із класу LevelsMaster, агент закінчує епізод навчання функцією EndEpisode, та автоматично викликається функція OnEpisodeBegin у класі MLAgent. Тоді, вона викликає функцію ReplayLevel класу LevelsMaster, що дозволяє легко оновити значення на кшталт рахунку і кількості ходів, та продовжити тренування агента.

Діаграма класів, створена у Visual Studio, демонструє їх основний зміст (рисунок 4.5).

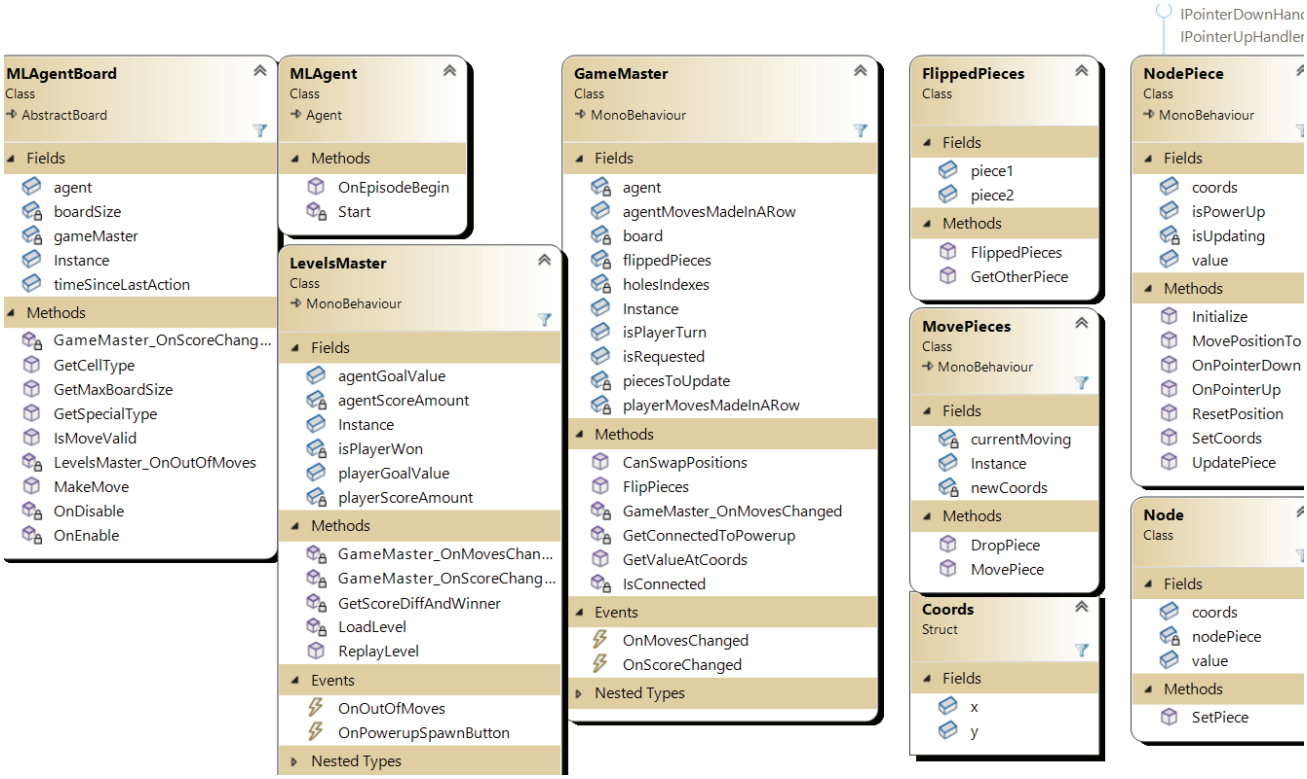


Рисунок 4.5 – Діаграма класів модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

Coords – це структура, яка вказує на координати ігрового поля, яка також має різні функції для полегшення перевірок координат. Кожна клітина ігрового поля має свої координати Coords.

Node – це клас, що вказує на клітину ігрового поля. Він необхідний через те, що клітина може містити не тільки якийсь фрукт, а ще різний задній план. Наприклад, це може бути діра, золотий чи залізний фон для складніших рівнів, та інше.

NodePiece – це клас, що вказує на фрукт, чи інший спеціальний об’єкт ігрового поля, який гравець чи NPC можуть рухати.

MovePieces – це клас, що відповідає за рух гравцем певного об’єкту ігрового поля (змінна currentMoving). Гравець одночасно може рухати максимум один об’єкт, а коли відпускає клавішу миші, залежачи від її положення, об’єкт міняється місцями з іншим, у відповідному напрямку, тобто викликається функція FlipPieces.

FlippedPieces – це клас, що містить в собі інформацію щодо двох об’єктів ігрового поля, які щойно помінялися місцями, та використовується як під час ходів гравця, так і під час ходів NPC.

GameMaster – це основний клас гри, який стосується всього ігрового поля. Він створює ігрове поле, заповнює його фруктами випадковим чином, але так, щоб не було ніяких комбінацій із самого початку. Коли викликається функція FlipPieces, в цьому класі перевіряється чи було створено якісь комбінації, створюються спеціальні об’єкти після великих комбінацій. Створюються нові фрукти після виконання ходу. В необхідний момент викликаються функції зміни рахунку, запиту на прийняття рішення від агента, зміни кількості ходів, та інше.

LevelsMaster – це клас, що використовується для зміни інформації інтерфейсу користувача (user interface, далі – UI).

MLAgentBoard – це клас, що наслідує AbstractBoard із бібліотеки MLAgents. Він використовується для виконання майже всіх дій агента, тобто для збору інформації щодо ігрового поля, знаходження можливих ходів, виконання ходу, та іншого.

MLAgent – це клас, що наслідує Agent із бібліотеки MLAgents. Він використовується для контролю життєвого циклу агента та призначення винагороди.

На діаграмі послідовності (рисунок 4.6) зображено, як різні об’єкти послідовно обмінюються повідомленнями в часі.

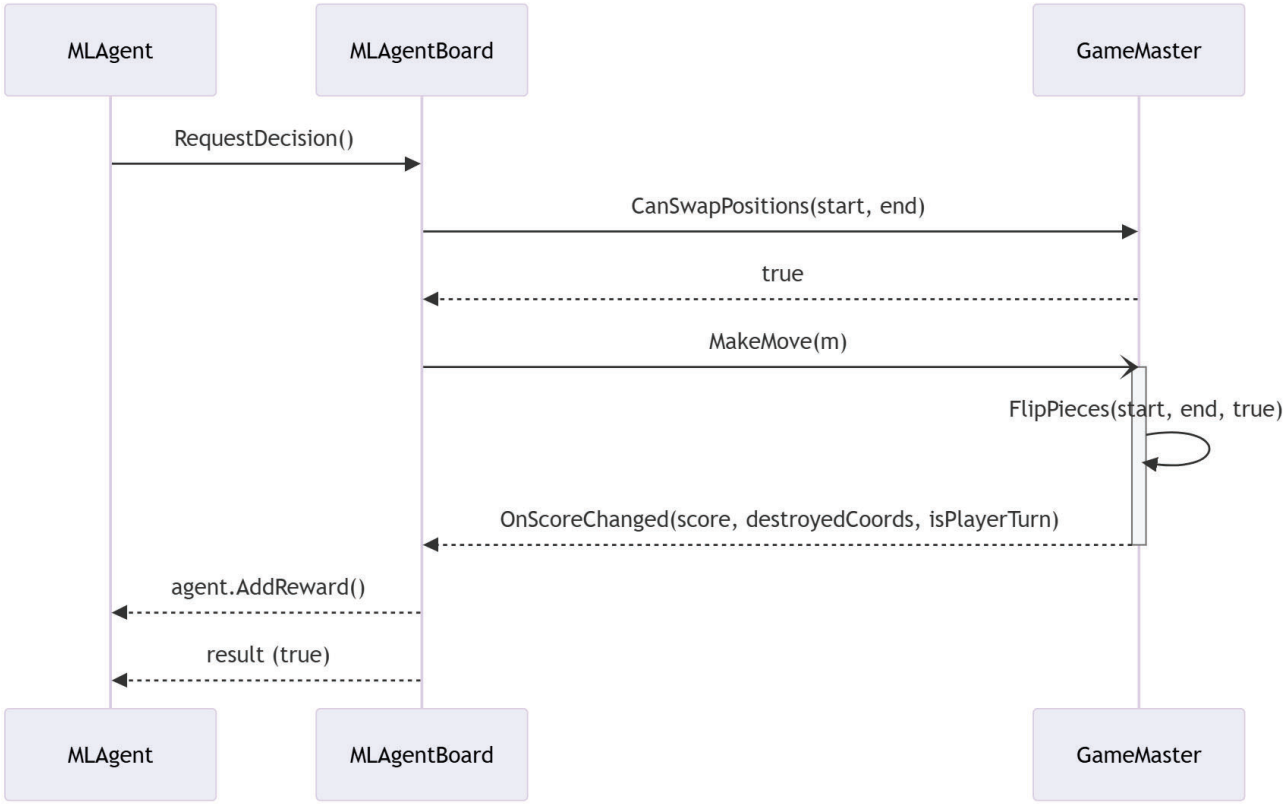


Рисунок 4.6 – Діаграма послідовності модуля «Оптимізація поведінки неігрового персонажа» інформаційної ігрової системи

На діаграмі послідовності видно, що без затримок виконується функція запиту на прийняття рішення від агента. Далі, так само швидко, перевіряються різні варіанти ходів, та якщо функція CanSwapPositions(start, end) повертає true, то цей хід додається до списку можливих, та для якогось із них агент виконається функція MakeMove(m).

Найбільше ж часу витрачається після виклику функції FlipPieces(start, end, true), бо вона запускає фізичний рух фруктів на ігровому полі, та вони це рухаються не миттєво, а плавно. Гравець та NPC не можуть виконувати ходи, поки якісь фрукти рухаються на ігровому полі. Також, після виконання ходу,

може статись таке, що додаткові фрукти також сформують між собою комбінацію з трьох або більше, тоді пройде ще більше часу, перш ніж програмний код перейде до виконання наступних етапів.

Після цього, зміна рахунку та додавання винагороди агенту відбуваються без затримок, а цикл повторюється, і знову виконується функція запиту на прийняття рішення від агента.

5 РОЗРОБКА Й ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ІНФОРМАЦІЙНОЇ ЗАБЕЗПЕЧУЮЧОЇ СИСТЕМИ

Основним елементом бази даних інформаційної ігрової системи «Три в ряд» є ігрове поле, тобто матриця об’єктів зі створеного класу Node. На початку гри вона заповнюється випадковим чином, пропускаючи дірки в полі. Також, перевіряється, щоб не було ніяких комбінацій з трьох чи більше фруктів у ряд із самого початку гри. Результат створення ігрового поля показано на рисунку 5.1.

Рисунок 5.1 – Створене ігрове поле у вікні Unity, із зазначеними індексами

Всі основні файли, що використовуються в інформаційній ігровій системі, створеної за допомогою ігрового рушія Unity, зберігаються у папці «Assets», як зображено на рисунку 5.2.

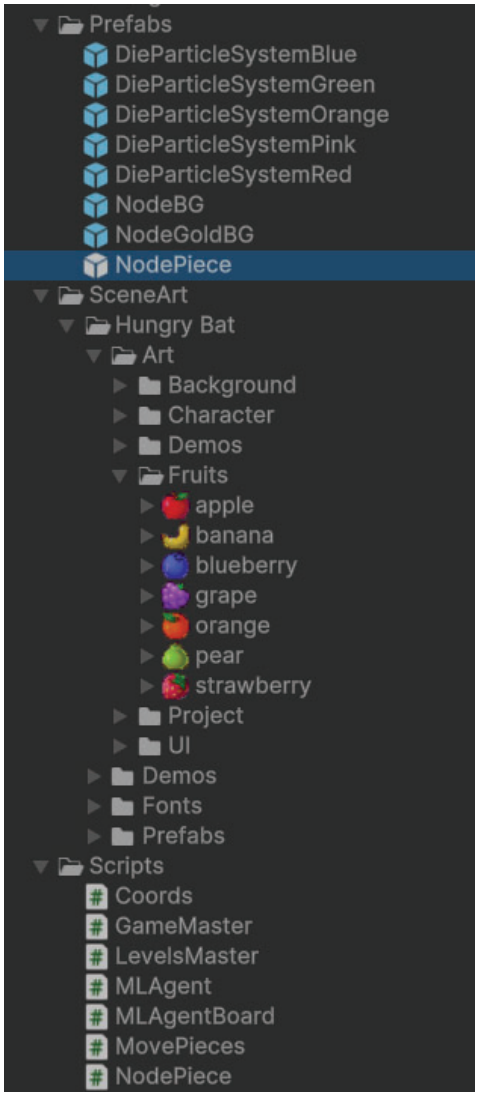


Рисунок 5.2 – Фрагмент основних файлів інформаційної ігрової системи з папки «Assets» у вікні Unity

У папці «Prefabs» зберігаються шаблони об’єктів гри, за допомогою яких, полегшується створення необхідних об’єктів на ігровій сцені з програмного коду. Наприклад, «NodePiece» використовується для створення всіх фруктів на ігровому полі. У папці «SceneArt» зберігаються всі зображення для гри, а у папці «Scripts» – всі скрипти для гри, написані мовою програмування C#.

6 РОЗРОБКА Й ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ПРОГРАМНОЇ ЗАБЕЗПЕЧУЮЧОЇ СИСТЕМИ

Для використання ML-Agents у Unity спочатку треба встановити необхідну версію Python, та за допомогою командного рядку, створити і запустити віртуальне середовище у папці проєкту Unity. Далі, слід встановити необхідну версію бібліотеки PyTorch [4]. Потім, встановлюється бібліотека ML-Agents та додається до проєкту в Unity через Unity Package Manager. Після цього, всі необхідні компоненти будуть доступні безпосередньо в Unity, що робить налаштування і тренування агентів досить зручним. Але, тренування агентів все одно запускається за допомогою командного рядку, бо воно відбувається через віртуальне середовище Python.

Лістинг 6.1 – завантаження бібліотеки ML-Agents за допомогою командного рядку у Windows 11

```
cd D:\Unity\Projects\MatchThree
py -m venv venv
venv\Scripts\activate
python -m pip install --upgrade pip
pip3 install torch~=2.2.1 --index-url
https://download.pytorch.org/whl/cu121

python -m pip install mlagents==1.1.0
```

Із встановленою бібліотекою ML-Agents в Unity, створено новий ігровий об’єкт із власним скріптом MLAgentBoard, який наслідує клас AbstractBoard із бібліотеки, та MLAgent, який наслідує клас Agent із бібліотеки. Також, підключені готові скріпти-компоненти з бібліотеки BehaviourParameters, Match3Sensor та Match3Actuator. BehaviourParameters відповідає за загальні налаштування агента, Match3Sensor збиратиме спостереження з ігрового поля, а Match3Actuator створюватиме перелік усіх можливих ходів (Move) у грі.

Основний скрипт для тренування агента – це MLAgentBoard (Додаток А, лістинг А.1). Функції GetMaxBoardSize(), GetCellType() та GetSpecialType(), автоматично використовує Match3Sensor для отримання спостережень. У GetMaxBoardSize() заповнюються поля Columns, Rows, NumCellTypes, NumSpecialTypes. Перші два – це ширина і висота ігрового поля. NumCellTypes – це кількість можливих різних типів звичайних об’єктів. В даному проєкті існує 5 різних фруктів, але також ігрове поле не звичайної форми, а з дірками, які можуть бути в різних місцях залежно від рівня (їх value завжди -1). Тобто, ці дірки також є повноцінним типом можливих клітинок, тому NumCellTypes = 6. NumSpecialTypes – це кількість спеціальних об’єктів, які не завжди присутні на ігровому полі, але іноді можуть з’являтися. В даному проєкті існує 4 спеціальних об’єкти за комбінації від 4 до 7 фруктів одночасно, тому NumSpecialTypes = 4. GetCellType(int row, int col) автоматично викликається, щоб агент розумів який звичайний об’єкт знаходиться в позиції row, col, повертаючи значення від 0 до NumCellTypes-1. GetSpecialType(int row, col) автоматично викликається, щоб агент розумів який спеціальний об’єкт знаходиться в позиції row, col, повертаючи значення від 0 до NumSpecialTypes-1.

Функції IsMoveValid(Move m) та MakeMove(Move m) автоматично виконуються через Match3Actuator, коли викликається RequestDecision(). IsMoveValid знаходить всі можливі ходи на ігровому полі, а MakeMove безпосередньо виконує хід.

Функція agent.AddReward(float increment) використовується для встановлення винагороди агенту під час навчання з підкріпленням. В даному проєкті нагорода видається за створення комбінацій з 4 або більше фруктів, за використання спеціальних об’єктів, за збиття фруктів, які є ціллю агента, а також невелика нагорода за збиття фруктів, які є ціллю гравця. У всіх інших випадках встановлюється невелика негативна винагорода. Поле agent заповнюється у скрипті MLAgent, який використовується для виклику функції OnEpisodeBegin (Додаток А, лістинг А.2).

Використання функції OnEpisodeBegin() дозволяє зручно обнулити необхідні змінні і продовжити навчання агента, адже вона викликається одразу після EndEpisode(), наприклад, коли у агента закінчилися ходи.

Скрипт GameMaster містить основні функції для застосування навчання з підкріпленням (Додаток А, лістинг А.3). Функція CanSwapPositions(start, end) використовується, коли агент у своєму скрипті MLAgentBoard перевіряє, чи може він зробити хід між координатами start та end. Тобто, перевіряється, чи буде після певного ходу створена якась комбінація з трьох в ряд чи більше. Таким чином, агент знаходить всі можливі ходи, а пізніше обирає один з них. Обраний хід виконується шляхом виклику функції FlipPieces(coords1, coords2, isFirstFlip), в результаті два об’єкти додаються до списку piecesToUpdate, а пізніше у функції Update() відбувається повна обробка зробленого ходу. Коли ж ця обробка закінчується, а список piecesToUpdate знову став порожній, викликається функція agent.RequestDecision(), щоб надіслати агенту запит на прийняття рішення щодо наступного ходу (якщо зараз не хід гравця).

Після створення описаних скриптів, було збережено файл гри .exe у режимі ServerBuild, щоб уникнути рендерингу графічних елементів. Потім, розпочато процес тренування агента методом навчання з підкріпленням, із використанням 4 сутностей гри одночасно, для прискорення процесу навчання командою «mlagents-learn config/Match3Agent.yaml –env=Build/MatchThree –num-envs=4 –run-id=BuildTest2» (рисунок 6.1).

Рисунок 6.1 – Запуск навчання з підкріпленням

Команда «config/Match3Agent.yaml» використовується для передачі файлу з гіперпараметрами (налаштуваннями) до процесу навчання з підкріпленням. Список гіперпараметрів наведено у лістингу 6.5.

Лістинг 6.5 – файл Match3Agent.yaml із переліком гіперпараметрів до навчання

```
default_settings:
  trainer_type: ppo
  hyperparameters:
    batch_size: 1024
    buffer_size: 10000
    learning_rate: 0.0003
    beta: 0.002
    epsilon: 0.2
    lambd: 0.99
    num_epoch: 5
    learning_rate_schedule: constant
  network_settings:
    normalize: true
    hidden_units: 256
```

```
    num_layers: 4
    vis_encode_type: match3
  reward_signals:
    extrinsic:
      gamma: 0.99
      strength: 1.0
  keep_checkpoints: 5
  max_steps: 5000000
  time_horizon: 128
  summary_freq: 2000

behaviors:
  Match3Agent:
    max_steps: 10000000
```

При проведенні навчання агента одночасно на 4 сутностях гри, CPU був задіяний в середньому на 60-70%, а використання оперативної пам’яті становило від 100 до 500 МБ на кожну активну сутність гри.

Початкові значення винагород агента становили близько 1000. Після перших 15 хвилин тренування середня значення нагороди, яку отримував агент становило 1100. Після 40 хвилин тренування, середня нагорода становила 1200. Тренування тривало майже 2 години (1,3 млн кроків), а фінальним значенням винагороди було 1267 (рисунок 6.2).

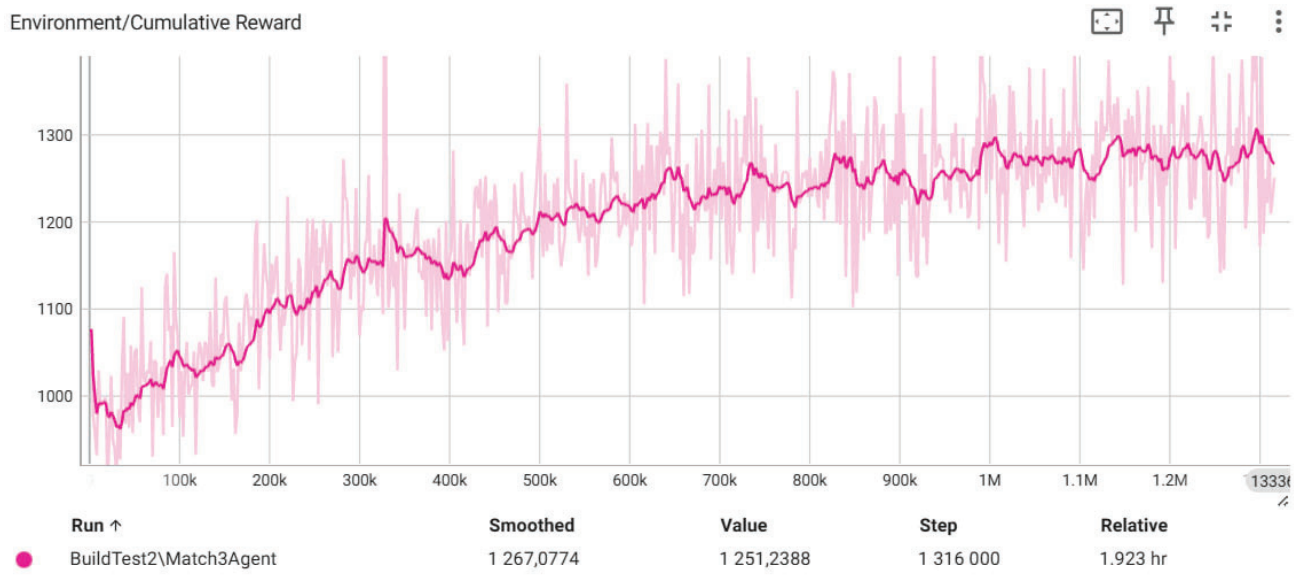


Рисунок 6.2 – Графік із усередненими значеннями нагород під час тренування

Далі, у налаштуваннях NPC в Unity, можна використовувати сформований файл моделі агента, отриманий після тренування. В такому разі, параметр Behavior Type змінюється з «Default» на «Inference Only», щоб агент більше не намагався вчитись, а просто використовував сформовану модель (рисунок 6.3).

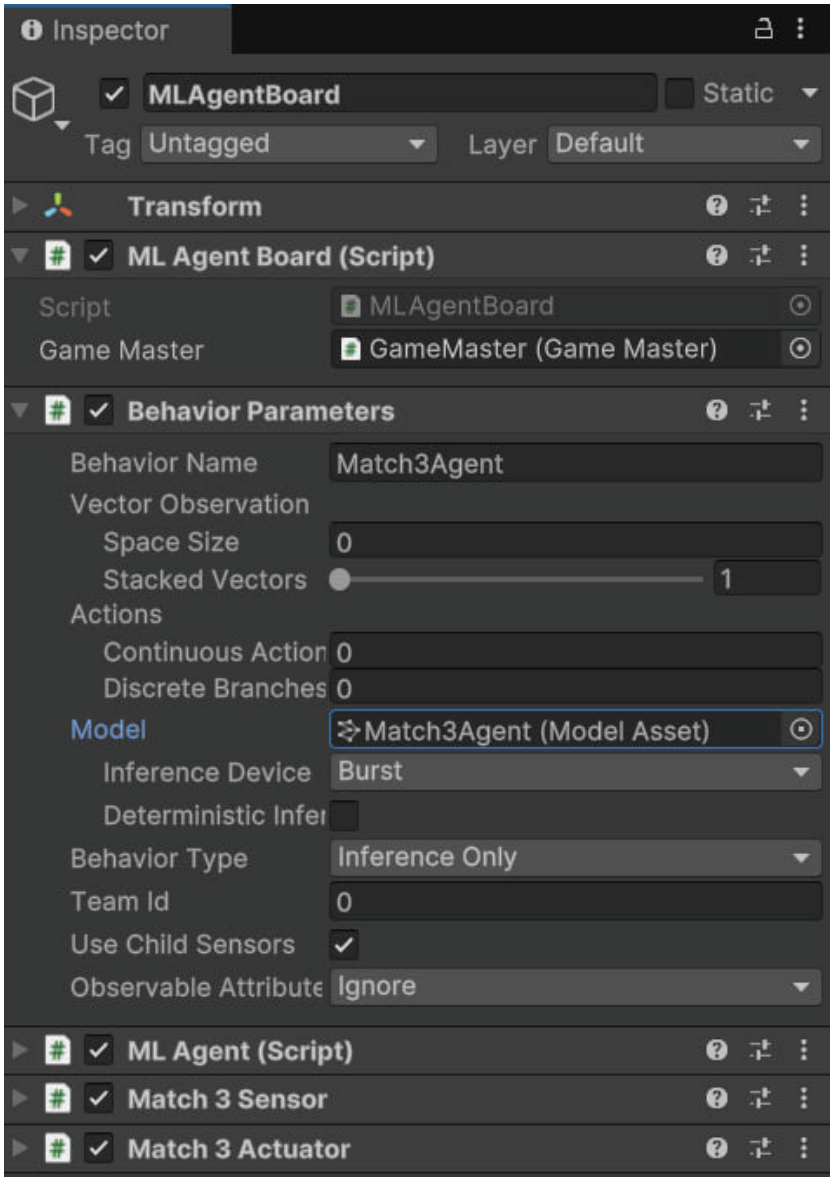


Рисунок 6.3 – Налаштування NPC в Unity із сформованим файлом моделі

7 РОЗРОБКА USER EXPERIENCE (UX) ТА USER INTERFACE (UI) РІШЕНЬ

У даному проєкті інформаційної ігрової системи, інтерфейс користувача (UI) створений таким чином, що на екрані завжди відображається інформація про кількість ходів (MOVES), що залишились у NPC (зверху) та у гравця (знизу). Також, у відповідних місцях розташовано кількість очок NPC і гравця, їхню ціль на цей рівень, та кількість, яку залишилось зібрати (рисунок 7.1).

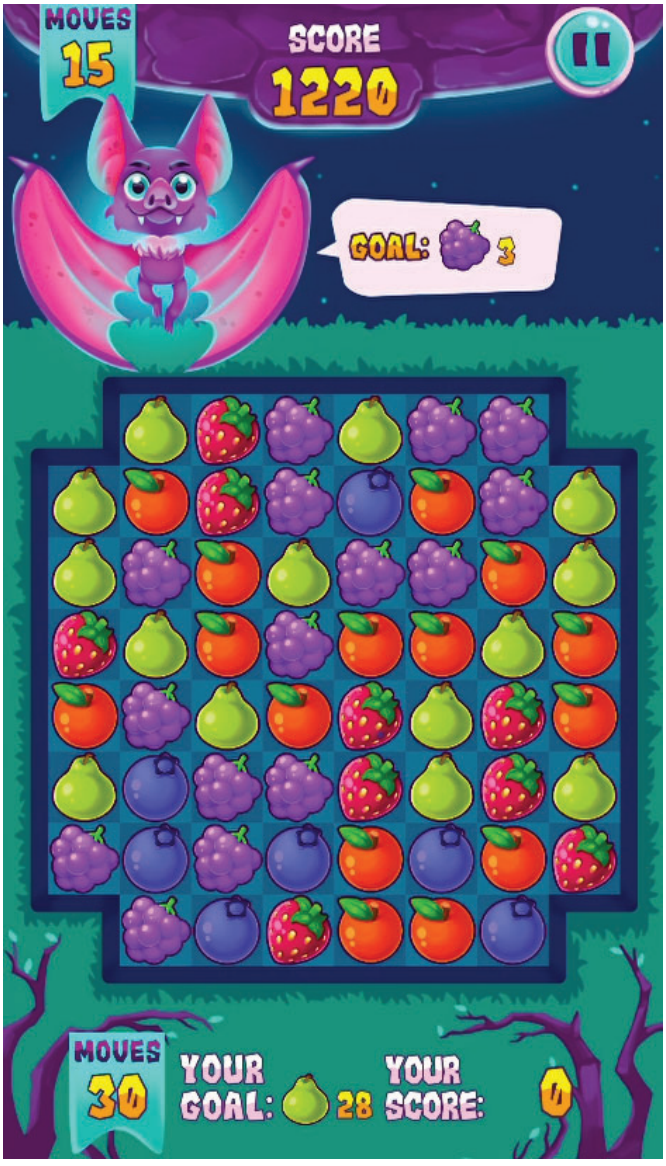


Рисунок 7.1 – Скріншот гри із зображенням елементів інтерфейсу користувача

Ігрові об’єкти рухаються плавно, та створюють ефекти відповідного кольору під час зникнення з ігрового поля, а нові об’єкти не просто з’являються на пустих місцях, а імітують випадання зверху, що створює приємний досвід для користувача.

Оновлення даних UI відбувається у скрипті LevelsMaster (Додаток А, лістинг А.4). Це відбувається тільки в той момент, коли було створено якусь комбінацію, шляхом виклику event OnScoreChanged. Також, коли у гравця і NPC закінчуються ходи, визначається переможець функцією GetScoreDiffAndWinner(). У гравця і агента є ціль для кожного рівня – набрати достатню кількість певного фрукту (як зазначено у полі «GOAL», див. рисунок 7.1). Перемагає той, хто зміг зібрати необхідну кількість фруктів. Якщо ж обидва учасники гри виконали свою ціль, перемагає той, хто набрав більше очок.

8 ТЕСТУВАННЯ ТА ОЦІНКА НАДІЙНОСТІ ФУНКЦІОНУВАННЯ НЕІГРОВОГО ПЕРСОНАЖА

Для того, щоб використовувати NPC, навченого за допомогою RL, у проєкті гри, він має відповідати всім поставленим вимогам. Для реалізації гри по черзі між гравцем і NPC, була створена додаткова проста функція (лістинг 8.1).

Лістинг 8.1 – функція слідкування за поточним ходом у скрипті GameMaster

```
private void GameMaster_OnMovesChanged(object sender,
OnMovesChangedEventArgs e)
{
    // player makes 15 moves, then agent makes 15 moves
    if (isPlayerTurn)
    {
        playerMovesMadeInARow++;
    }

    if (playerMovesMadeInARow == 15)
    {
        playerMovesMadeInARow = 0;
        isPlayerTurn = false;
    }
    else if (agentMovesMadeInARow == 15)
    {
        agentMovesMadeInARow = 0;
        isPlayerTurn = true;
    }
}
```

У ході тестування режиму гри проти створеного NPC, було виявлено, що він дійсно дуже часто виконує оптимальні ходи, які призводять до створення великих комбінацій, та виконання поставленої мети на рівень. На рисунку 8.1 зображено приклад комбінації ходів NPC, де він спочатку об’єднує 5 фруктів у ряд, а потім одразу використовує створений спеціальний об’єкт для збиття

великої кількості фруктів, та навіть направив його у потрібний напрям (угору), щоб зібрати найбільше необхідних фруктів.

а) б)

Рисунок 8.1 – Приклад оптимальної комбінації ходів NPC: а) перший хід; б) другий хід

Також, було проведене тестування гри проти NPC із визначенням переможця за 30 ходів. Перемагає той, хто зібрав необхідну кількість фруктів потрібного типу. Якщо і гравець, і агент все зібрали, тоді перемагає той, у кого більше очок. NPC переміг чотири рази з восьми (рисунок 8.2).

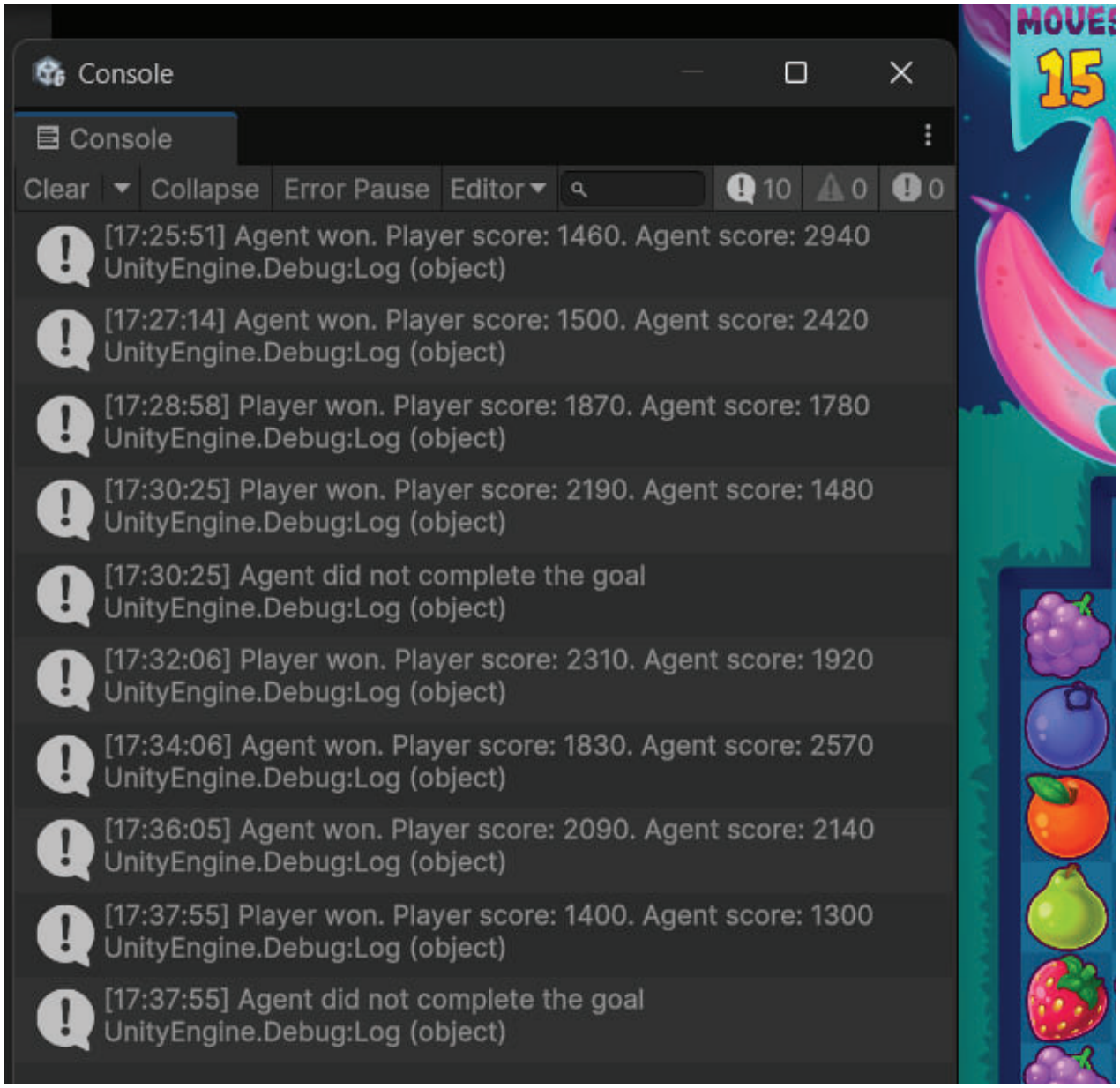


Рисунок 8.2 – Тестування гри проти NPC

ВИСНОВКИ

Неігровий персонаж, натренований методом навчання з підкріпленням для даної гри жанру «Три в ряд», часто використовує оптимальні ходи, але він це робить не завжди, а приблизно у 60% ходів, тому він не є ідеальним. Скоріше за все, результат покращився б при використанні потужнішого комп'ютера для навчання агента, бо якість CPU/GPU впливає на швидкість проходження кроків навчання. Наприклад, це дозволило б пройти не 1,3 млн кроків, а 10 млн за 2 години, що є суттєвою різницею. Також, даного NPC можна покращити поєднанням декількох методів машинного навчання, наприклад додавши imitation learning, або curriculum learning.

З іншого боку, для деяких розробників інформаційних ігрових систем, навіть NPC, доведений до, хоча б, 80% оптимальних рішень, вже може бути гарною альтернативою NPC із стандартними методами поведінки. Серед багатьох гравців у відеоігри спостерігається невдоволення, коли NPC стає передбачуваним і відчувається слабким, а також ніхто не хоче змагатися із суперником, який не робить помилок, і якого неможливо перемогти.

Тому, на мою думку, використання методів машинного навчання для оптимізації поведінки NPC інформаційних ігрових систем може бути дуже вдалим рішенням для майбутніх проєктів відеоігор, із новими поглядами на взаємодію між гравцем і NPC.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Чинний від 2017-07-01. – Київ: ДП «УкрНДНЦ», 2016.

2. Wolf D. – Deep Reinforcement Learning: Shaping the Future of Game Design. URL:

https://www.linkedin.com/pulse/deep-reinforcement-learning-shaping-future-game-design-torres-twxyc

3. IBM What is reinforcement learning? URL: https://www.ibm.com/think/topics/reinforcement-learning

4. Документація Unity ML-Agents Toolkit. URL: https://github.com/Unity-Technologies/ml-agents

5. Zeerek A. State Machines vs Behaviour Trees. URL: https://www.polymathrobotics.com/blog/state-machines-vs-behavior-trees

6. GeeksForGeeks Hierarchical Reinforcement learning (HRL) in AI. URL: https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

7. The AlphaStar team Grandmaster level in StarCraft II using multi-agent reinforcement learning. URL: https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/

8. Методичні вказівки до організації виконання та захисту кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти для студентів спеціальності 122 «Комп'ютерні науки» за освітньою програмою «Інформаційні технології управління». [Електронний ресурс] / Упоряд.: К. Е. Петров, А. В. Міхнова, М. С. Кудрявцева, М. В. Євланов, Т. І. Борисенко. – Електронне видання. – Харків: ХНУРЕ, 2023. – 68 с.