Godot RML Agents

Creating RL Agents to solve games on the Godot engine

Sponsor: Ron Wright

Godot Agents

Luke Flock and Cole Clark

12/8/2024

Table of Contents

Introduction

I. Project Introduction

The recent advancements in machine learning (ML) and artificial intelligence (AI) have made it an essential skill to learn to stay competitive in the future workforce. The goal is to introduce foundational knowledge of AI and ML into a K-12 curriculum as it needs to teach students the current and most modern technologies.

This project seeks to address that gap by introducing a new competitive event in collaboration with SciOly and SkillsUSA, where students are tasked with designing and training a Reinforcement Machine Learning (RML) agent to complete a video game challenge. This will be made to provide students with an introduction to RL techniques by allowing them to build their agents and apply theoretical concepts in a fun, practical environment.

Wahkiakum School District is pioneering this initiative as part of a K-12 curriculum focused on equipping students with RL skills. This project required us to create multiple RML agents that can serve as benchmarks for student teams. Our client, Ron Wright, will use these agents and our experience creating them to adjust rules for the student projects. Ron's goal is to turn this project into a Science Olympiad (SciOly) competition. By participating, students will not only gain valuable technical skills but also become more familiar with the broader field of AI.

II. Background and Related Work

This project applies RL techniques to video games, a field that has received significant attention due to the success of RL algorithms in various games, such as AlphaGo [3] and OpenAI's agents for Dota 2 [4]. Games provide a great environment for RL because they offer complex and dynamic challenges, making them suitable for testing agents' adaptability and problem-solving skills.

Our work focuses on translating current RL techniques into an educational context, making them accessible to K-12 students who are just beginning to explore the field. The agents will be designed to solve simple game-specific challenges, collect rewards, and avoid obstacles using RL principles. Additionally, we will create a game for next year's competition.

This project does not aim to innovate new RL algorithms but rather focuses on creating a simplified, practical framework where students can learn core RL concepts. Our primary contribution will be the development of multiple RL agents that serve as purely educational tools.

Overall, here is a brief overview of the technical skills required for this project:

GD Script for Godot

- The team will need to become proficient in Godot's programming language, GD Script, which closely resembles Python [5]. Given the team's experience with Python, GD Script should be relatively easy to learn and apply.

Proximal Policy Optimization Reinforcement Learning

- The team must understand how the proximal policy optimization algorithm can be applied to RL, particularly in the context of game environments where episodes can be fully simulated, and rewards collected over time.

Godot Game Engine

- The team will need a strong understanding of how the Godot game engine works, including handling input, managing scenes, and creating game objects. This knowledge is crucial for embedding RL agents within the game.

Integration of RL Libraries in Godot

- Familiarity with integrating RL libraries or custom RL solutions into the Godot engine is necessary.

Documentation and Curriculum Design

- Since this project involves creating educational resources, the team will need to develop clear, step-by-step documentation. This will help students understand how to implement RL solutions in their own games, making the learning process accessible and engaging.

III. Project Overview

In the last few years, it has become increasingly popular to develop reinforcement machine learning algorithms for playing games. Many videos on YouTube show algorithms that have learned to play Pacman, Flappy Bird, Pokemon, Chess, Pool, and several other games [1]. With the increasing interest in RML, our project is focused on helping our client teach the concepts of reinforcement machine learning to middle and high school students.

This project's goal is to assist Ron Wright with a student competition that will require the students to develop reinforcement machine learning models. Students will be in groups of three and work to develop a game in Godot. After they have completed the game implementation they will work to create a RML agent that can play the game. Students will also be required to submit documentation for their work. This will be graded based on the game, RML agent, and a multiple-choice test on RML.

We are assisting Ron with finalizing the rules for the competition along with providing RML models to solve a simple game in Godot. In the game, the player is a tank moving on a surface collecting daisies. The goal is to collect daisies for 30 seconds and then return to the starting point within 5 seconds after the initial 30. Only one daisy will be on the map at a time with its location given to the player as coordinates on the map. There are no enemies or walls in the game, however the player must avoid falling off the edge of the map. Once a player falls off the edge they are unable to get back and collect daisies.

Our solution to the daisy collecting game will utilize the Deep Q-Network (DQN) algorithm, providing an example of how reinforcement learning can be applied to a simple game environment. The DQN agent is designed to navigate the game map, collect daisies, and return to the starting point within the specified time frame. Using a neural network, the agent learns to determine optimal actions based on input data, such as the tank's position, velocity, and the coordinates of the daisy. A reward-based system guides the learning process, encouraging the

agent to maximize performance through trial and error while avoiding penalties, such as falling off the map. This implementation will serve as a benchmark solution, offering students a practical demonstration of reinforcement learning in action. Comprehensive documentation will accompany the DQN agent, ensuring that students can follow the process and understand the underlying principles, making it an effective tool for learning reinforcement machine learning techniques.

In addition to the RML solution for the daisy collection game, we selected an environment from the OpenAI gymnasium Python library and changed it to a different theme. We selected the lunar lander environment and changed the colors to give the game an egg-drop theme. The player can change the way the wind is blowing to affect how the egg lands on the ground. Points are gained by moving closer to the landing pad, moving slowly, keeping the egg horizontal, and making the egg touch the ground. The player will also get a 100-point reward for a successful landing or a -100 penalty for crashing. The egg drop is considered successful if the player scores over 200 points.

To solve this environment we used the PPO (proximal policy optimization algorithm. This works by creating a neural network to choose the best action based on the egg's position and velocity. Then that action is assessed to determine how good it was and the agent is updated.

In addition to creating RML agents to solve the games described above we also created two games that can be used as examples for the competition. Both of these games are based on a simple Godot tutorial game [12]. The original game allows the player to move left, right, or jump in a 2D environment. The player's goal is to collect green coins which are at different locations in the environment. We have made two separate modifications to this game. The first modification involves moving platforms that the player must jump on. The player needs to jump on several platforms to reach the top of the environment and collect a single green coin. Since the player doesn't move with the platforms it can be difficult to stay on and reach the top. Our second modification to the game keeps the original platforms in place but includes enemies that chase the player around the environment while they try to collect coins. These enemies can move through platforms while chasing the player making them difficult to avoid. If one of the enemies touches the player before all green coins are collected the game is over.

Both games have been created using Godot. The Godot game engine is free to use and gives users full ownership of the games they develop using the engine ("Introduction"). We chose to use Godot since it is free to use and is the recommended game engine for students in the competition.

IV. Client and Stakeholder Identification and Preferences

Our primary client is Ron Wright. Our goal with this project is to provide Ron with some RML algorithms to solve his snake game in Godot. This will prepare Ron for different solutions that he might see from students when they complete this task. Our RML models will also make student solutions easier to grade by providing complete solutions for comparison.

Since we are helping Ron decide on some of the rules for this student competition, the students who will be participating are also stakeholders. We need to make sure we are giving complete and high-quality solutions and rule recommendations to ensure that students are graded fairly.

Students will also be stakeholders for the games and video tutorials we create. For this task, we will need to create bug-free games that students can create RML models to solve. These games will need to have the right amount of complexity to make the competition interesting while also being simple enough for students to solve with RML. The games we create should be well-designed and documented for students to learn the basics of RML.

Team Members - Bios and Project Roles

Cole Clark

Computer science and accounting student interested in machine learning, software, and taxes. Prior projects include a Myers-Briggs personality type predictor using machine learning algorithms, A stock price predictor, and a spreadsheet application. Responsibilities for this project included learning reinforcement machine learning, modifying the openAI lunar lander environment to have an egg drop theme, and training an agent on the egg drop/lunar lander game.

Luke Flock

Computer science student interested in machine learning, game development, and data analysis. Past projects include a League of Legends game outcome classifier using scikit-learn, a Chrome extension for calculating similarity scores on Letterboxd, and a React movie tracker web app with personalized recommendations. Responsibilities for this project include designing and coding the DQN agent for the daisy collection game, debugging and optimizing the reinforcement learning model, integrating RL agents into the Godot game engine, and creating documentation to support the student competition.

Project Requirements

The Project Requirements section outlines the functional and non-functional specifications of the project, detailing each function's core capabilities, usage, and integration within the overall system. It includes comprehensive descriptions, use cases, user stories, and a traceability matrix to ensure all features and functionalities meet user and system requirements. This section serves as the cornerstone for the design and implementation of all project features.

I. Functional Requirements

Each Functional requirement is listed below with a description, source, and priority level.

II.1. Two Reinforcement Machine Learning Solutions

Functional Requirement	[FR-1] PPO (Proximal Policy Optimization)
Description	We will use the PPO algorithm to solve the egg drop game. The algorithm will train on several simulations to adjust the weights of its neural networks.

	After training, the algorithm should be able to play the game better than a human player.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

Functional Requirement	[FR-2] Deep Q-Network Agent
Description	For the daisy collection game, we will develop a Deep Q-Network (DQN) agent. This agent will utilize two neural networks, one for evaluating the current policy and another for updating target values. Through reinforcement learning, the DQN agent will learn a policy to determine the optimal actions for collecting daisies, avoiding the edge of the map, and efficiently collecting all the necessary coins.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

II.2. Two Games

Functional Requirement	[FR-3] Two different games
Description	Both games need to have a starting splash page and an ending splash. The ending page will show results after the game is completed by a player or agent. The games must be from different genres.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

II. Non-Functional Requirements

Non-functional Requirement	Description
[NFR-1] Bug-Free	The games and agents must be bug-free and work as expected in order to give students a proper example for their projects. Students should be able to make RML agents for our games without struggling to fix the bugs we left. Our RML agents should function without errors to give Ron (client) a benchmark for student projects.
[NFR-2] Complete Documentation	Our documentation for these games and agents should be complete and easy to understand. This is necessary to ensure that students have no trouble understanding and learning from our code. It is also required so that our client can easily understand and modify our code if needed for future use.
[NFR-3] Game Simplicity	The games we create should be simple so that students can understand them and create reinforcement machine learning agents to solve them. This means that creating a game with too much complexity or one that is hard to understand would not satisfy our requirements for this project.
[NFR-4] Interesting Games	While the game must be simple to use and solve they must also be interesting. The games we make should allow students to solve them with a wide range of different solutions. Making games that are interesting to solve will also increase student interest in their games and solutions.
[NFR-5] Game engine	The games we create must use the Godot game engine. This is required because the Godot game engine is free to use and will most likely be the game engine that students use to complete their projects.

III. Use Cases

Figure 1 shown below is a use-case diagram showing the functionality of our agents and modified games. Users will have the ability to play the game or run the agent on the game. Either of these options will lead to a result screen that shows the score for the episode. The documentation button will show an explanation for how to play the game and how we implemented our agents. The quit button allows the user to stop the program.

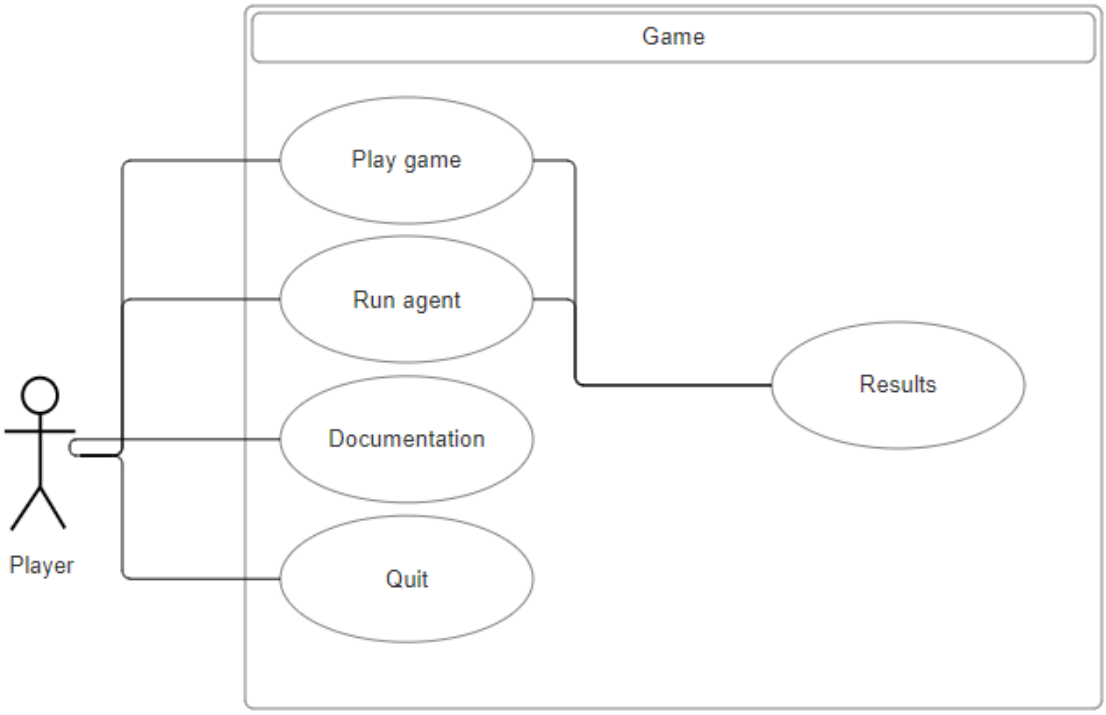


Figure 1: use-case diagram for games we modify and create during this project.

Use Case 1: Play Game

Pre-condition	- The user is on the main menu
Post-condition	- Results are shown for the player's activity in the game
Basic Path	1) The user selects Play Game from the main menu 2) The user completes the game 3) The user is shown a results screen indicating how well the player completed the game
Alt Path	-

Related Requirements	- A game environment - Game menu screen - Game results screen

Use Case 2: Train RL Agent

Pre-condition	- The user is on the train RL menu and selects the RL algorithm of choice
Post-condition	- The RL agent is trained using the algorithm chosen - Its learning and progress is saved to a file for future use
Basic Path	4) The user selects Train RL Agent 5) The user selects the RL algorithm 6) The agent is entered into the environment and begins learning how to interact and get points in the environment 7) When the user wants to stop training, they can click stop training 8) The training data is saved and is ready to be used
Alt Path	-
Related Requirements	- A game environment - Pre-defined RL algorithm, either Monte Carlo or - The game menu screen an

Use Case 3: Test RL agent

Pre-condition	- The user has trained the RL agent, and the user is on the Run RL Agent screen
Post-condition	- The agent is loaded and begins playing the game
Basic Path	1) The user clicks “Run RL Agent” 2) The user selects their algorithm of choice

	3) The RL agent navigates the level based on its trained policies 4) After 30 seconds the game ends and the agent's score is calculated
Alt Path	- At step 2, the user could have no training data (the game should indicate that the user needs to train the agent first)
Related Requirements	- RL agent training data - Game menu screen - Game mechanics

Use Case 4: View Documentation

Pre-condition	- The user is on the game menu screen, and they clicked the view documentation button
Post-condition	- Documentation and explanation for RL algorithm implementations are shown - Rules for the game are also shown
Basic Path	1) The user clicks the “View Documentation” button 2) Lets the user view the documentation of their RL agent, and describes how it works
Alt Path	-
Related Requirements	- Pre-written documentation - Pre-written game rules - Documentation integration into the game engine

Use Case 5: Quit

Pre-condition	- The user is on the main menu
Post-condition	- The game is closed

Basic Path	1) The user selects “Quit” from the main menu 2) The game is closed
Alt Path	-
Related Requirements	- A main menu

Use Case 6: Game Results

Pre-condition	- The user or agent has completed the game
Post-condition	- Game results are displayed
Basic Path	1) The user plays the game 2) The results of the game are displayed
Alt Path	1) The agent plays the game 2) The results of the game are displayed
Related Requirements	- A game environment - Game menu screen

IV. User Stories

User stories describe ways a user wants to interact with the system and the proper system response.

User Story US1: Play Game

As a user, I want to be able to play the game, so that I can see how the agent should play the game.

Feature: Play Game

Scenario: The user presses the “Play Game” button

Given the user is on the main menu

When they press the “Play Game” button

Then the game should run allowing them to play it.

User Story US2: Train RL Agent

As a user, I want to be able to train the RL agent so that it will perform better in the game.

Feature: Train RL Agent

Scenario: RL Agent Training

Given the user is on the main menu

When they press the “Train Agent” button

Then the agent should be trained to complete the game

User Story US3: Test RL Agent

As a user, I want to run the trained agent on the game, so that I can evaluate its performance.

Feature: Test RL Agent

Scenario: RL Agent Testing

Given the user is on the main menu and the agent is trained

When they press the “Test Agent” button

Then the trained agent should complete the game

User Story US4: View Agent Documentation

As a user, I want to view the documentation of a reinforcement learning agent, so that I can understand and learn how the algorithm is working

Feature: View Documentation

Scenario: View Documentation

Given the user is on the main menu

When they press View Agent Documentation

Then the documentation file should be displayed and viewable

User Story US5: Quit

As a User, I want to exit the program, so that I can use other programs.

Feature: Quit

Scenario: Quit

Given the user is on the main menu

When they press the “Quit” button

Then the program should stop running

User Story US6: View Game Results

As a user, I want to view the agent's performance and score after a game, so that I can see how well the agent played the game

Feature: View Agent results

Scenario: User Views Agent game result

Given the user has run the trained agent

When the game ends

Then the game score should be displayed

V. Traceability Matrix

Table 1 below maps functional requirements to their respective use cases and user stories. This ensures that all requirements are accounted for and linked to user scenarios.

Functional Requirement	Use Case	User Story	Priority
FR-1 PPO (Proximal Policy Optimization)	UC-2: Train RL Agent UC-3: Test RL Agent	US2: Train RL Agent US3: Test RL Agent	0
FR-2 DQN agent	UC-2: Train RL Agent UC-3: Test RL Agent	US2: Train RL Agent US3: Test RL Agent	0
FR-3 Two Different Games	UC-1 Play Game UC-2 Train RL Agent UC-3 Test RL Agent UC-4 View Documentation UC-5 Quit UC-6 Game Results	US1 Play Game US2 Train RL Agent US3 Test RL Agent US4 View Agent Documentation US5 Quit US6 View Game Results	0

Table 1: traceability matrix that maps requirements to use cases and user stories.

VI. System Evolution

The nature of our project is interested in showcasing the knowledge and understanding of Reinforcement Learning algorithms to K-12 grade students. User requirements might evolve as developers using the system may eventually require more advanced features, such as the inclusion of newer RL algorithms or support for multi-agent environments. To address these potential shifts, we plan to keep the system flexible and modular, allowing for easy refactoring or expansions.

Additionally, software dependencies, particularly third-party RL libraries, and Godot's version updates could impact our project. Currently, we are assuming stability in these libraries and tools, but changes in Godot's engine (such as updates from version 4.2) or the deprecation of certain libraries might necessitate reworking parts of our system. To mitigate this risk, we are prepared to regularly review and update our software stack to ensure compatibility and future-proof the project.

Moreover, we are tasked with creating our own Godot games and implementing custom RL agents within those games. As the scope of the games and user interactions are refined over time, the RL agents and their corresponding learning algorithms will need to be adapted accordingly. Game mechanics might evolve, necessitating the tweaking of learning policies, reward structures, and agent behavior to align with new educational goals or gameplay objectives. By anticipating these potential changes, we can proactively build a system that supports modifications and fine-tuning as the project evolves.

Software Design

The purpose of this document is to provide a detailed design overview of the reinforcement learning agents being developed using the Godot engine. This document is aimed at technical stakeholders, developers, and clients, particularly Ron Wright, who is overseeing the project. It outlines the system architecture, subsystems, and the integration of reinforcement learning algorithms (PPO and Deep Q-Network) into the game, along with the design decisions, algorithms used, and interface details.

The project's goal is to create two reinforcement machine learning agents that will provide students with examples of applying RML to games. The Deep Q Network (DQN) algorithm will be used on the daisy collection game in Godot while the Proximal Policy Optimization (PPO) algorithm will be applied to the egg drop game (modified openAI lunar lander environment).

I. System Overview

For this project, we will train two reinforcement machine learning agents in different environments in the Fall and create two games in the Spring. They will have the following components:

- **Game Interface**: Players or agents interact with the game via the provided interface. The game includes buttons for playing, training, and testing agents.
- **Agent Algorithms**: The agent component will contain the training and testing features for its corresponding algorithm.

- **Training and Testing Features**: The system will allow users to train agents, save training data, and load previously trained agents for testing.
- **User Interface for Agents**: A user interface will allow the user to choose the algorithm for training and control other aspects of the simulation.
- **Viewable documentation:** A way to view our agent's documentation and steps for implementation.

II. Architecture Design

II.1 Overview

The architecture of this project is set up using a layered game design approach. For the Fall portion of this project, our game environments were provided. The Godot game was created by our client and the egg drop game was made by changing the theme of openAI's lunar lander environment. Our agents were made to work with these existing environments. Figure 2 shows the components that will be created for our Godot games in the Spring.

- **Interface and Menu (UI)**: This handles all user input and output, managing buttons like "Train Agent," "Play Game," "Run Trained Agent," and "Quit Game."
- **Game Logic Layer**: Takes care of the main game mechanics and global variables. It updates scores, changes daisy point locations in the Godot game, updates positions and velocities in the egg drop game, and handles the global variables.
- **Reinforcement Learning Layer**: Where the RML algorithms live. This determines the best action to take based on the state of the environment.
- **Data Management Layer**: Deals with saving and loading agent training data.

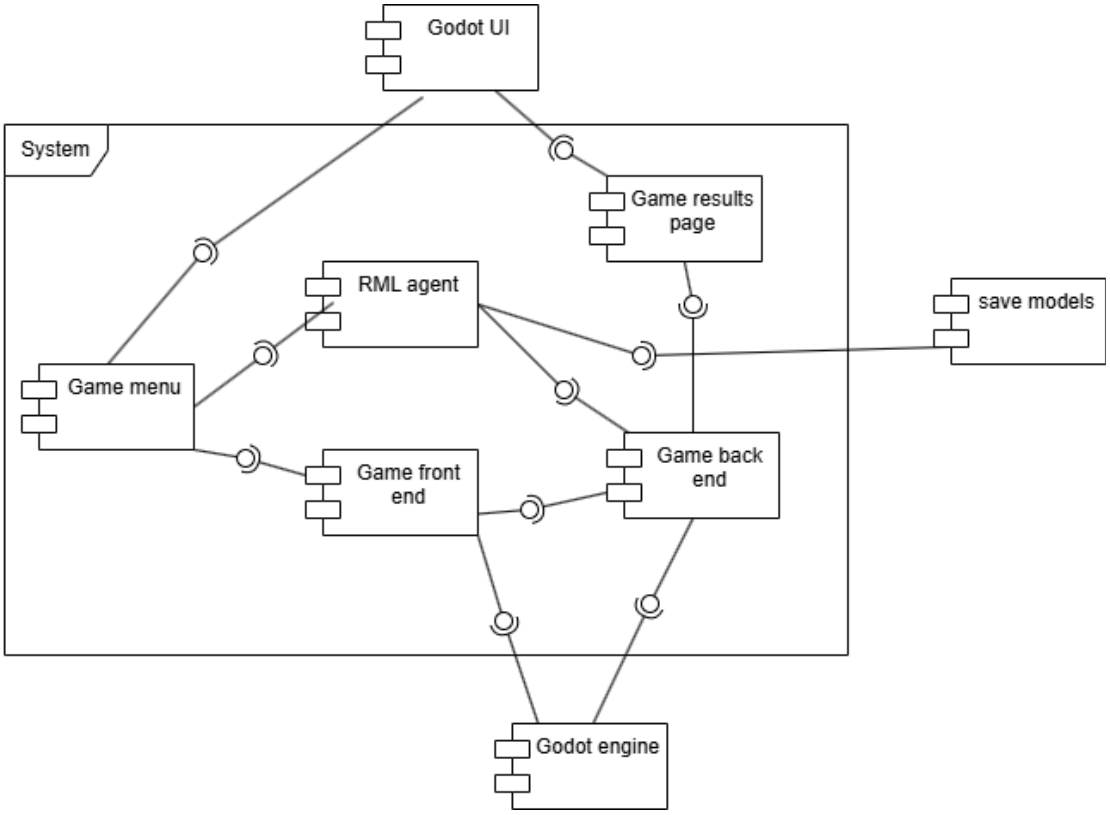


Figure 2: component diagram for our games in Godot.

II.2 Subsystem Decomposition

II.2.1. PPO (Proximal Policy Optimization) Agent

a) Description

The PPO agent will train on the game environment by running simulations and updating its neural networks. Once training is complete, the algorithm should be able to use its policy to play the game. This subsystem is responsible for the training and testing capabilities of the PPO algorithm on the game.

b) Concepts and Algorithms Generated

The PPO algorithm will be the primary algorithm used in this subsystem. Training will happen by running several simulations on the game. Actions in the environment will have a reward or penalty value for the result. These rewards or penalties will be used to update the neural networks of the agent. After training is done this subsystem will allow the user to test it on the game environment. Testing will determine the agent's state and then choose the best move with the highest expected reward according to its policy.

c) Interface Description

Services Provided

Service Name	Services Provided To	Description
Train PPO Agent	Game Menu	The user will be able to train the PPO agent from the game menu by using the PPO component
Run PPO Agent	Game Menu	The user will be able to run the trained PPO algorithm on the game from the game menu

Services Required	
Service Name	**Service Provided From**
get game information for training and testing	Game back end
save trained agent	Save models

II.2.2. DQN Agent

a) Description

The DQN agent learns using a deep neural network. During training, it takes in the current game state, runs it through a neural network to predict the best action, and updates its Q-values through experience replay and other mechanisms. This subsystem will be responsible for training and testing the DQN agent.

b) Concepts and Algorithms Generated

The DQN agent utilizes two neural networks to approximate Q-values for each state-action pair [1]. This means that as the agent interacts with the game, it stores its experiences (state, action, reward, next state) in a replay buffer. This buffer helps the agent learn by randomly sampling from past experiences, which breaks the sequence of events and makes learning more stable [2]. The agent uses the replay buffer to train a Q-network, which will learn the best action to take depending on the state of the agent and game. Once a good amount of episodes and iterations are performed, the Q-network weights and values are copied to the target network [3], which will be used to create a feedback loop of updates between the two networks in order to improve and find the best policy.

c) Interface Description

Services Provided

Service Name	Services Provided To	Description
Train DQN Agent	Game Menu	The user will be able to train the DQN agent from the game menu by using the Monte Carlo component
Run DQN Agent	Game Menu	The user will be able to run the trained DQN algorithm on the game from the game menu

Services Required	
Service Name	**Service Provided From**
get game information for training and testing	Game back end
save trained agent	Save models

II.2.3. Game Menu

a) Description

The game menu will allow the user to choose between training or testing the agent, playing the game, viewing documentation for the agent, or exiting the program.

b) Concepts and Algorithms Generated

The game menu will have buttons to train or test the agent, play the game, view documentation for the agent and game, or exit the program.

c) Interface Description

Services Required	
Service Name	**Service Provided From**
UI tools (buttons etc.)	Godot UI
PPO	train PPO agent
PPO	test PPO agent

DQN	train DQN agent
DQN	test DQN agent
Play Game	Game front end

II.2.4. Game Front End

a) Description

The game front-end component will show the user the game being played and relevant information that the agent will have access to. This may include player or agent health, distance in the game, or current score.

b) Concepts and Algorithms Generated

The game front end will allow the user to see their progress and current state in the game which will have some separation from the game back end layer that updates the values in the game.

c) Interface Description

Services Provided		
Service Name	**Services Provided To**	**Description**
Play game	Game Menu	The user will be able to play the game. relevant information to complete the game will be provided to the user and updated appropriately

Services Required

Service Name	Service Provided From
information from the game as it is running	Game back end
graphics	Godot engine

II.2.5. Game Back End

d) Description

The back end of the game will update the game information based on moves from the user or agent. relevant information will be communicated to the game front end if the user is playing or the agent is running.

e) Concepts and Algorithms Generated

The game's back end will be somewhat separate from the front end to keep code well organized, understandable, and easy to update.

f) Interface Description

Services Provided		
Service Name	**Services Provided To**	**Description**
update game	game front end, PPO, DQN	The game information will be updated and necessary information will be sent to the front end if the user is playing or the agent currently running.
provide game results	game results page	The game back end will send game results to the results page to be displayed to the user after they play the game or run one of the trained agents.

Services Required

Service Name	Service Provided From
game resources	Godot engine

II.2.6 Game Results Page

g) Description

The game results page will display the results of the game to the user. This will happen after the game is played by the user or after one of the trained agents is tested on the game. Information like the score, ending health, or time taken may be shown on this page.

h) Concepts and Algorithms Generated

The game results page will allow the user to see how they or the trained agent performed while playing the game. This will be important to determine how well the agent learned the game while training and the user will be able to compare their own score to the agent's.

i) Interface Description

Services Required	
Service Name	**Service Provided From**
UI tools (buttons etc.)	Godot UI
game results	Game Back End

III. Data Design

In order to train our agents, we need to be able to store their current policies. Storing the policies for our agents will allow training to be paused and resumed later by reusing a saved policy that has been partially trained. Storing our agents will also allow them to be tested with an already trained model rather than training a new model every time the user wants to test.

IV. User Interface Design

The user interface of our design is broken into three parts. The first piece is the game menu. The game menu shown in Figure 3 has buttons that allow the user to choose between playing

the game, training or testing an agent, testing a function player, viewing documentation for the agents or game, or exiting the program.

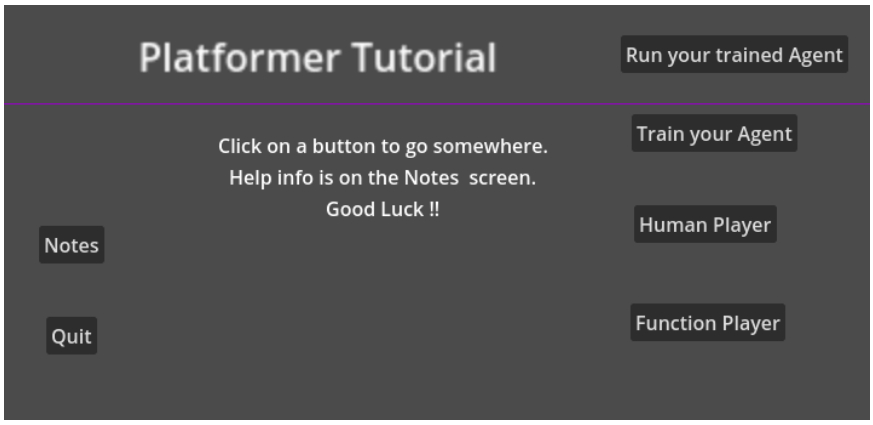


Figure 3: Main menu for our games in Godot.

The second part of our user interface will be shown while the user is playing the game. The user will be able to see important information for the game including their score. The last part of our user interface will be the results page shown in Figure 4. This page appears when the user or agent has completed the game.

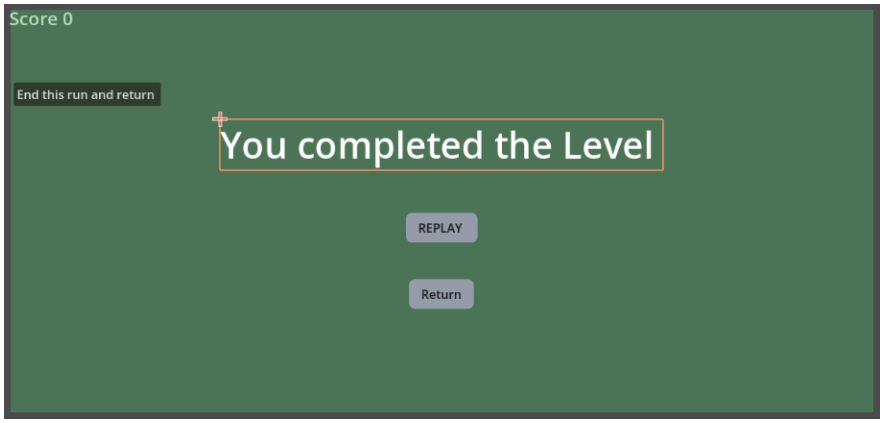


Figure 4: Result page of our games in Godot.

Testing and Acceptance Plan

Testing Overview For the Fall portion of this project we need to test the agents we train while in the Spring we will also need to test the games we create to provide reasonable certainty they can run without error.

Test Objectives and Schedule

The primary objective of the tests is to ensure that the RL agents can function correctly within the game environment, perform tasks as specified, and meet the educational objectives set forth by the project. The automated and manual tests will help ensure that the agents behave as expected, and that future updates do not break existing functionality.

Our tests will focus on ensuring all key functionalities operate as intended, from the RL logic to game-environment interactions. This is especially important as we move into user trials, where students will use the system to understand RL concepts. Automated tests will be used to streamline development, safeguard against regressions, and improve agility as we iterate based on feedback.

The project uses Godot, a game engine that presents its own set of challenges for testing. For unit testing, we will rely on built-in Godot testing frameworks, and integration testing will ensure RL agents function properly within the game. System testing will largely involve manual verification of game performance and RL behavior during gameplay.

The deliverables for this testing phase include the RL agents, full documentation of the testing process, and a CI pipeline. This CI pipeline will automate the testing of new code submissions and ensure continuous verification of the system's integrity.

Scope

The upcoming section outlines the strategy for testing the "Godot RML Agents" project, covering unit, integration, system, and user acceptance testing. The focus is on ensuring that the RL agents can complete their tasks in the game, as well as perform efficiently and consistently. It also includes a description of the testing tools and environments that will be used, along with timelines for the testing process.

I. Testing Strategy

1. Identify systems that need to be tested in the games we create.
2. Create tests for each system that requires testing.
3. Ensure that edge cases are tested and that there are enough tests to check the system properly.
4. Identify proper results for each test.
5. Document tests and expected results.
6. perform the tests.
7. Document how tests were performed and their results
8. Create an issue on the Kanban board in Git Hub if a test fails.
9. Fix the problem and mark the issue as complete in Git Hub.

II. Test Plans

II.1 Unit Testing

The game menu and results page will be run to ensure they are displayed correctly.

The game back end will be tested to make sure related values update properly.

The RML agents will be tested to ensure their policies can be updated properly and used to select moves.

II.2 Integration Testing

Both agents will need to be tested with the save models component to make sure their policies can be stored and loaded properly. This allows the agent training to be broken up and they can be trained ahead of time.

The game menu will be tested with the agents and game front end to ensure it moves the user to the proper page.

The Game results page will be tested with the game to make sure it displays the proper results from the game

Both agents will be tested to ensure they can train on the game back end. This will involve assessing their performance in the environment after training.

II.3 System Testing

System testing is a type of black box testing that tests all the components together, seen as a single system to identify faults with respect to the scenarios from the overall requirements specifications. The entire system is tested as per the requirements.

During system testing, several activities are performed:

II.3.1. Functional testing

To test the function of the system we will focus on the game, menus, and agents. When the system is fully connected we should be able to use the main menu screen to train or test the agents, play the game, or view documentation. All of these functions will need to be tested from the main menu.

Our first test will involve playing the game without one of the agents. The user will select the “play game” button on the game menu and then should be able to play the game without the agent. After the game, the user should be shown their results on the results page.

Our next test will be on training an agent. The agent should be able to run simulations on the environment and update its policy. This test will start with the user selecting the option to train the agent. After training the agent on several simulations the user will test the agent on the environment. After the test is complete the user should be shown the results for the agent’s test on the results page.

II.3.2. Performance testing

The performance of our agents will be based on their score in the game environment. After training on enough simulations the agent should be able to score higher than most people would be able to. We will compare agent scores on the environment to our own to see if there is a noticeable difference.

Our game should be able to handle one player or agent testing or training at a time. We will make sure this is the case by testing each of these functions.

II.3.3. User Acceptance Testing

In collaboration with the client, Ron Wright, we will conduct playtesting across several rounds to ensure the game and agents function as expected for educational purposes. The following outlines our user acceptance testing strategy, including necessary resources, step-by-step instructions, and a feedback loop to inform ongoing improvements.

Playtesting Process:

- Prepare Playtest Build: We will create a playtest-specific build of the game, ensuring it's optimized for client demonstration and efficient feedback gathering.
- Create Documentation for Current Build: Documentation covering the current state, features, and known limitations of this build will be provided to guide Ron through the showcase.
- Collect Feedback: During the showcase, we will present the game and RL agents directly to Ron, gathering immediate, verbal feedback on functionality, agent behavior, and engagement level.
- Analyze Feedback: Post-showcase, we will review the feedback notes to identify and prioritize areas for improvement.
- Adjust and Improve: Based on the feedback, we will make any necessary adjustments to enhance agent performance, gameplay clarity, and the overall user experience in preparation for the next sprint.
- This iterative process allows us to ensure that each build meets client expectations and aligns with the project's educational goals before final deployment.

III. Environment Requirements

The system software should include Windows, macOS, or Linux OS with Godot version 4.2 or higher, along with Python for supplementary scripting in OpenAI gym. There are no specific hardware requirements.

IV. Test Results

Aspect Being Tested	Expected Result	Observed Result	Test Result	Test Case Requirements
PPO agent on egg drop game	The agent should improve significantly after being trained	The agent starts scoring mostly negative points during episodes but improves to complete several 200+ point episodes	Pass	200k steps 2.5e-4 learning rate 128 batch size 0.99 gamma

DQN on coin collection game				
new game with enemies	The game should be completable with no observed bugs	The proper end screen is displayed and enemies/player stop when the game is complete	Pass	250 enemy speed
new game with moving platforms	The game should be completable with no observed bugs	The level completed screen is displayed when the coin is collected and platforms/player stop moving	Pass	700 platform speed

Projects and Tools Used

Tool/Library/framework	Used for
OpenAI gymnasium	modified lunar lander egg drop game
PyTorch	PPO agent
Godot	new games
NNET Godot	Neural network calculations in Godot engine

Languages used
Python
Godot Script

Final Prototype Description

During this project, we modified the OpenAI gymnasium lunar lander environment to be an egg drop-themed game, trained a PPO agent on the game, created a document describing the process of modifying the game and training the agent, and trained an agent on the daisy

collection game created by our client in godot. We also modified a simple game from a Godot tutorial to include enemies and moving platforms.

I. Egg Drop Game

The step in this section was to modify the colors of OpenAI's lunar lander environment to create an egg drop game. This involved creating a custom environment that inherits from the lunar lander environment class and changes the render and reset functions to change colors and shapes in the game. Figure 5 shows the original lunar lander environment while Figure 6 shows the modified lunar lander egg game.

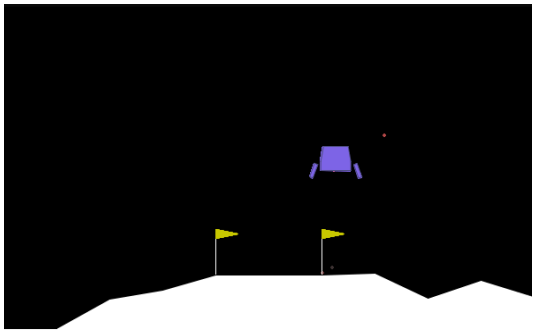


Figure 5: Original lunar lander

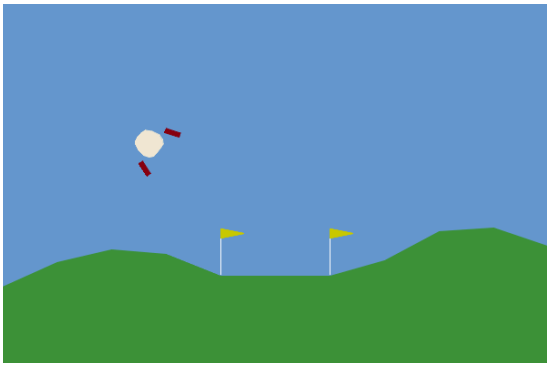


Figure 6: egg game

After this, the proximal policy optimization algorithm was trained on the environment using the existing reward/penalty system of the lunar lander game. The results of this training are shown below.

Figure 7 shows the number of points earned by the agent for each episode of training. An episode with over 200 points is considered a successful landing. The agent improved from scoring many negative points to scoring fewer negative points with several successful landings above 200 points.

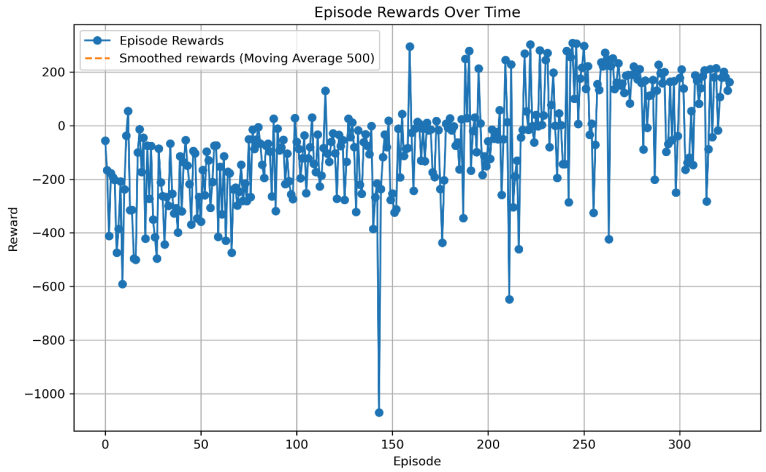


Figure 7: PPO algorithm rewards each episode while training (200k steps)

II. Deep Q-Network Prototype

II.1. Daisy Collecting Game

This step required implementing the DQN agent for the daisy collection game. The agent uses two neural networks: the Q-network for predicting the optimal actions and the Target network for stabilizing learning by providing a reference for Q-value updates. Figure 8 shows the agent training along with the users the elapsed time, how many points the agent collected, its velocity, and power node location.

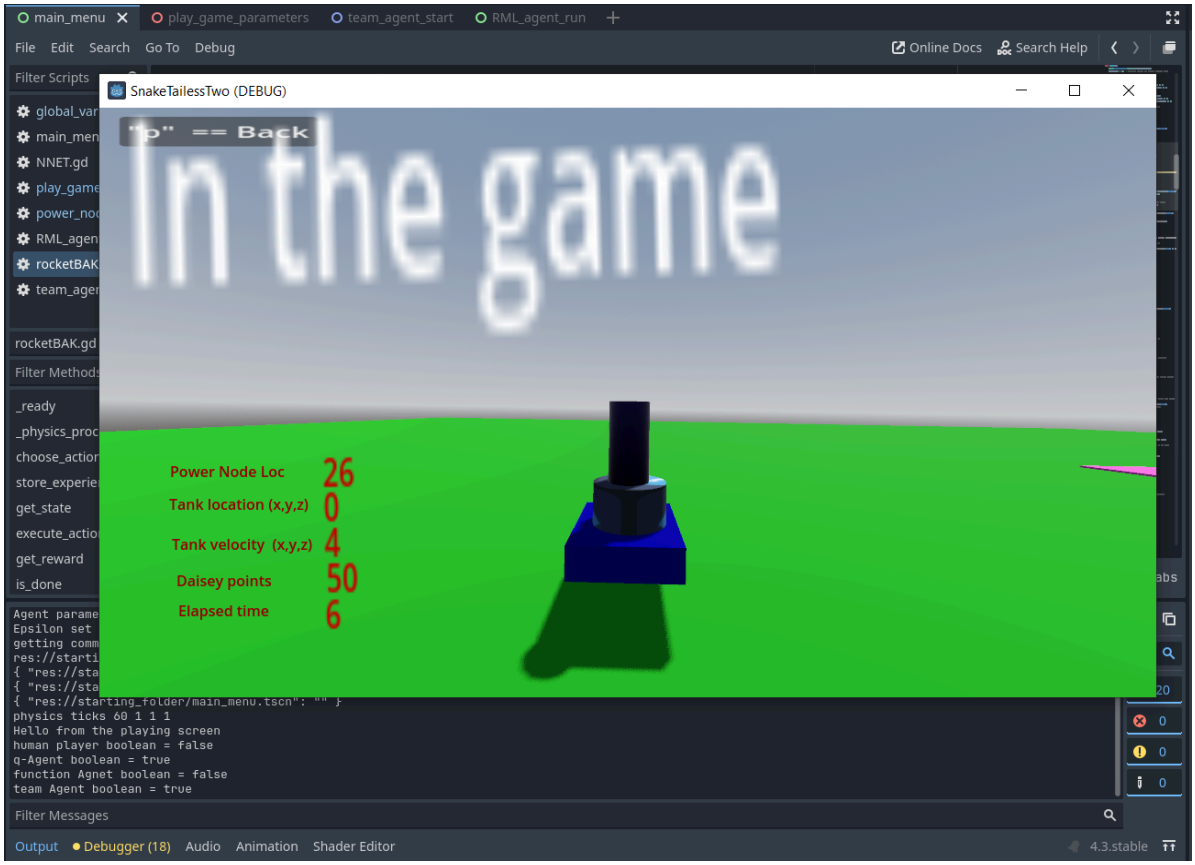


Figure 8: Image of the daisy collection game.

II.2 Subsystems Implemented

State Representation: Encodes the agent's position, velocity, and distance to the daisy into normalized inputs.

Action Selection: Implements an epsilon-greedy policy to balance exploration and exploitation during learning.

Reward System: Assigns rewards for moving closer to the daisy, penalties for moving away, and negative rewards for falling off the map.

Experience Replay: Maintains a replay buffer to store past experiences for batch training, reducing correlation in updates.

Training Loop: Includes batch training of the Q-network using Mean Squared Error (MSE) loss and periodic synchronization of the Target network with the Q-network.

Current progress: The DQN agent is functional, capable of interacting with the game environment, learning from its actions, and improving its performance over time. The remaining work includes optimizing the hyperparameters and further testing with different map configurations.

II.3. Preliminary Tests

We ran some basic tests on the DQN agent, and it can successfully move towards daisies most of the time and avoid falling off the map. Due to limitations with the Godot game engine, it was difficult to provide visualizations of training progress, but much of the testing was done through the console, where printing the change in loss or rewards slowly improved over time.

The DQN agent can read game states, pick actions, and apply rewards correctly. We also added a way to save the agent's progress, so it doesn't have to start over every time we test it.

III. Coin Game With Enemies

We modified a Godot tutorial game to include enemies that chase the player while they try to collect green coins. If the player collects all the green coins in the environment, the game ends and the player wins. However, if an enemy touches the player before they collect all the coins the player loses. Figure 9 shows the original tutorial game where the player collects coins with no enemies while Figure 10 shows the enemies added that chase the player.

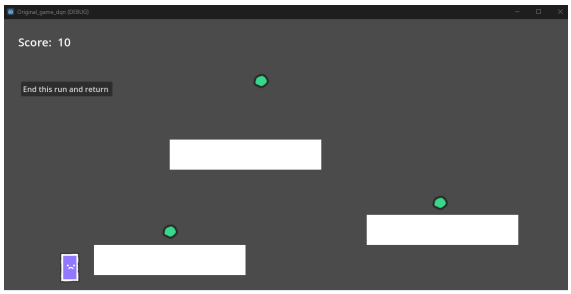


Figure 9: Original game from the tutorial.

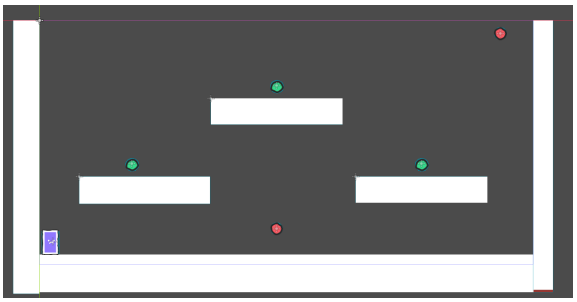


Figure 10: The game with enemies.

IV. Game With Moving Platforms

We modified the same tutorial game above to include platforms that move left and right between two walls. The player must jump on these platforms and reach the top without falling off to collect a single green coin. Figure 11 shows the modified game with moving platforms.

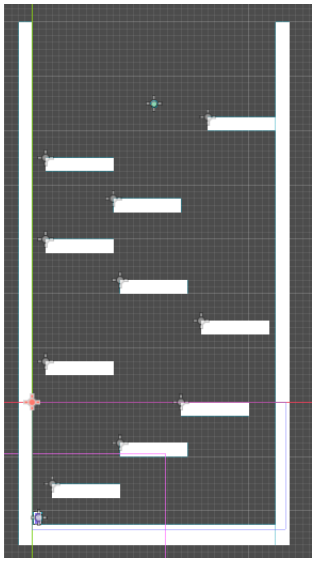


Figure 11: Modified game with moving platforms.

Product Delivery Status

We have completed the PPO agent on the modified lunar lander environment, the DQN agent in Godot for the Daisy collection game, and the two modified games. We still need to implement DQN agents to play the modified games and make tutorial videos showing our DQN agents and game modifications for the students. The completed PPO and DQN agents have been delivered through GitHub. They are in the main branch while the two modified games are in the new-game-dqn branch. Here is the link to our GitHub repository: https://github.com/luwke1/godot-agents/tree/main

Future Work

In the future, more games and game modifications could be made and used to create tutorial videos for students in the competition. More solutions could also be shown including different algorithms or more efficient ways to train agents. More resources would allow students to be more creative during the competition and learn more from the experience.

Acknowledgments

We thank our sponsor Ron Wright and Professor Parteek Kumar for helping us navigate and learn from this project.

Glossary

Artificial Intelligence (AI): A branch of computer science focused on creating systems capable of performing tasks requiring human intelligence, such as learning, decision-making, and problem-solving.

Deep Q-Network (DQN): A reinforcement learning algorithm that uses a deep neural network to approximate Q-values, representing expected rewards for actions in a given state.

Functional Requirement: A specification of behaviors or functions that a system must support, often associated with features or actions performed by the software.

Godot: A popular open-source game engine for creating 2D and 3D games, known for its flexibility and user-friendly interface.

Machine Learning (ML): A subset of AI involving training algorithms to learn from data and make decisions or predictions without explicit programming.

Non-Functional Requirement: Criteria describing how a system operates, such as performance, security, or usability, rather than its behaviors or functions.

Policy: In reinforcement learning, a strategy or set of rules an agent follows to decide actions based on the current state of the environment.

Proximal Policy Optimization Algorithm: A machine learning algorithm that uses neural networks to predict the best action from a given state and updates its weights based on the outcomes.

Q-values: In reinforcement learning, Q-values represent the expected total reward for an agent taking a specific action in a given state and following the optimal policy afterward.

Replay Buffer: A memory structure storing past experiences (state, action, reward, next state) for training, allowing random sampling to break correlations between consecutive experiences.

Reinforcement Learning (RL): A type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative rewards.

Target Network: In DQN, a copy of the main neural network is updated less frequently to stabilize training by providing consistent learning targets.

Use Case: A detailed description of how a user interacts with a system under specific conditions to achieve a particular goal.

References

1. “Code Bullet.” *YouTube*, YouTube, www.youtube.com/codebullet. Accessed 22 Sept. 2024.
2. “Introduction.” *Godot Engine Documentation*, docs.godotengine.org/en/stable/about/introduction.html#before-you-start. Accessed 22 Sept. 2024.
3. M. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, pp. 484–489, Jan. 2016.
4. OpenAI, "Dota 2 with large-scale deep reinforcement learning," OpenAI, Dec. 2019. https://openai.com/research/dota-2-with-large-scale-deep-reinforcement-learning.
5. "Introduction." Godot Engine Documentation. Accessed 22 Sept. 2024. [Available: https://docs.godotengine.org/en/stable/about/introduction.html].
6. “Use Case.” *Wikipedia*, Wikimedia Foundation, 7 Sept. 2024, en.wikipedia.org/wiki/Use_case.
7. “Functional Requirement.” *Wikipedia*, Wikimedia Foundation, 29 July 2024, en.wikipedia.org/wiki/Functional_requirement.
8. “Non-Functional Requirement.” *Wikipedia*, Wikimedia Foundation, 15 Sept. 2024, en.wikipedia.org/wiki/Non-functional_requirement.
9. AI Wiki, "Deep Q-Network (DQN)," AI Wiki, 2013. [Online]. Available: https://aiwiki.ai/wiki/Deep_Q-Network. [Accessed: 19- Oct- 2024].
10. "DQN," LessWrong, 2015. [Online]. Available: https://www.lesswrong.com. [Accessed: 19- Oct- 2024].
11. "Introduction to RL and Deep Q Networks," TensorFlow, 2023. [Online]. Available: https://www.tensorflow.org. [Accessed: 19- Oct- 2024].
12. “Getting Started With Godot 4 | Godot 4.1 Tutorial.” *YouTube*, Muddy Wolf, 18 Sept. 2023, www.youtube.com/watch?v=gnboGSpjHVQ&t=612s&pp=ygUZbXVkZHkgd29sZiBnb2RvdCB0dXRvcmlhbA%3D%3D.

I. Appendix A – Team Information

Luke

Cole

II. Appendix B - Example Testing Strategy Reporting

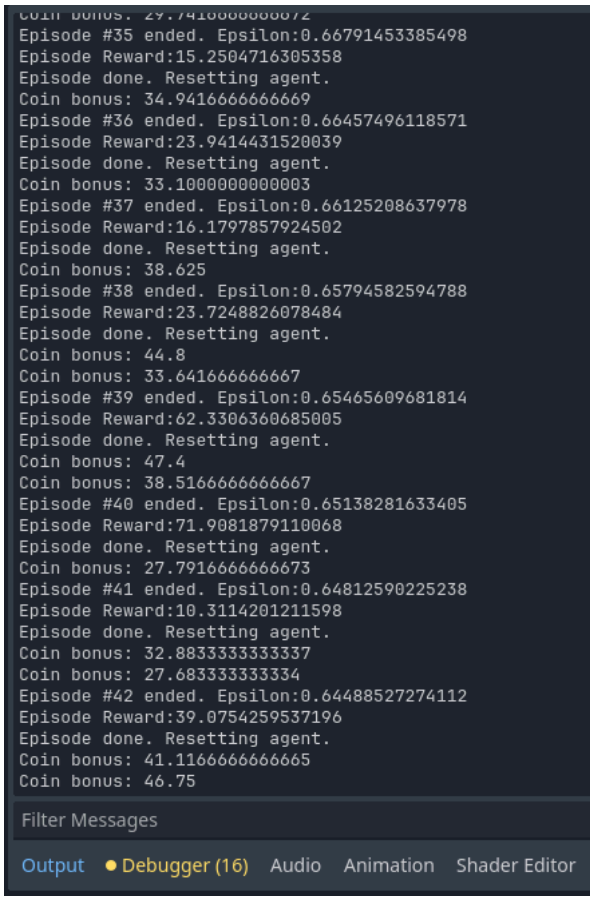
1) Requirements Tested

- **Functional Requirements:**
  - FR-1 (PPO for egg drop game)
  - FR-2 (DQN for daisy collection game)
  - FR-3 (Two games with splash pages and results screens)
- **Non-Functional Requirements:**
  - NFR-1 (Bug-free implementation)
  - NFR-2 (Complete documentation)

- NFR-3 (Game simplicity)

2) Test Results

- **Automated/Manual Testing:**
  - **PPO Agent on Egg Drop Game:**
    - Expected: Agent improves to score >200 points.
    - Observed: Agent achieved 200+ points after 200k training steps (see Test Results Table, p. 27).
  - **DQN Agent on Coin Collection Game:**
    - Testing via console logs confirmed decreasing loss and improving reward metrics.
  - **Game Modifications (Enemies & Moving Platforms):**
    - Confirmed bug-free completion with end screens (test cases: 250 enemy speed, 700 platform speed).
- **Screenshots:**
  - Example: Training progress visualization for PPO (Figure 7, p. 28).



Episode	Reward	Collected	Epsilon	Time
1	-0.86589	0	0.796	25.13333
2	10.18363	1	0.79202	41.81667
3	15.29272	1	0.78806	34.45
4	10.52342	1	0.78412	42.68333
5	10.28169	1	0.780199	42.46667
6	16.42203	1	0.776298	31.41667
1	14.89991	1	0.796	32.93333
2	17.11062	1	0.79202	30.55
3	-1.72545	0	0.78806	25.13333
4	12.89469	1	0.78412	39.21667
1	-2.10537	0	0.796	25.13333
2	15.0584	1	0.79202	34.66667
3	10.41134	1	0.78806	43.55
4	-1.24881	0	0.78412	25.13333
5	-1.8273	0	0.780199	25.13333
6	7.109536	1	0.776298	48.75
7	-1.90452	0	0.772417	25.13333
8	-1.07463	0	0.768554	25.13333
9	17.97134	1	0.764712	28.6
10	-1.07319	0	0.760888	25.13333
11	12.7838	1	0.757084	35.31667
12	10.53228	1	0.753298	39
13	-1.96849	0	0.749532	25.13333
14	-1.04626	0	0.745784	25.13333
15	-1.75643	0	0.742055	25.13333
16	8.277013	1	0.738345	46.58333
17	15.61906	1	0.734653	32.28333
18	-1.83808	0	0.73098	25.13333

3) User Testing Feedback

Not much user testing has been completed. We have only shown our progress to our project mentor Ron Wright. He has been pleased with the progress and documentation.

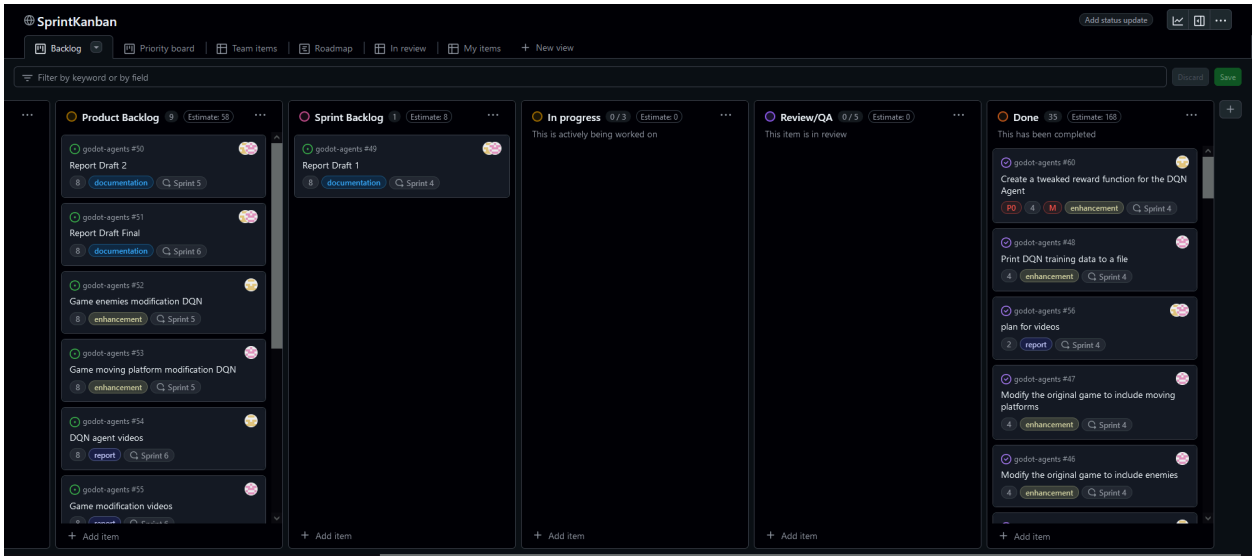
II. Appendix C - Project Management

Weekly Schedule

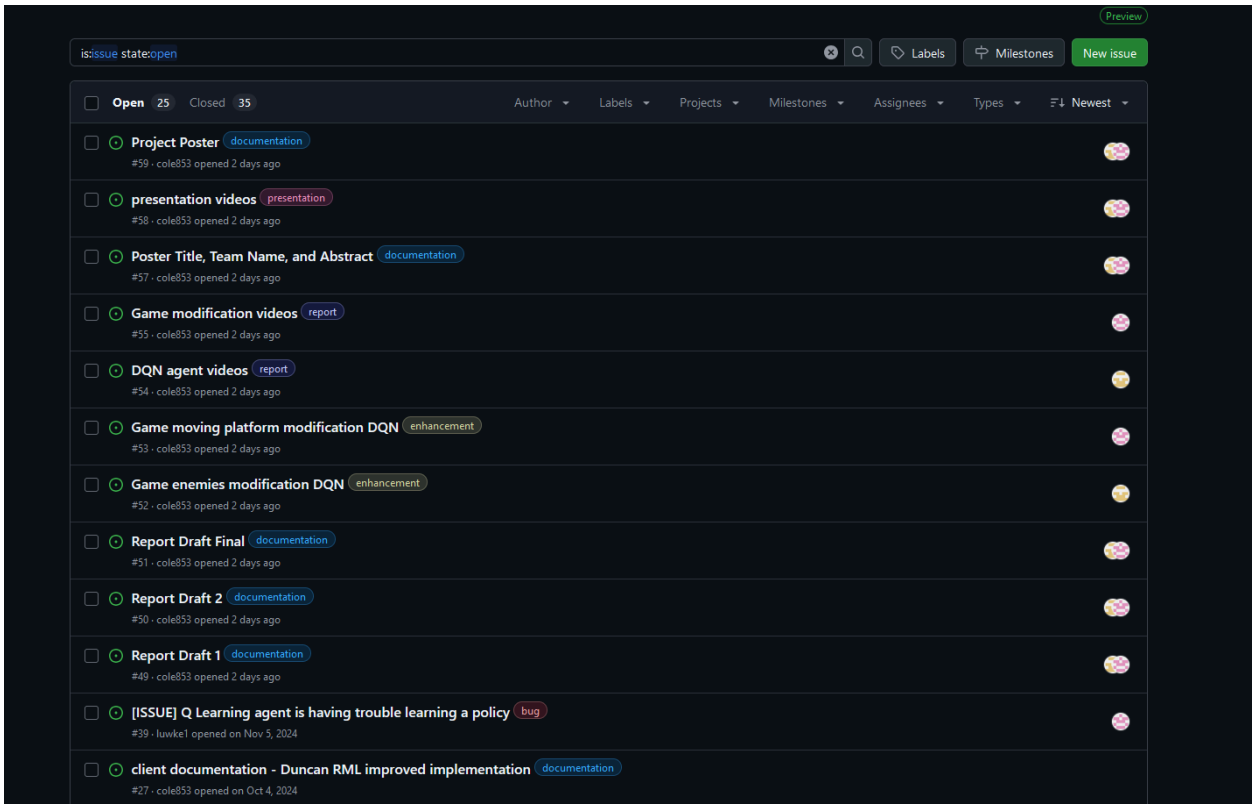
1. **Weekly Mentor Meetings:**
    - **Purpose:** Align with client (Ron Wright) on priorities, review sprint goals, and address technical blockers.
    - **Agenda:**
        - Progress updates on agents/games.
        - Feedback on documentation and curriculum design.
        - Adjustments to sprint timelines.
2. **Biweekly Team Internal Meetings:**
    - **Purpose:** Task allocation, code reviews, and troubleshooting.
    - **Agenda:**
        - Review GitHub issues (e.g., DQN integration challenges).
        - Plan testing strategies for upcoming deliverables.
3. **Sprint Cadence:**
    - **Start of Sprint:** Kickoff meeting to define goals (e.g., “Complete DQN hyperparameter tuning”).
    - **Mid-Sprint Check-ins:** Every 2 weeks to demo progress (e.g., DQN training results).
    - **End of Sprint:** Final review with client to validate deliverables and plan next sprint.

Most Beneficial Activities:

- **End-of-Sprint Reviews:** Critical for aligning with client expectations and refining the next sprint’s scope.
- **GitHub Issue Tracking:** Streamlined task prioritization and accountability.



(image 1)



(image 2)