Godot RML Agents

Creating RL Agents to solve games on the Godot engine

Sponsor: Ron Wright

{TEAM NAME UNDECIDED}

Luke Flock and Cole Clark

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 in order 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 Learning Machine (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 own agents and apply theoretical concepts in a fun, practical environment.

Wahkiakum School District is pioneering this initiative as part of a K-12 curriculum strand focused on equipping students with RL skills. The project requires the creation of multiple RML agent solutions that can serve as benchmarks for student teams. These agents will be tested during a Trial Event, with the ultimate goal of refining the event for potential use at the WA State 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.

I.1 Background and Related Work

This project falls within the domain of applying RL techniques to video games, a field that has gotten a significant amount of 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 will focus on translating current RL techniques into an educational context, making it accessible for 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 be creating a game for next year's competition as well.

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.

Monte Carlo Reinforcement Learning

- The team must understand how Monte Carlo methods apply 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.

I.2 Project Overview

In the last few years, it has become increasingly popular to develop reinforcement machine learning algorithms to play games. Many videos on YouTube show algorithms that have learned to play Pacman, Flappy Bird, Pokemon, Chess, and Pool along with several other games (“Code Bullet”). With the increasing interest in RML, our project is focused on helping our client teach the concepts of reinforcement machine learning to middle 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 first solution to this game will use the Monte Carlo algorithm. This will involve using a sample of simulations to develop a policy for which actions the agent should take. The agent will need to make different decisions depending on its distance from the next daisy, the direction to the daisy, its position on the board, and the time it has been playing.

Our second solution will use a different algorithm to complete the same game. The purpose for creating multiple algorithms is to provide our client with a variety of solution methods. This will allow Ron to better prepare for the competition and give him a better idea of how to grade different solutions.

In the second part of our project, we will be creating games for next year's competition. It will be important to make games that are fun to play with the right level of complexity. The games will need to be simple enough that students can create RML agents to play them while being complex enough to allow a wide variety of solutions. Support documents for these games along with well-documented code will be needed to ensure that students can spend their time learning about RML rather than struggling to understand the game.

We will develop our games using the Godot game engine. The Godot game engine is free to use and gives users full ownership of the games they develop using the engine ("Introduction"). This will allow us to create the games for free without worrying about how they are used in the competition.

Games completed for this second part of the project will need to be different from each other so that there is a variety to choose from. This will increase the variety of solutions and give students more ideas for the game they design and solve. Our solutions will need to be well documented to help the students build on their understanding of programming languages, game design, and reinforcement machine learning.

I.3 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 be done to 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.

In the spring, it will be even more important to recognize the students as stakeholders since we will be creating games to be used in next year's competition. 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.

Project Requirements

II. System Requirements Specification

This section contains the functional requirements, nonfunctional requirements, use cases, user stories, and the impact of system evolution for our project.

II.1. Functional Requirements

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

II.1.1. Two Reinforcement Machine Learning Solutions

Functional Requirement	[FR-1] Monte Carlo Agent
Description	The first reinforcement machine learning model to solve our client's daisy collecting game will use the Monte Carlo algorithm. To train, the algorithm must complete simulations that represent the population of situations the agent might be in. After the end of each simulation, the model will update weights for each position in its path during that simulation depending on the reward it received.
Source	Ron Wright (client) requested a Monte Carlo solution
Priority	0 Essential and required functionality

Functional Requirement	[FR-2] Policy Gradient Agent
Description	In addition to a Monte Carlo solution for the daisy collection game, we will create a second agent. This agent will use a policy gradient RL algorithm and will be implemented into the Godot game given.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

II.1.2. Two Games

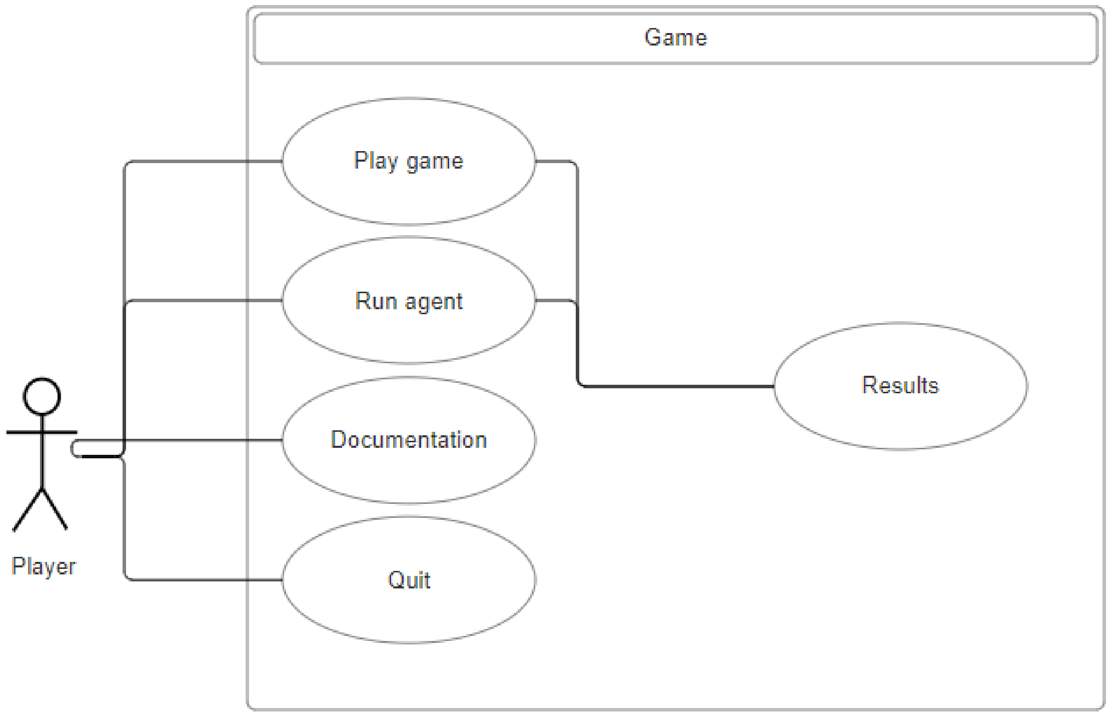
Functional Requirement	[FR-3] Two different games
Description	Both games need to have a starting splash page and an ending splash. The ending page will show results after the game is completed by a player or agent. The games must be from different genres.
Source	Ron Wright (client)

Priority	0 Essential and required functionality

II.2 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.

II.3 Use Cases



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

II.4 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

II.5. Traceability Matrix

The table 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 Monte Carlo agent	UC-2: Train RL Agent UC-3: Test RL Agent	US2: Train RL Agent US3: Test RL Agent	0

FR-2 Policy Gradient 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

II.6. 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.

III. Glossary

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

Machine Learning (ML): A subset of AI that involves training algorithms to learn from data and make decisions or predictions based on that data without being explicitly programmed.

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

Monte Carlo Algorithm: A reinforcement learning algorithm that makes decisions based on the average outcome of simulations run from a particular state, updating its policy based on rewards received at the end of each simulation.

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

Godot: A popular open-source game engine that is used for creating 2D and 3D games. It is known for its flexibility and user-friendly interface, making it suitable for educational projects.

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

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

Non-Functional Requirement: Criteria that describe the operation of a system rather than its behaviors or functions, such as performance, security, or usability requirements.

IV. 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.