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

I. Introduction

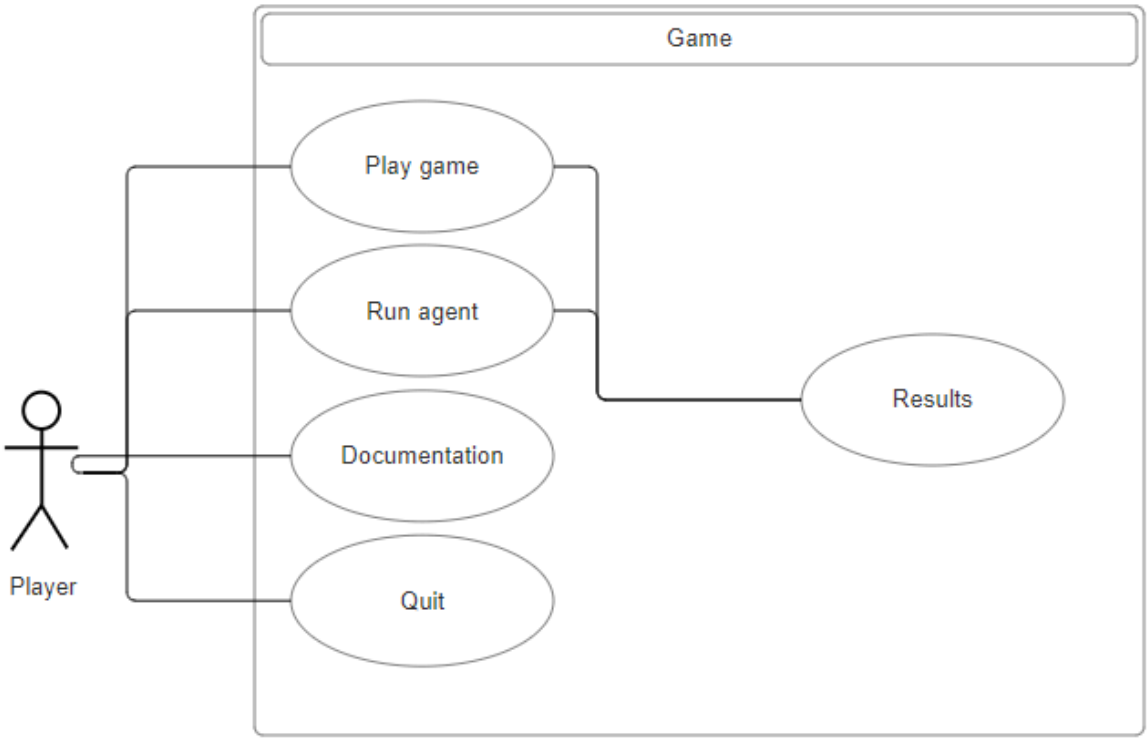
The recent advancements of machine learning (ML) and artificial intelligence (AI) have made it an essential skill to learn in order to stay competitive in the future workforce. The goal is to introduce foundational knowledge of AI and ML into a K-12 curriculum as it currently lacks teaching students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

II. System Requirements Specification

II.1. Use Cases



Play game

Pre-condtion	- The user is on the main menu
Post-condition	- Results are shown for the players activity in the game
Basic Path	1) 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

Train RL Agent

Pre-condtion	- The user is on train RL menu, and selects 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) 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 - Game menu screen an

Run trained RL agent

Pre-condtion	- The user has trained the RL agent, and the user is 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 agents score is calculated

Alt Path	- At step 2, the user could have no training data, so indicate that the user needs to train the agent first
Related Requirements	- RL agent training data - Game menu screen - Game mechanics

View Documentation

Pre-condtion	- The user is on game menu screen, and they clicked view documentation button
Post-condition	- Documentation and explanation for RL algorithm implementations is shown - Rules for the game are also shown
Basic Path	1) User clicks “View Documentation” button 2) Lets the user view 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

Quit

Pre-condtion	- The user is on the main menu
Post-condition	- The game is closed
Basic Path	1) User selects “Quit” from the main menu 2) The game is closed
Alt Path	-
Related Requirements	- A main menu

Results

Pre-condtion	- 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

(Note that the diagrams will not be counted for the 3 pages text length specified for this document.)

II.2. Functional Requirements

II.2.1. Two Reinforcement Machine Learning Solutions

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

Another agent:

Description	In addition to a Monte Carlo solution for the daisy collection game we will create a second agent. This agent will not use the Monte Carlo algorithm or the Q reinforcement machine learning algorithm.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

Train agents:

Description	Both agents will need training to complete the game. Both will have some form of policy that they will use to make decisions. This policy will be updated while training until the agent is able to use it to play the game.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

Test agents:

Description	Once the agents have been trained they should be able to play the game. The agents should be able to play the game at least as well as a human. This will involve moving the character in order to collect daisies then moving

	back to park on the starting platform.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

II.2.2. Two Games

Starting splash page:

Description	Both games will need to have a starting splash page with the following buttons: “help” - showing the rules for the game “Quit” - allowing the player to exit the game “Play” - allowing the player to start the game “Agent” - runs the game with the agent playing
Source	Ron Wright (client)
Priority	0 Essential and required functionality

Ending splash page:

Description	The ending splash page will be displayed after the game is run either with a human player or RML agent. It will show the results of the game and how well the player or agent did.
Source	Ron Wright (client)
Priority	0 Essential and required functionality

II.3. Non-Functional Requirements

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 bugs we left. Our RML agents should function without errors to give Ron (client) a benchmark for student projects.

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 in 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.

Simple Games:
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.

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.

Godot:
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. Using Godot will make it easier for students to learn from our game examples since it will be more similar to what they are doing than if we used another game engine.

III. 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.

IV. 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.

V. References

"Use Case." *Wikipedia*, Wikimedia Foundation, 7 Sept. 2024, en.wikipedia.org/wiki/Use_case.

"Functional Requirement." *Wikipedia*, Wikimedia Foundation, 29 July 2024, en.wikipedia.org/wiki/Functional_requirement.

"Non-Functional Requirement." *Wikipedia*, Wikimedia Foundation, 15 Sept. 2024, en.wikipedia.org/wiki/Non-functional_requirement.