**Super MaRLo Bros**

Project Report

Project X Mentorship Programme

at

Community of Coders, Veermata Jijabai Technological Institute

October 2024

**ACKNOWLEDGEMENT**

Firstly, we would like to thank our mentors, **Vedant Mehra** and **Labhansh Naik** for their commendable role as our mentor throughout this project. Their timely guidance, expertise in the domains and insightful feedback were crucial in shaping our project.

We are also grateful to VJTI’s COC for continuing the tradition of their seniors and providing this golden opportunity that made our project possible.

**Mentors**

Labhansh Naik

labhanshvnaik2104@gmail.com

Vedant Mehra

[@gmail.com](mailto:mvpawar2005@gmail.com)

**Mentees**

Vidhi Rohira

[rohiraviidhi@gmail.com](mailto:rohiraviidhi@gmail.com)

Ayush Bothra

[abc@gmail.com](mailto:abc@gmail.com)

Niharika Hariharan

[abc@gmail.com](mailto:abc@gmail.com)

1. **OVERVIEW**

**Super MaRLo Bros Project**

The aim of this project is to develop an agent that can learn to play a customized game environment inspired by the classic *Super Mario Bros* using reinforcement learning (RL). The project builds on the Malmo platform, which is designed for AI experimentation in the context of games. Reinforcement learning algorithms allow agents to learn optimal behaviours through repeated interactions with their environment. By receiving feedback in the form of rewards or penalties based on their actions, the agent gradually improves its decision-making process, ultimately aiming to maximize its cumulative reward.

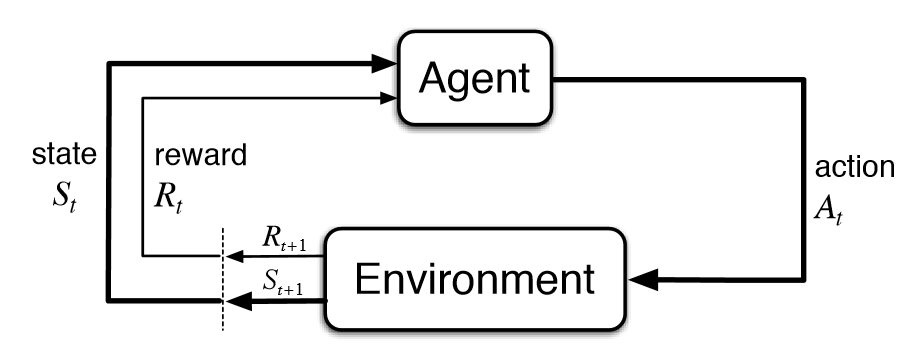
In this project, the environment has been created using the Minecraft-based Malmo environment. The game consists of multiple levels with stationary and dynamic obstacles, where the agent must navigate and interact with these elements to progress and learn efficient strategies. The agent is trained using algorithms from the StableBaselines3 library, which provides a variety of RL algorithms designed for continuous learning in complex environments.

We employed both Multi-Layer Perceptron (MLP) networks and Convolutional Neural Networks (CNNs) to process the observations from the environment. Initially, we used the Deep-Q-Network (DQN) algorithm, but later transitioned to the Proximal Policy Optimization (PPO) algorithm, which provided better performance in terms of stability and efficiency.

1. **Some Terminology**

**1] Reinforcement Learning:**

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent receives feedback in the form of rewards or penalties based on its actions and aims to maximize its cumulative reward over time. Unlike supervised learning, where the correct answer is provided for each example, RL involves learning through trial and error. The agent explores various actions and learns from the outcomes to develop optimal strategies. RL is widely used in areas such as robotics, game playing, and autonomous systems for solving complex decision-making tasks.



**2] Pygame:**

Pygame is an open-source Python library designed for creating 2D games and multimedia applications. It provides a set of modules to handle graphics, sound, input devices, and more, making it easier to develop games with Python. Pygame simplifies rendering images, animating objects, and detecting collisions, while also managing game loops and real-time user interactions. It's widely used for prototyping, hobbyist game development, and educational purposes due to its simplicity and flexibility. Although primarily focused on 2D games, it can also be used for basic simulations and interactive graphical programs.

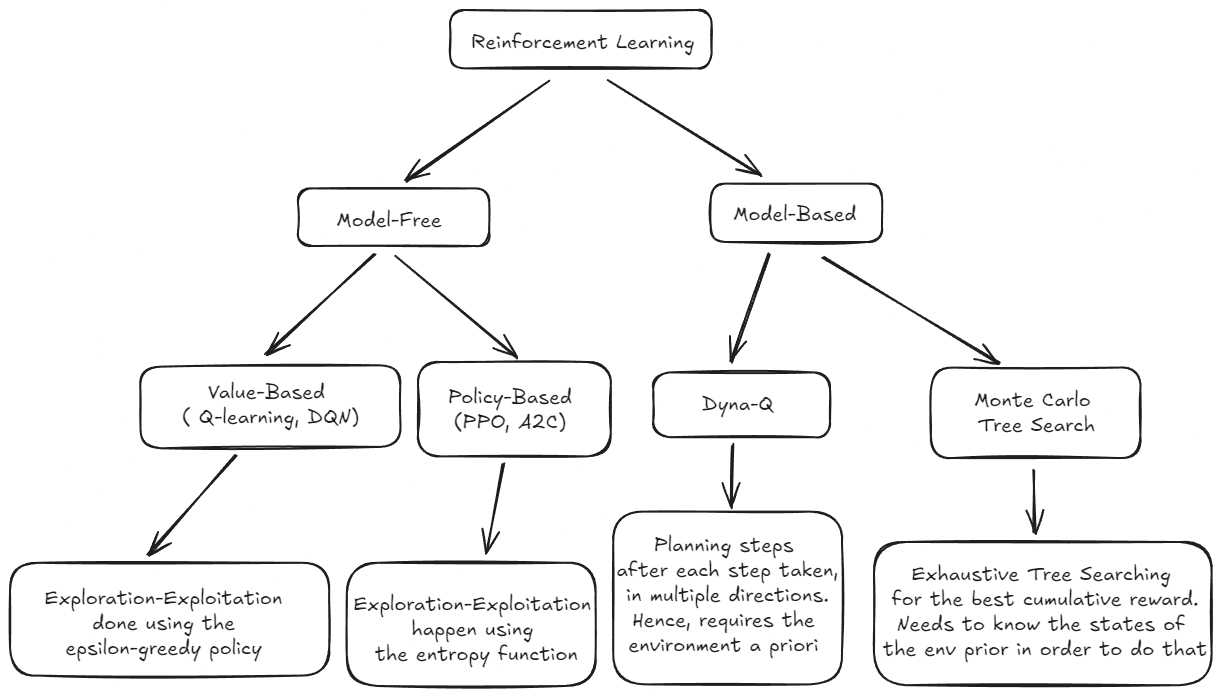
**3] Game Development:**

Game development is the process of designing, creating, and producing a video game. It involves several stages, including concept development, game design, programming, asset creation (such as graphics, sound, and animations), and testing. Game developers use various tools and technologies, like game engines (e.g., Unity, Unreal Engine) and programming languages, to build the game's mechanics, visuals, and user interface. Collaboration between designers, artists, and programmers is key to transforming an idea into an interactive experience. Game development can range from small indie projects to large-scale productions with complex gameplay, immersive storytelling, and advanced graphics.



**4. Reinforcement Learning**

Reinforcement learning (RL) is a machine learning framework where an agent learns optimal behaviours by interacting with an environment. The agent receives feedback in the form of rewards or penalties, with the goal of maximizing its cumulative reward over time.



### Exploration vs. Exploitation Dilemma

### A fundamental challenge in RL is the balance between exploration and exploitation:

* **Exploration** refers to the agent trying out new actions to discover their potential rewards, even if these actions have uncertain outcomes. This helps the agent gather information about the environment, which can lead to better long-term rewards.
* **Exploitation** occurs when the agent chooses actions it already knows will give the highest reward, based on its past experiences. This maximizes the agent’s immediate reward but may prevent it from discovering potentially better strategies.

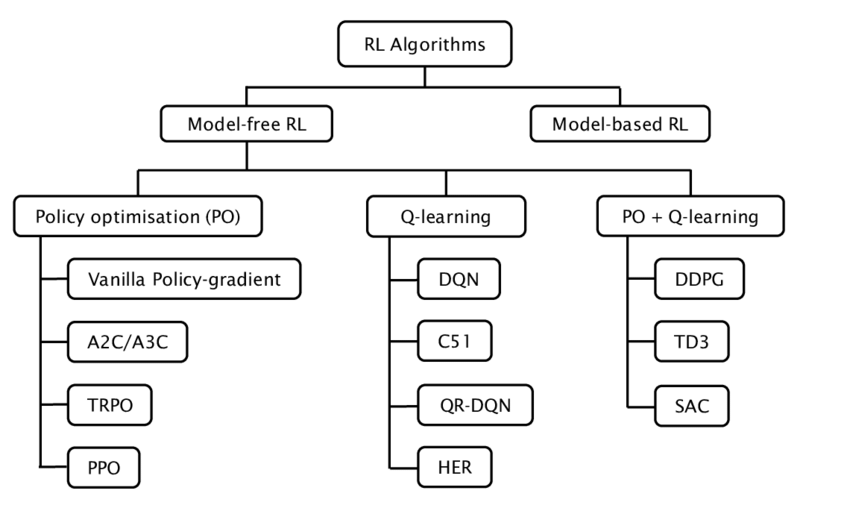
The **exploration-exploitation trade-off** requires the agent to find a balance between exploring new actions to improve its knowledge and exploiting what it has already learned to maximize rewards.

#### Greedy and Epsilon-Greedy Methods

One of the simplest ways to manage the exploration-exploitation trade-off is through **greedy** and **epsilon-greedy** methods:

* **Greedy Method:** The agent always selects the action with the highest expected reward (exploitation). This can lead to suboptimal behavior if the agent hasn’t fully explored the environment.
* **Epsilon-Greedy Method:** To promote exploration, the agent occasionally selects a random action (exploration), while most of the time, it chooses the action with the highest reward. The parameter **epsilon (ε)** controls the probability of taking a random action. A small epsilon (e.g., 0.1) encourages mostly exploitation, with occasional exploration to avoid getting stuck in local optima.



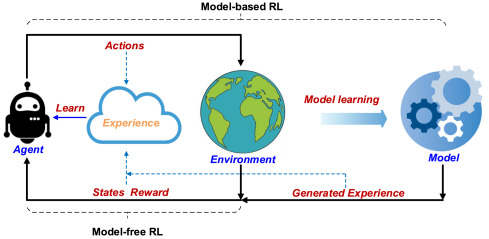


### 1.] Model-Based RL

In model-based RL, the agent builds an internal model of the environment, which it uses to predict future states and rewards. This enables the agent to plan its actions by simulating different strategies before acting. Model-based methods naturally involve more **exploitation** since the agent leverages its model to choose optimal actions, but the agent can explore using the model to simulate various future scenarios without interacting directly with the environment.

**Examples of Model-Based Algorithms:**

* **Dyna-Q:** Combines elements of model-free RL with planning. The agent learns both a model of the environment and updates its value function, using simulated experiences to balance exploration and exploitation.
* **Monte Carlo Tree Search (MCTS):** Often used in game-playing AI (e.g., AlphaGo), MCTS builds a search tree of future actions, balancing exploration of promising moves (by expanding different branches of the tree) with exploitation of the known best paths.



### 2.] Model-Free RL

Model-free RL doesn’t require the agent to build an explicit model of the environment. Instead, the agent learns directly from interactions with the environment, relying heavily on the **exploration-exploitation trade-off** to navigate and learn the best strategy. Model-free RL methods are popular for their simplicity and ability to scale to complex environments where it’s impractical to build an accurate model.

#### Value-Based Methods

Value-based methods estimate the expected future reward (value) for each state-action pair. These methods are typically based on **Q-learning**, where the agent learns a Q-value function that helps it choose the best action. Exploration and exploitation are managed by techniques like epsilon-greedy.

* **Q-Learning:** A model-free method that updates a Q-table based on rewards received from taking actions. Q-learning typically uses the epsilon-greedy strategy to ensure that the agent explores new actions while also exploiting learned knowledge. Over time, the agent converges on the optimal policy.
* **Deep Q-Network (DQN):** A deep learning extension of Q-learning that uses a neural network to approximate the Q-values in environments with large state spaces. DQN uses epsilon-greedy for exploration and periodically reduces epsilon as the agent learns.

#### Policy-Based Methods

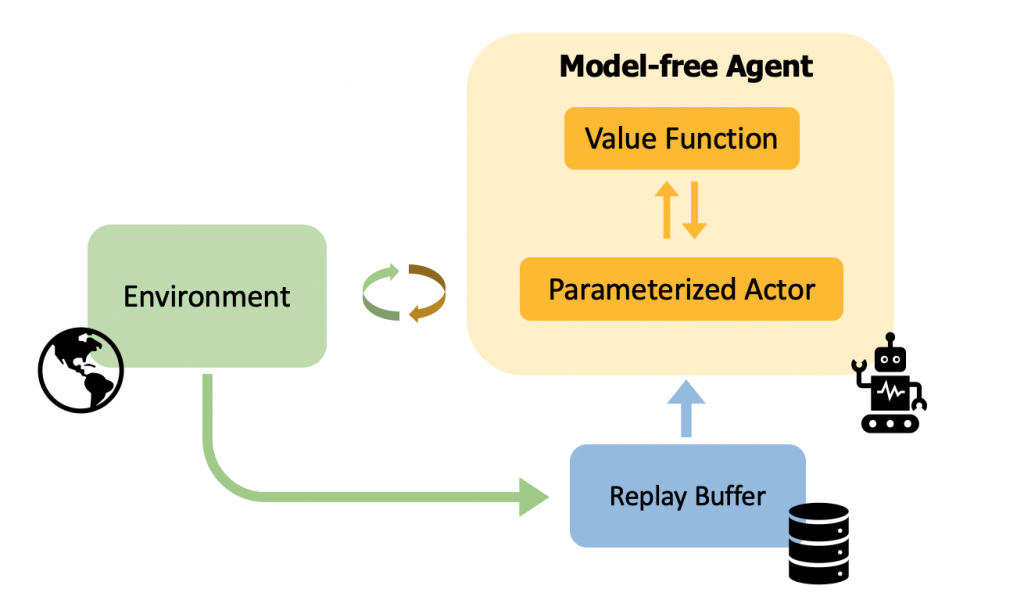
Policy-based methods directly learn a policy (a mapping from states to actions), which the agent optimizes using techniques such as gradient ascent. These methods can handle continuous action spaces better than value-based methods.

* **REINFORCE:** A Monte Carlo-based algorithm that updates the policy by maximizing the expected return. Since policy-based methods focus on optimizing the policy itself, they can naturally include exploration by adjusting how frequently certain actions are chosen based on their potential reward.
* **Proximal Policy Optimization (PPO):** PPO is widely used for its stability and efficiency. It adjusts the policy incrementally to ensure that the agent doesn’t make drastic updates, thus balancing exploration with stable exploitation of the current best-known actions.

#### Actor-Critic Methods

Actor-critic methods combine both value-based and policy-based approaches. The **actor** learns the policy, and the **critic** learns the value function, providing feedback to the actor on how to improve the policy.

* **Advantage Actor-Critic (A2C):** This algorithm balances exploration by having the critic evaluate the advantage of the current action compared to the baseline value, helping the actor update its policy in a more stable way.
* **Deep Deterministic Policy Gradient (DDPG):** Suitable for continuous action spaces, DDPG combines the benefits of deep Q-learning and policy gradient methods to effectively explore and exploit actions.



**5. Approaches to accomplish the Project**

For our project, **Super MaRLo Bros**, we implemented a reinforcement learning (RL) architecture using a **model-free algorithm** to train an AI agent capable of playing the game. This model-free approach allows the agent to learn optimal strategies through trial and error, enhancing its performance over time without the need for a predefined model of the game environment. Such algorithms are well-suited for handling the dynamic nature of gameplay, enabling the agent to adapt its strategies based on its interactions with the environment.

#### Model-Free Reinforcement Learning

In reinforcement learning, **model-free algorithms** allow the agent to learn decision-making strategies without requiring an explicit model of the environment. This means the agent doesn't attempt to predict how the environment behaves in response to its actions (i.e., state transitions or rewards). Instead, it directly learns from real-world experience to maximize cumulative rewards.

##### Key Characteristics of Model-Free Algorithms:

* **No Environment Model:** The agent doesn't try to understand how the environment works internally, such as predicting state transitions or rewards.
* **Direct Policy or Value Learning:** The agent focuses on learning either a **policy** (mapping states to actions) or a **value function** (estimating the expected long-term reward for states or actions).
* **Data-Driven:** The agent improves its decision-making purely from observed interactions (state transitions and rewards), instead of simulating or predicting the environment.

#### Types of Model-Free Algorithms:

1. **Value-Based Methods:** These methods aim to estimate a value function, which tells how beneficial it is to be in a certain state or to take a specific action.
   * **Example:** **Q-Learning**  
     Q-learning is a widely used value-based method in which the agent learns the Q-value (action-value function), predicting the expected reward of taking an action in a specific state and following the optimal policy afterward.
2. **Policy-Based Methods:** These methods directly learn the policy (a function that maps states to actions) without needing to estimate value functions.
   * **Example:** **REINFORCE**  
     REINFORCE uses **policy gradients** to update the policy directly based on the rewards received by the agent.
3. **Actor-Critic Methods:** Actor-critic methods combine both value-based and policy-based approaches, consisting of an **actor** (which selects actions) and a **critic** (which evaluates how good those actions are).
   * **Example:** **Proximal Policy Optimization (PPO)**  
     PPO is a popular actor-critic algorithm that balances simplicity, efficiency, and performance by maintaining stable policy updates.

### Proximal Policy Optimization (PPO)

In our project, we utilized **Proximal Policy Optimization (PPO)**, a policy-based algorithm with policy function approximation. PPO was chosen for its robust performance and ability to address the instability issues seen in earlier policy gradient methods.

PPO ensures stable learning by clipping the probability ratio between the new and old policies during training, preventing excessive policy updates that could degrade performance. This **clipping mechanism** enables smoother and more reliable training compared to algorithms like **vanilla policy gradient** or **trust region policy optimization (TRPO)**, while maintaining computational efficiency.

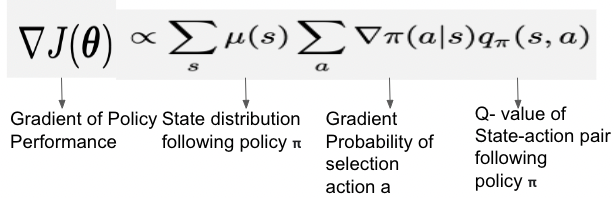
#### Why PPO?

* **Stable Learning:** By limiting the magnitude of policy updates, PPO minimizes instability during training.
* **Efficiency:** It is computationally efficient and easier to implement than many other advanced RL algorithms.
* **Flexibility:** PPO works well across a variety of tasks, including both continuous and discrete action spaces, making it ideal for challenges such as robotic control, game playing, and navigating complex environments.

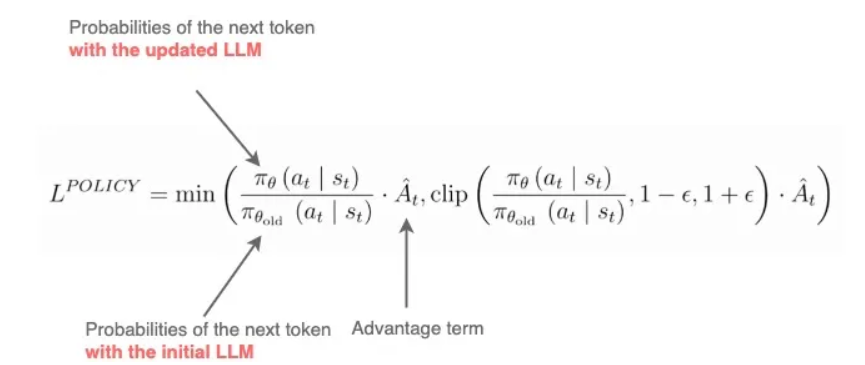
The combination of these strengths makes PPO an optimal choice for training our AI agent in **Super MaRLo Bros**, ensuring that the agent adapts to the game's complexity while maintaining stable learning and improving its performance over time.

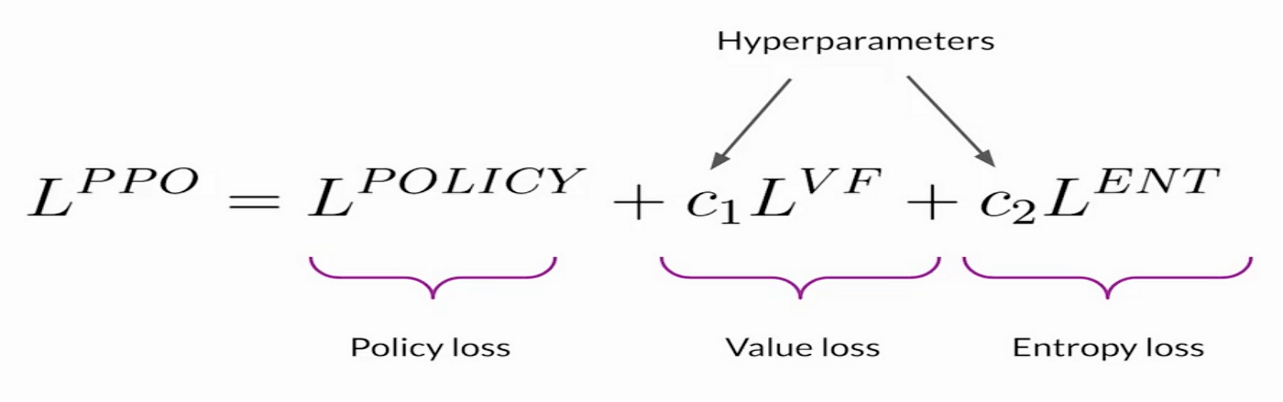
### 1. Basic Policy Approximation Formula (Policy Gradient)

The basic policy gradient can be expressed as:



### 2. Proximal Policy Optimization (PPO) Formula





**Custom Environment:**

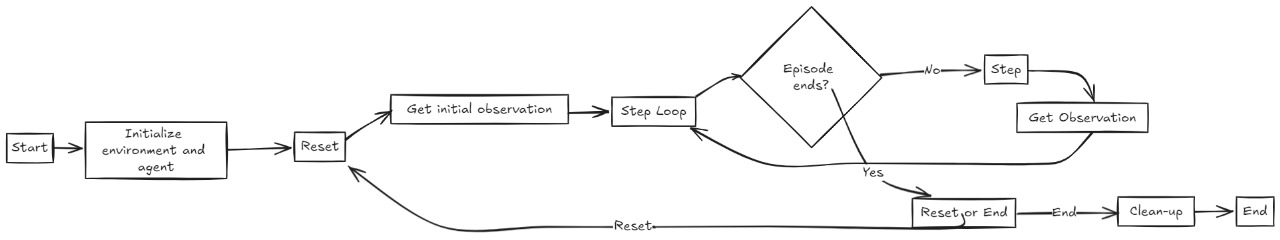
In order to allow the model to interact with the environment that we created, we must create a class that inherits from the gym.env class, this class must contain in it the reset method and the step method. Some extra methods are close for clean-up and render for displaying the environment.

The class contains the reset function, that resets the game environment whenever an episode ends

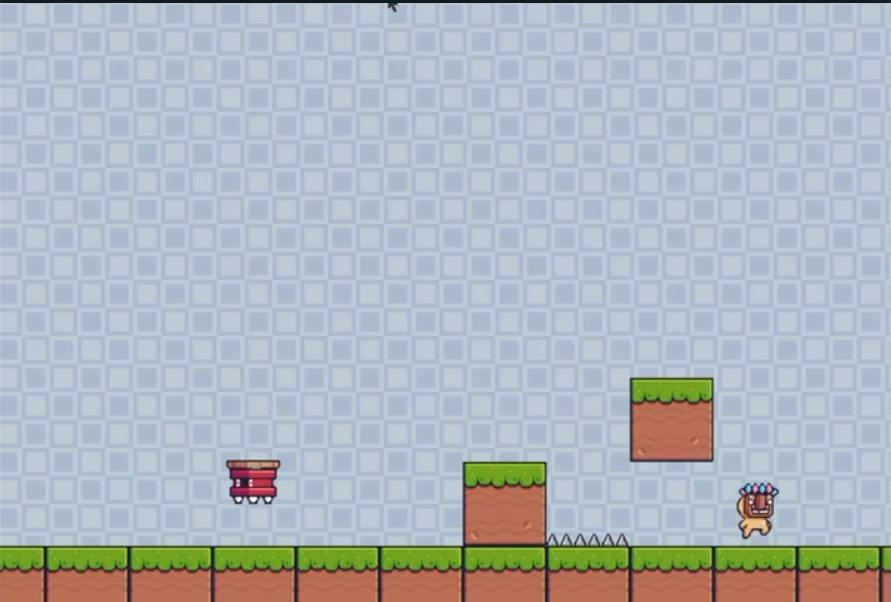
There is then the step function, which carries out the action in the environment that the model chose.

There is the get\_obs function that is used to obtain an instance of the environment for the model to train on. This is the most important function when it comes to training

A diagram displaying the function calls will help in the understanding:



**OUR FINAL GAME**

****

**Drive link to watch a video of our game:**

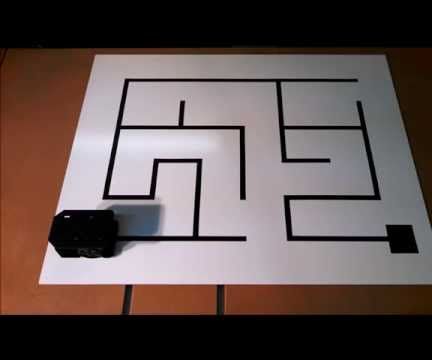
<https://drive.google.com/file/d/1JlHrDJPEddvRV9pRT5sGlIE_m39W_Qn5/view?usp=sharing>

**6. Future Prospects**

In the future, we plan to expand our project by integrating a physical robot that will utilize our reinforcement learning code to navigate a 2D maze. This exciting development will allow us to apply the theoretical foundations of reinforcement learning in a practical, real-world setting. We will experiment with both Q-learning and Deep Q-Networks (DQN) to determine which approach yields the best performance based on pixel data from the maze.

The robot will be equipped with sensors to perceive its environment and gather information about the maze's layout, including obstacles and pathways. By feeding this information into our RL algorithms, the robot will learn to make decisions about movement and strategy through trial and error, ultimately improving its ability to navigate the maze efficiently.

We anticipate that this integration will not only enhance the learning capabilities of the robot but also provide valuable insights into the effectiveness of different reinforcement learning methods in dynamic environments. This advancement could pave the way for future applications in robotics, where intelligent agents can autonomously explore and interact with complex environments.



**7. References and Resources**

1.] Reinforcement Learning Course by David Silver

<https://www.youtube.com/playlist?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ>

2.] Neural Networks Course from Coursera

<https://www.coursera.org/account/accomplishments/verify/S2C8DYZN8C54?utm_source=ios&utm_medium=certificate&utm_content=cert_image&utm_campaign=sharing_cta&utm_product=course>

3.] Convolutional Neural Networks from Coursera

<https://www.v7labs.com/blog/convolutional-neural-networks-guide>

**Some RL Papers that we referred to:**

1.] Paper 1:

<https://arxiv.org/abs/2304.00026>

2.] Paper 2:

<https://ar5iv.labs.arxiv.org/html/1708.05866>