# Training and Competing Against a Reinforcement Learning Agent in a 2-Dimensional Grid-World

Natalie Huante
*Fowler School of Engineering*
*Chapman University*
Orange, California, USA
huante@chapman.edu

*Max Starreveld*
*Fowler School of Engineering*
*Chapman University*
Orange, California, USA
starreveld@chapman.edu

***Abstract*—Artificial Intelligence algorithms can be rather complex for non-engineers to understand, especially reinforcement learning. This paper presents the development of a 2d game using unity game engine to allow users to simulate and visualize the learning of an agent using two common RL algorithms: Q-Learning and SARSA.**

## I. Introduction

Reinforcement Learning (RL) has experienced a rise in popularity and use when it comes to agents operating long term in a consistent environment, enabling them to learn more and more optimal strategies in these environments over time. In this project, we created an interactive 2D grid-world game using the Unity game engine to showcase the capabilities of RL through direct user interaction. The system allows players to challenge an agent trained to capture them, and also provides a visualized training mode where the training process can be observed in real-time.

## II. Methodology

### A. Environment Design

The game takes place in a discrete, 2D environment in the shape of a square board. The Agent (henceforth referred to as the chaser) and the player always start at random positions on the board, and they will never randomly start from the same position. There are boundary walls on the outsides of the arena so that neither may move off the board, and obstacles/dead spots can be easily added in for a future redesign. The player and chaser both take turns moving one step to any of the 4 non-diagonal neighboring locations. In training visualization, the players position is locked and only the chaser will be allowed to move in the effort to make it learn the best path at each tile.

### B. Game Modes

There are two modes of use for the system, with different purposes. Firstly, there is the Training Visualization mode, referred to as "The Training Grounds" in-game. This mode does not involve player movement input, it is more of a simulation. The chaser is placed at a random position on the grid at the beginning of each episode, and at the early stages of training has no learned experience whatsoever. The player is also placed at a randomly selected tile on the grid and remains static for the entirety of the episode. The chaser will then follow their current policy to determine which neighboring tile is the best. As they do so and observe the rewards they receive at each step, the chaser will update its policy to reflect its training experience. Each episode ends either when the player is caught by the chaser or a maximum number of steps has been reached. Q-Tables are updated in full time so improvement begins to be visible as the chaser learns.

Alternatively, there is the playing mode, referred to as "The Playing Grounds" in-game, which is more game-style. The player is controlled by a user in an attempt to escape an already trained agent or a completely un-trained agent. The chaser's policy either holds no information as to how valuable each action or tile might be or is pre-loaded from a previously saved Q-Table (in the form of a JSON file). The sole purpose of the player is to evade capture as long as possible. The intention is to display the prowess of a fully trained chaser, in contrast to the seemingly random movements of a fresh untrained chaser.

### C. Algorithms

We implemented two different reinforcement learning algorithms:

*1) Q-Learning, an off policy algorithm in which the agent updates its Q-values using the maximum estimated future reward:* $Q(s,a) := Q(s,a) + \alpha \left[ r + \gamma \max_{\{a'\}Q(s',a')} - Q(s,a) \right]$

*2) SARSA, an on-policy method using the agents actual next action to update policy:* $Q(s,a) := Q(s,a) + \alpha \left[ r + \gamma\, Q(s',a') - Q(s,a) \right]$

### D. State and Action Space

The state space consists of all possible tuples of player and chaser position. In each step, the action space consists of the 4

Identify applicable funding agency here. If none, delete this text box.

cardinal directions: up, down, left and right. The reward space is defined as follows:

- A reward of positive 10 is given if the player is caught.
- A reward of positive 0.05 is given if the distance to player decreases.
- A reward of negative 0.05 is given when the distance to the player increases.
- A reward of negative 0.2 is given when the chaser attempts to make an invalid move.

## III. Experiments

This section describes the experimental setup used to evaluate the learning behavior and performance of the Q-learning and SARSA agents in the 2D grid-world environment. Both agents were trained separately in identical conditions to facilitate a fair comparison.

### *A. Environment Configuration*

The game environment was implemented as a 5×5 grid in Unity. Each cell represents a discrete position that can be occupied by either the player or the agent. The environment is deterministic, with no stochastic transitions. The chaser and the player are both assigned a random position at the beginning of each episode. The player will stay stationary in training mode, or can be controlled by a user in playing mode. The environment includes:

- **Boundary constraints** to prevent out-of-bounds movement.
- **Obstacle-free layout** to isolate learning behavior from environmental complexity (future work could introduce static or dynamic obstacles).

Each agent operates under a fixed episode length of 20 time steps. Episodes terminate early if the agent successfully captures the player by occupying the same cell.

### *B. Training Parameters*

The following hyperparameters were used for both Q learning and SARSA agents unless otherwise noted:

- **Learning rate (α)**: 0.1
- **Discount factor (γ)**: 0.9
- **Exploration strategy (ε)**: 0.2
- **Training episodes**: 50, 100, 500, 800
- **State representation**: {chaser x, chaser y, player x, player y}
- **Action space**: {Up, Down, Left, Right}

These values represent the default values of the hyperparameters as well as the range in which the player may update them to be. In Training Mode, there is an interactive UI panel for the player to experiment with different configurations of these hyperparameters. Therefore, we provide default values, but there is some flexibility as the purpose of this mode is to allow the user to learn through interaction.

Agents maintained a Q-table of shape 2,500. The state space was bounded to a maximum distance of ±4 cells in either direction due to the 5×5 grid constraints. Given the dimensions of the state space, both the player and the chaser have 25 possible tiles to occupy at any timestep, each of which has 4 actions available. Therefore, the Q-table that represents each state-action pair would be 25 x 25 x 4 = 2500 Q-values.

### *C. Evaluation Protocol*

There is no real evaluation protocol for playing mode; playing mode is meant to display the abilities of a learned agent vs an untrained agent. For the training visualization mode, average reward over all episodes, total reward for the current episode, and the player catch rate are all displayed dynamically in real time, so you can see the current progress.

## IV. Results and Analysis

Although this project is meant to be a learning tool, we can see the results of how the trained agents and untrained agents performed. To evaluate the effectiveness of the SARSA and Q-learning agents, both were trained under identical hyperparameter settings for 1000 episodes. These hyperparameter values are as followed: Alpha = 0.10, Gamma = 0.90, Epsilon = 0.20. Key metrics, such as average rewards per episode and the cumulative catch rate, were tracked and visualized using the real-time UI panel in “The Training Grounds”.

After approximately 300 episodes, both agents began to demonstrate noticeable improvements in catching the player more frequently. SARSA displayed a smoother learning curve, with catch rate gradually improving and stabilizing earlier than Q-learning. On the other hand, Q-learning was more volatile in the earlier training episodes but ultimately converging to a slightly more aggressive policy by around 800 episodes. In most trials, SARSA reached a stable average rewards earlier, around episode 150, whereas Q-learning continued to fluctuate before stabilizing off closer to episode 700.

In one trial, the SARSA agent had a catch rate of 97% and average reward across episodes of 9.72, whereas the Q-learning agent had a catch rate of 20% and average reward across episodes of -1.00. However, it is important to note that these specs are from the training process only. SARSA appeared to learn quicker, so it learned to catch the player much earlier on, resulting in the majority of its training episodes to end in the player caught state. Q-learning, however, takes more episodes for it to learn a more optimal policy, resulting in the majority of

its training episodes to not finish in the player caught state. These metrics are not from testing, but from the training process only. Additional testing is needed to compare the optimality of each final policy.

When comparing the trained agents in "The Playing Grounds", SARSA and Q-learning both performed well, following a generally direct path to the player given both of their current positions. It is clear the benefit that RL had on the agent's ability to make an intelligent decision on the board. Overall, both algorithms successfully learned policies that outperformed random movement. Against either of the trained agents, it is fairly difficult to evade capture for long, since they were able to learn pretty optimal policies. Against untrained agents, the player can evade capture indefinitely, since the agents do not make generally intelligent movements and move randomly across the board.

## CONCLUSION

This project successfully demonstrates how reinforcement learning agents can be trained and deployed in an interactive 2D grid environment using Unity. By simulating and training agents with both SARSA and Q-learning, we offer an accessible and engaging way for users to observe the learning process and compare different RL approaches.

The training visualization mode provides valuable insight into how policies are developed over time and how hyperparameters like α, γ, and ε affect learning outcomes. The playing mode complements this by turning the simulation into a game where users can experience first-hand how a trained agent differs from a randomly moving one.

Work could extend the environment with stochastic transitions, dynamic obstacles, or multi-agent setups, or even more complex worlds.

The full source code and instructions for running the simulation and playing the game can be found at our GitHub repository:
https://github.com/nhuante/TheRlCastle_RLSimulatorGame

## FUTURE ADDITIONS

To further expand the learning potential and gameplay value of this project, several enhancements and research directions could be explored. We include a few categories of additions we would like to explore given more time and resources:

**Additional RL Algorithms**

- *Deep Q-Networks (DQN):* Using neural networks instead of a Q-table to generalize to larger or continuous state spaces
- *Monte Carlo Methods:* Exploring episodic vs step-wise value updates

**Enhanced Game Mechanics**

- *Win/Loss Conditions:* Victory condition for the player (e.g. reaching a goal tile) to make playing mode more competitive and engaging
- *Collectibles/Coin System:* Scoring system for the player to interact with as they evade the agent, rewarding long-term evasion
- *Multiple Agents:* Introducing chasers of different training backgrounds to compare multiple RL approaches at once

**Environmental Complexity**

- *Larger Grids:* Extended grid (e.g. 10x10)
- *Obstacles/Walls:* Environments with barriers, hallways, or mazes to force more strategic behavior
- *Hazard Tiles:* Cells that penalize either the player or agent for stepping on them, encouraging smarter pathing

**Educational and Accessibility Features**

- *Step-by-Step Learning Mode:* Slow down training with explanations for what the agent is doing and why
- *Glossary and Tooltips:* In-game definitions and references for RL terms

.

## REFERENCES


[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, May 1992.

[3] T. Sanders and M. Fitch, "Teaching reinforcement learning through interactive simulation," *Journal of Computing Sciences in Colleges*, vol. 33, no. 6, pp. 21–28, Jun. 2018.

[4] Unity Technologies, *Unity Manual – Reinforcement Learning and Game AI*. [Online]. Available: https://docs.unity3d.com

[5] G. Brockman et al., "OpenAI Gym," *arXiv preprint*, arXiv:1606.01540, 2016. [Online]. Available: https://arxiv.org/abs/1606.01540

[6] Fowler School of Engineering, CPSC 531 Advanced Artificial Intelligence, "Course materials on Reinforcement Learning," Chapman University, 2025. [Class excercises, projects, and lectures].