# Report

## Pursuit AI - Nostalgia of Pakdam Pakdai

DS 251: Artificial Intelligence
Group-2

---

## Introduction:

Reinforcement Learning (RL) is a powerful paradigm in artificial intelligence where agents learn to make decisions by interacting with an environment. In this project, we embark on a journey to implement RL in a game environment, bridging the gap between theory and application. The game, known as Pursuit AI, draws inspiration from the childhood game "Pakdam Pakdai" and serves as a platform for learning and applying RL techniques.

The game features two types of agents: Catchers and Runners, engaged in a pursuit scenario. The decision-making mechanism for these agents is driven by the epsilon-greedy policy, a crucial concept in RL.

Noteworthy is the utilization of pre-trained Q tables to initialize strategic policies for the agents. These Q tables serve as repositories of learned knowledge, providing a head start for our agents to navigate the game environment. Throughout the gameplay, these Q tables are dynamically updated to adapt and refine the agents' decision-making strategies.

---

## Methodology:

### 1) Game Overview:

**Agents:**

- **Catchers**: These are the pursuers, equipped with RL algorithms to optimize their pursuit strategies.
- **Runners**: The elusive targets that use RL to intelligently evade from getting caught by the Catcher.

**Rules:**

- Two teams, each comprising two catchers and six runners.
- Catchers aim to capture runners by strategically moving in the game environment through the obstacles in the arenat.

- Catchers can decide whether to freeze a runner within a specified range, introducing a strategic element.
- Special powers and dynamic interactions add depth to the game.

**Goal:**

- For catchers, the goal is to capture runners efficiently.
- For runners, the goal is to evade capture and strategically navigate the environment.

## 2) Role of RL:

In Pursuit AI, RL plays a crucial role in shaping the behavior of both catchers and runners. The fundamental elements of RL manifest in the game:

**Policy:** Catchers and runners follow extensive policies dictated by RL algorithms to make decisions based on their observations and experiences.
**Reward Function:** The game's reward system reinforces desirable behaviors. Catchers receive rewards for capturing runners, while runners are rewarded for evading capture.
**Value Function**: RL algorithms estimate the value of taking certain actions in specific states, guiding agents toward optimal decision-making.
**Model:** The game's dynamics, state transitions, and reward structures serve as the model that RL agents learn from.

## 3) Mathematical Aspects

### a) Q-value Update Formula (Catcher):

*Q(s, a)=(1 − a) · Q(s, a) + a· [r+y· max, Q(s', a)]*
where:

- Q(s, a) is the Q-value for state-action pair (s, a).
- a is the learning rate (here, 0.1).
- r is the reward obtained.
- y is the discount factor (here, 0.9).
- s' is the next state.
- a is the chosen action

### b) Q-value Update Formula (Runner):

*Q(s, a) = (1 - a) · Q(s, a) + a· [r+y· max, Q(s', a)]*

- where the symbols have the same meaning as above.

### c) Epsilon Greedy Action Selection :

The expression,

$$\pi(a/s) = \{ random\ action\ with\ probability\ \epsilon \\ argmax(x)\ Q(s,a)\ with\ probability\ 1 - \epsilon \}$$

represents the policy π for selecting actions in the epsilon-greedy strategy, a common approach in reinforcement learning.

**D) Q-Table:**

- Dimensions: (state, actions)
- State_space for catchers is defined as list of the tuples containing the difference in coordinates of the catcher and the closest runner to it
- State_space for runners is defined as list of the tuples containing the difference in coordinates of the runner and the closest catcher to it; and also the distance of closest obstacle.

---

# Coding Aspects:

- Designed a custom-built environment using OpenAI Gym and Pygame for visualization.
- 4 Files => Catch_init.py, Run_init.py, Train.py and Visualize.py

**1. Defined Classes:**

**Obstacle Class:**

- Represents an obstacle in the game.
- Initialized with a given position.

**CatcherAgent Class:**

- Represents a catcher agent in the game.
- Initialized with an initial position and Q-learning parameters.
- Q-table is loaded from a file or initialized if not available.
- Methods for getting Q-value, choosing an action, updating Q-value, moving, and using a special power.

**RunnerAgent Class:**

- Represents a runner agent in the game.
- Similar structure to CatcherAgent with additional methods for freezing, unfreezing, and updating freeze duration.

**TagGameEnv Class:**

- Inherits from gym.env, making it compatible with OpenAI Gym environments.
- Represents the overall game environment.
- Initializes catchers, runners, obstacles, and defines observation and action spaces.
- Methods for resetting the environment, taking a step, calculating rewards, checking termination conditions, calculating distances, and obtaining the current observation.
- Q-tables can be saved (as per the requirement).

## 2. Initialization:

- Initialize the game environment with the specified numbers of catchers, runners, and obstacles.
- Run the files in sequence to ensure error-free execution.

### 3. Training Loop:

For each episode:

- Since the training is cumbersome, it is segmented into various parts.
    - Catch_init.py file is to pretrain the catchers (when runners are moving randomly)
    - Run_init.py is to pretrain the runners (when catchers are moving randomly).
    - These two steps ensure that the q-tables of catchers and runners are not initialized with zeroes.
    - Train.py is for the main training (when both runners and catchers are using Q-learning algorithm)
- Environment is reset, and the initial state is regained.
- The RL agents (catchers and runners) choose actions based on their policies.
- Game environment is updated, and Q-learning is applied for both catchers and runners.
- Episode progress is printed.

### 4. Testing Loop:

- After training episodes, the environment is reset, and random actions are taken for catchers and runners to observe their behavior.
- Visualize.py is for the visualization of trained Catchers and Runners. It's the only file where the updates (training updates) are shown visually.

---

## Conclusion:

Through the implementation of Pursuit AI, we dwell in to the world of RL, witnessing

- The application of Q-learning in optimizing the strategies of both catchers and runners.
- The game project provided hands-on experience in understanding RL concepts such as exploration-exploitation trade-offs, state representation, and dynamic decision-making.
- By achieving a balance between nostalgic gameplay and cutting-edge technology, this project highlights the potential of gamified environments in facilitating RL learning.
- The journey not only resulted in a functional Pursuit AI game but also deepened our understanding of RL algorithms and their practical applications.
- For choosing appropriate state-space for catcher and runner, we explored and implemented various different configurations and finally, came up with the one implemented in the code (Relative position of closest entity of opposite team).