Advanced AI

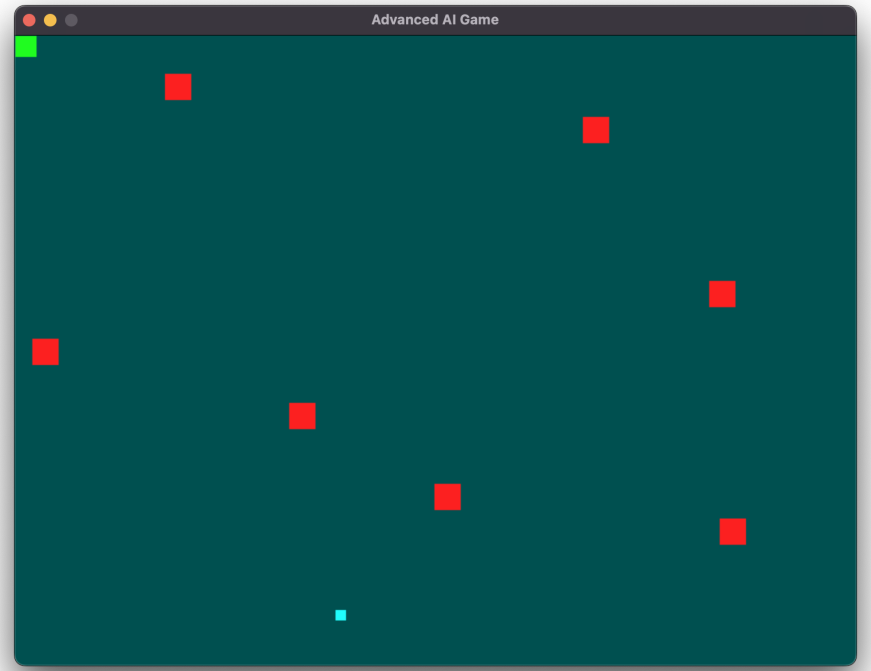
Assignment 4 Report

Mukul Shingwani (B20AI023) & Jaimin Sanjay Gajjar (B20AI014)

💡 **Problem statement**

- You need to train an AI to control that can play the game given in this assignment. The game is
composed of :-
    - A **goal state** : A rectangular location in the game screen.
    - **Multiple enemies** : Rectangular blocks that move with a constant speed and bounce around the
game screen.
    - **Player entity** : A rectangular block that the player/AI must control in the 4 cardinal directions.
- The player must reach the goal state (that is, the player rectangle and the goal state rectangle must
intersect) while avoiding the enemies.
- Every time the player reaches the goal state, or gets attacked, the goal state is reset, and the game
continues from that point.

- **Goal state** is depicted with a blue square 🟦
- **Player** is depicted with a green square 🟩
- **enemies** are depicted as red squares 🟥
- **GAME_SEED** = `22019014`, since my roll number is B20AI014



Part 1

1. **Agent**

    a. In this game, the green square 🟩 represents the `player` object, which acts as the agent. The primary objective of the agent is to reach the goal state by learning from its interactions with the environment. The agent achieves this by maximizing its rewards, which serve as a form of guidance. This learning process occurs recursively, allowing the agent to continually improve its performance over time.

2. **Environment**

    a. The overall game that opens up is our environment in this case. This environment is responsible for defining all the rules, including the `state space, action space, and reward function`. In our `current setup`, the game simulation serves as our environment, determining the player's movements and limitations, as well as the actions and locations of its enemies and the goal state. Various game variables, such as location speed and acceleration also fall under the purview of the environment.

    b. We have a `fully observable environment` in this case

3. **Actions**

   a. An action refers to a decision or choice made by the agent that results in a change of state in the environment.

   b. `Left, Right, Up or Down` are the possible actions in this game (⬅️, ➡️, ⬆️, ⬇️)

   c. The set of all possible actions that the agent can take in a given state is called the action space. The agent selects actions based on its policy, which is a set of rules or decision-making criteria that it uses to select actions. The ultimate objective of the agent is to choose actions that maximize its cumulative reward over time.

4. **Observations**

   a. The `effect of an action` taken by an agent in the environment refers to the observations for that particular game.

5. **Distance Term**

   a. We calculates the Euclidean distance between the center points of two entities in a game state: **the player-controlled agent and the goal entity**. We first retrieve the coordinates of the center points of the two entities and their respective widths and heights.

   b. We then calculates the squared horizontal and vertical distances between the center points of the two entities using the retrieved information.

6. **Reward**

   Reward consists of 3 factors

   1. avoid_enemy : After many re-runs on different values, we found that -100 was coming to be a good amount of penalty for hitting an enemy.

   2. goal_reward : When we hit the goal, we get tis reward.

      $\frac{1000}{distance\_term}$

   3. idle_penalty : When the distance remains constant from the goal, we impose this penalty. Here, -1 is a constant factor decided after many re-runs.

      $$-3 + e^{-2} * distance\_term$$

Part 2 (In code)

`Deep Q-Learning (DQL`) is a machine learning algorithm used to train an agent to make decisions in an environment based on a reward signal. It is an extension of Q-Learning, a classic reinforcement learning algorithm, that incorporates deep neural networks to learn a `Q-value function` that estimates the optimal action-value function. The Q-value function represents the expected discounted future reward for each action in a given state, and the optimal action-value function provides the best possible value achievable by the agent in any given state.

The DQL algorithm involves training a deep neural network to approximate the Q-value function. The neural network takes the current state of the environment as input and produces Q-values for all possible actions. The agent selects actions with the highest Q-value, which is updated using the `Bellman equation` to incorporate the observed reward and the predicted Q-value of the next state. The network is trained through backpropagation to minimize the difference between the predicted Q-values and the observed Q-values obtained through interactions with the environment.

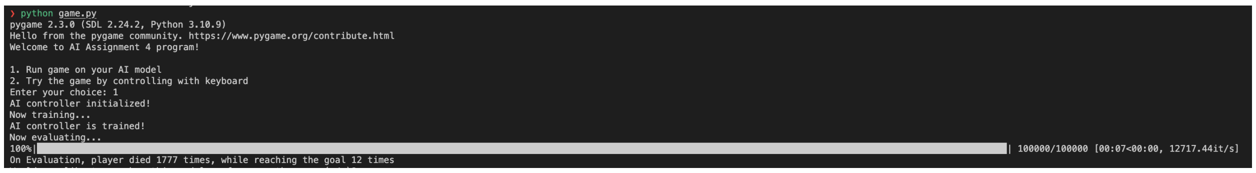
```
class QNetwork(nn.Module):
    def _init_(self, state_dims, action_dims):
        super()._init_()
        self.fc1 = nn.Linear(state_dims, 64)
        self.fc2 = nn.Linear(64, 128)
        self.fc3 = nn.Linear(128, 256)
        self.fc4 = nn.Linear(256, 128)
        self.fc5 = nn.Linear(128, 64)
        self.fc6 = nn.Linear(64, action_dims)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = torch.relu(self.fc5(x))
        x = self.fc6(x)
        return x
```

Part 3

on running the `game.py` file, the game played through the AI gives results like this

```
On Evaluation, player died 1777 times, while reaching the goal 12 times
```



```
❯ python game.py
pygame 2.3.0 (SDL 2.24.2, Python 3.10.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
Welcome to AI Assignment 4 program!

1. Run game on your AI model
2. Try the game by controlling with keyboard
Enter your choice: 1
AI controller initialized!
Now training...
AI controller is trained!
Now evaluating...
100%|████████████████████████████████████████████████████████████████████████████████| 100000/100000 [00:07<00:00, 12717.44it/s]
On Evaluation, player died 1777 times, while reaching the goal 12 times
```

- **Game seed:** We observed, no major changes even when game seed was not constant
- **Game width and Height:** This shouldn't impact the performance since our game is not considering these variables while training or evaluation. It is independent of the game arena.
- **Goal_size:** This should help in increasing the performance of our model, since it has already learned how to reach the goal despite its smaller size, when its contact area would be increased it would be easier for the player to hit the goal.
- **Enemy_count:** When this would be increased, it should result in a decreased performance, since the model was trained with fewer number of adversaries and on increasing that, it would get slightly dominated, since it has not learnt it yet. on the contrary, if the count is decreased it should perform better since it has already learnt how to perform well with larger number of enemies.
- **Game_friction:** This can affect the agent to its benefit and loss both. Increasing the game friction will reduce the moving capabilities of the agent but that would eventually be applicable to all, so it might help agent to perform well. If friction is reduced, moving capabilities would be increased which may result is more collisions and make it hard for agent to avoid the enemies and reach the goal.
- **FPS:** If this is increased, it should result in a better performance since there could be more frequent and real time updates for the RL algorithm to take into account and learn from accordingly.