Assignment-4: Deep RL

(CSL 7580 Advanced AI)

Report

Assignment	**04**
Roll No.	**B20CS018**
Name	**Harshita Gupta**

INTRODUCTION

The problem at hand involves a simple game where the player is tasked with reaching a rectangular goal state while avoiding enemies. The player entity can move in four cardinal directions, and the enemies move with a constant speed and bounce around the game screen. Every time the player reaches the goal state or gets attacked by an enemy, the game restarts from that point. The game can be controlled either by keyboard input or a custom AI controller.

To run the game, the pygame package needs to be installed, and a valid integer must be assigned to the constant GAME_SEED in the game_constants.py file. The value of the seed should be assigned appropriately to ensure the reproducibility of the game. The game offers a simple yet engaging environment to test and develop AI control strategies, making it an ideal problem for learning and experimentation.

GAME_SEED = 220319018(B20CS018)

NOTE:

The provided code was not running and giving the following error in game_constants.py file-

```
harshita@harshita-inspiron-15:~/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code$ python3 game.py
pygame 2.3.0 (SDL 2.24.2, Python 3.8.10)
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: 2
Traceback (most recent call last):
  File "game.py", line 97, in <module>
    my_game.UpdateFrame()
  File "game.py", line 34, in UpdateFrame
    self.__innerState.Update(action)
  File "/home/harshita/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code/common/game_state.py", line 95, in Update
    self.PlayerEntity.Move(input_vector, self.Boundary)
  File "/home/harshita/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code/common/game_constants.py", line 127, in Move
    (self.entity, self.velocity) = Move(self.entity, self.velocity, boundary)
  File "/home/harshita/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code/common/game_constants.py", line 71, in Move
    def Clamp(rv:int, bv:int, rs:int, bs:int, vv:float)->tuple[int, float]:
TypeError: 'type' object is not subscriptable
```

So, I did the following change in the code-

```python
from typing import Tuple
GAME_WIDTH = 800
GAME_HEIGHT = 600
PLAYER_SIZE = 20

GAME_FRICTION = 0.05
```

```python
# Moves a rectangle with constant speed, and handles bounce logic with a given boundary
def Move(rect:GameRectangle, v:Vector, boundary: GameRectangle) -> None:
    # Move with constant velocity
    rect.x = rect.x + v.x
    rect.y = rect.y + v.y

    def Clamp(rv:int, bv:int, rs:int, bs:int, vv:float)->Tuple[int, float]:
        if rv <= bv or rv + rs >= bv + bs:
            vv = -vv
            if rv <= bv:
                rv = bv + 1
            else:
                rv = bv + bs - rs - 1
        return (rv, vv)
```

PART 1

Description of Game

Agent: The agent in this game is an artificial intelligence that controls the player entity in the 4 cardinal directions. Its goal is to reach the goal state while avoiding enemies.

Environment: The environment is the game screen, which contains the player entity, the goal state, and the enemies. The player block and the goal state are both rectangular blocks within the game screen. Multiple enemies, which are also rectangular blocks, move with a constant speed and bounce around the game screen.

Actions: The agent can take one of the five actions: No_action, Up, Down, Left, Right. These actions move the player entity in the corresponding direction.

Observations: The agent can observe the following things in the environment: Nothing (when there is no change in the environment), Enemy_Attacked (when the player entity collides with an enemy), and Reached_Goal (when the player entity reaches the goal state).

Reward Function: A suitable reward function for this game would be one that rewards the AI when it reaches the goal state and penalizes it when it gets attacked by an enemy. We can define the reward function as follows:

- If the player entity reaches the goal state, the agent gets a positive reward.

- If the player entity collides with an enemy, the agent gets a negative reward.
- Otherwise, the agent gets a zero reward.

This reward function is valid because it incentivizes the agent to reach the goal state while avoiding enemies. The positive reward for reaching the goal state encourages the agent to move toward the goal state. The negative reward for colliding with an enemy encourages the agent to avoid the enemies. The zero rewards for other observations indicate that the agent should keep exploring the environment until it reaches the goal state or collides with an enemy.

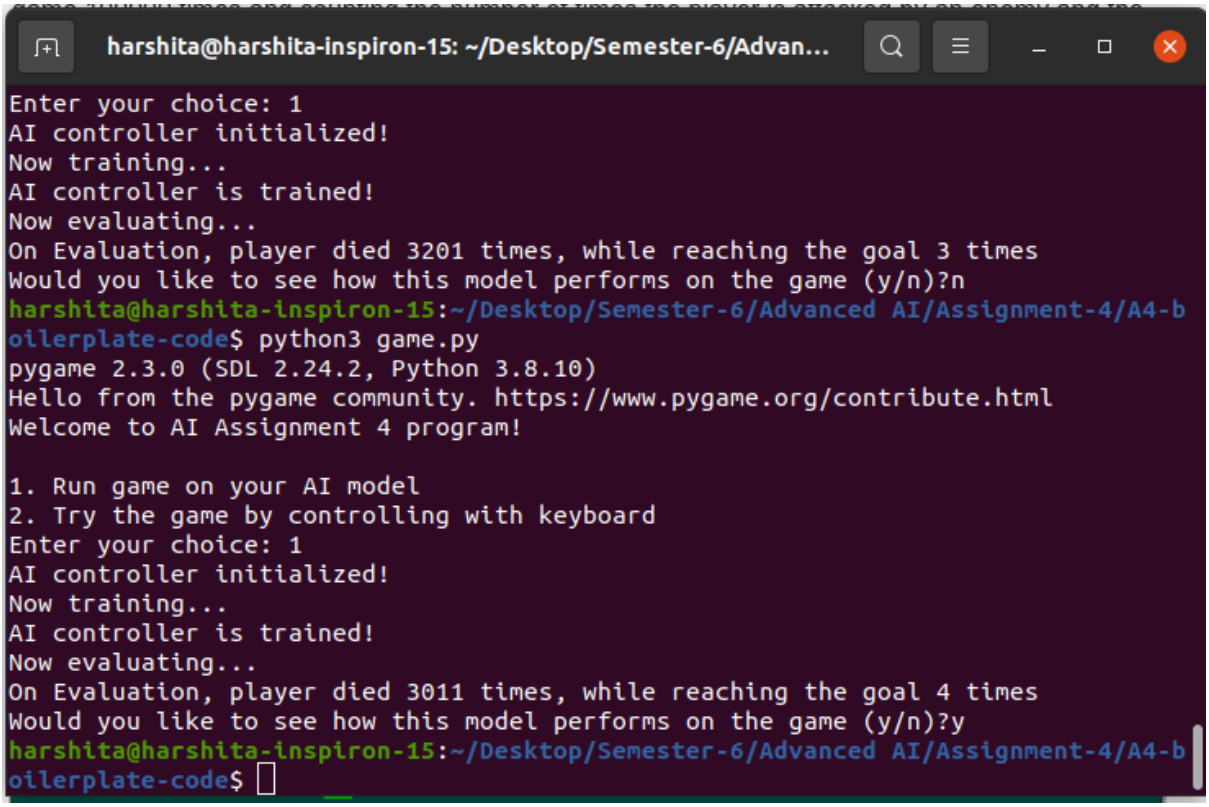
PART 2

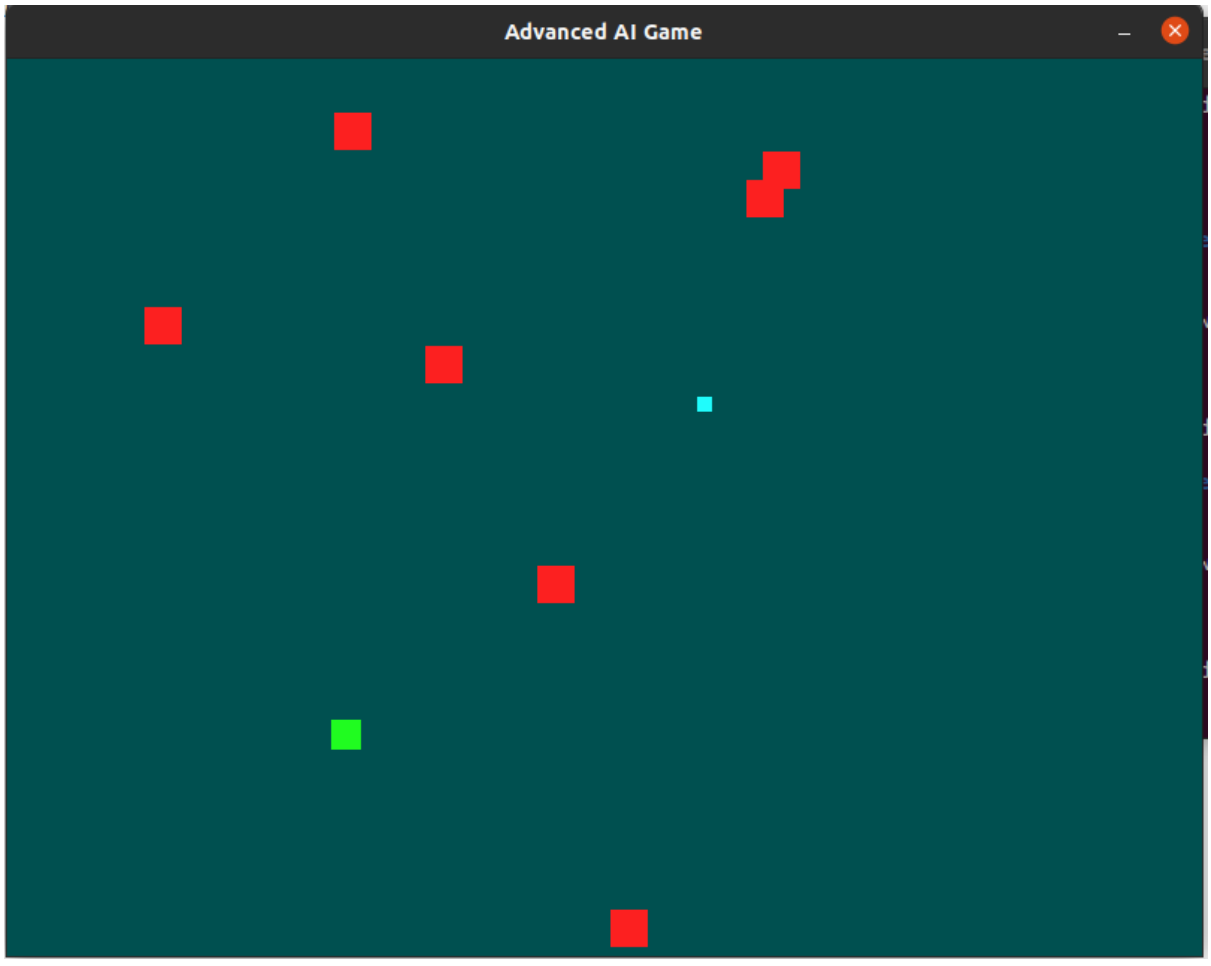
The code implements an AIController class that uses a Q-learning algorithm to learn how to navigate a game environment. The AIController has a Q-table that stores the expected rewards for each state-action pair. The Q-learning algorithm works by repeatedly playing the game and updating the Q-table based on the observed rewards.

The AIController has a GetAction method that takes in the current state of the game and returns an action to take. If the AIController's epsilon value (which determines the probability of taking a random action) is greater than a randomly generated number, the AIController takes a random action. Otherwise, the AIController takes the action with the highest expected reward based on the Q-table.

The AIController has a TrainModel method that trains the Q-table. The training consists of playing the game for a specified number of epochs and updating the Q-table based on the observed rewards. The Q-table is saved to a file at the end of the training.

The AIController has an EvaluateModel method that evaluates the trained Q-table by playing the game 1000 times and counting the number of times the player is attacked by an enemy and the number of times the player reaches the goal.


harshita@harshita-inspiron-15: ~/Desktop/Semester-6/Advan...
Enter your choice: 1
AI controller initialized!
Now training...
AI controller is trained!
Now evaluating...
On Evaluation, player died 3201 times, while reaching the goal 3 times
Would you like to see how this model performs on the game (y/n)?n
harshita@harshita-inspiron-15:~/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code$ python3 game.py
pygame 2.3.0 (SDL 2.24.2, Python 3.8.10)
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...
On Evaluation, player died 3011 times, while reaching the goal 4 times
Would you like to see how this model performs on the game (y/n)?y
harshita@harshita-inspiron-15:~/Desktop/Semester-6/Advanced AI/Assignment-4/A4-boilerplate-code$


Advanced AI Game

PART 3

The model may not perform well after changing the game constants. This is because the state representation generated by `to_array()` method depends on the position of the player, goal and enemies, which are all affected by the game constants. If the game constants change, the state representation will change, and the Q-table will not be useful for selecting actions in the new state space.

Therefore, the model needs to be retrained with the new game constants to perform well in the modified game.

1. If GAME_SEED is selected randomly, rather than keeping it a constant number:
   The performance of the trained model may vary when the game seed is selected randomly because it will result in a different game environment. The trained model may not perform as well on a new environment because the Q-values for the new state-action pairs would not have been learned. However, the model should still be able to perform reasonably well since it has learned the optimal actions for similar state-action pairs.
2. If the dimensions of the game, i.e GAME_WIDTH and GAME_HEIGHT are changed:
   If the dimensions of the game are changed, it will affect the state space of the game, resulting in new state-action pairs. This means that the trained model will not be able to generalize well to the new state space since it has only learned the optimal actions for the previous state space. Therefore, the performance of the model will likely decrease when the dimensions of the game are changed.
3. If the variable GOAL_SIZE is changed:
   The size of the goal affects the state space of the game, but the impact may not be as significant as changing the dimensions of the game. If the goal size is increased, the model may perform better since it will be easier to reach the goal. Conversely, if the goal size is decreased, the model may perform worse since it will be harder to reach the goal.
4. If the variable ENEMY_COUNT is changed:
   Changing the number of enemies in the game will affect the state space of the game, resulting in new state-action pairs. Therefore, the trained model will not be able to generalize well to the new state space since it has only learned the optimal actions for the previous state space. Therefore, the performance of the model will likely decrease when the number of enemies in the game is changed.

5. If the variable GAME_FRICTION is changed:
   Changing the friction of the game will affect the movement of the player and enemies in the game. Therefore, the Q-values learned by the model may not be optimal for the new friction value. However, the impact may not be significant, and the trained model may still perform well.
6. If the variable FPS is changed:
   Changing the FPS (frames per second) of the game will affect the speed at which the game runs. The model may still perform well since it has learned the optimal actions for the state-action pairs, but the game's faster or slower speed may affect the timing of the actions. Therefore, the model's performance may decrease if the FPS is significantly changed.