NURETTİN ERTUĞRUL GÜLMEZ

190101050

Reinforcement Learning With AI Project Report

PROJECT NAME:Dino

1. Introduction

This report is prepared on a game called Dino. The main aim of the Dino game is to control a dinosaur and make it jump over trees. The game is developed using Python and Pygame and employs Reinforcement Learning algorithms to enable an agent to learn the game. This report will cover the structure of the game's code, its working principles, and technical details.

2. Game Environment (dino_env.py)

This section explains the basic components of the Dino game and the creation of the game environment.

2.1. Libraries and Constants

```
import gym
from gym import spaces
import numpy as np
import pygame
import os

# Constants
SCREEN_WIDTH, SCREEN_HEIGHT = 800, 600
GROUND_HEIGHT = 550
```

```
GRAVITY = 2

FPS = 60

JUMP_COOLDOWN = 20
```

- gym and spaces: Used to create the game environment and define action and observation spaces.

- numpy: Used for numerical operations.

- pygame: Used to handle game graphics and user interaction.

- os: Used for file path operations.

2.2. DinoEnv Class

The DinoEnv class represents the game environment and is derived from Gym environments.

```
class DinoEnv(gym.Env):

   def __init__(self):

      super(DinoEnv, self).__init__()

      self.action_space = spaces.Discrete(2)

      self.observation_space = spaces.Box(low=np.array([0, 0, 0, -30, 0, 0]), high=np.array([SCREEN_HEIGHT, SCREEN_WIDTH, SCREEN_HEIGHT, 30, SCREEN_WIDTH, JUMP_COOLDOWN]), dtype=np.float32)


      pygame.init()

      self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))

      pygame.display.set_caption("Dino")


      current_dir = os.path.dirname(os.path.abspath(__file__))

      self.dino_image = pygame.image.load(os.path.join(current_dir, 'assets/dino.png'))

      self.tree_image = pygame.image.load(os.path.join(current_dir, 'assets/tree.png'))


      self.dino_width, self.dino_height = self.dino_image.get_size()

      self.tree_width, self.tree_height = self.tree_image.get_size()


      self.clock = pygame.time.Clock()
```

```
        self.font = pygame.font.Font(None, 36)

        self.reset()
```

- action_space: Represents two actions (jump or not jump).

- observation_space: Observation space includes the positions of the dinosaur and tree, the dinosaur's velocity, and jump cooldown.

- pygame.init(): Initializes Pygame and creates the game window.

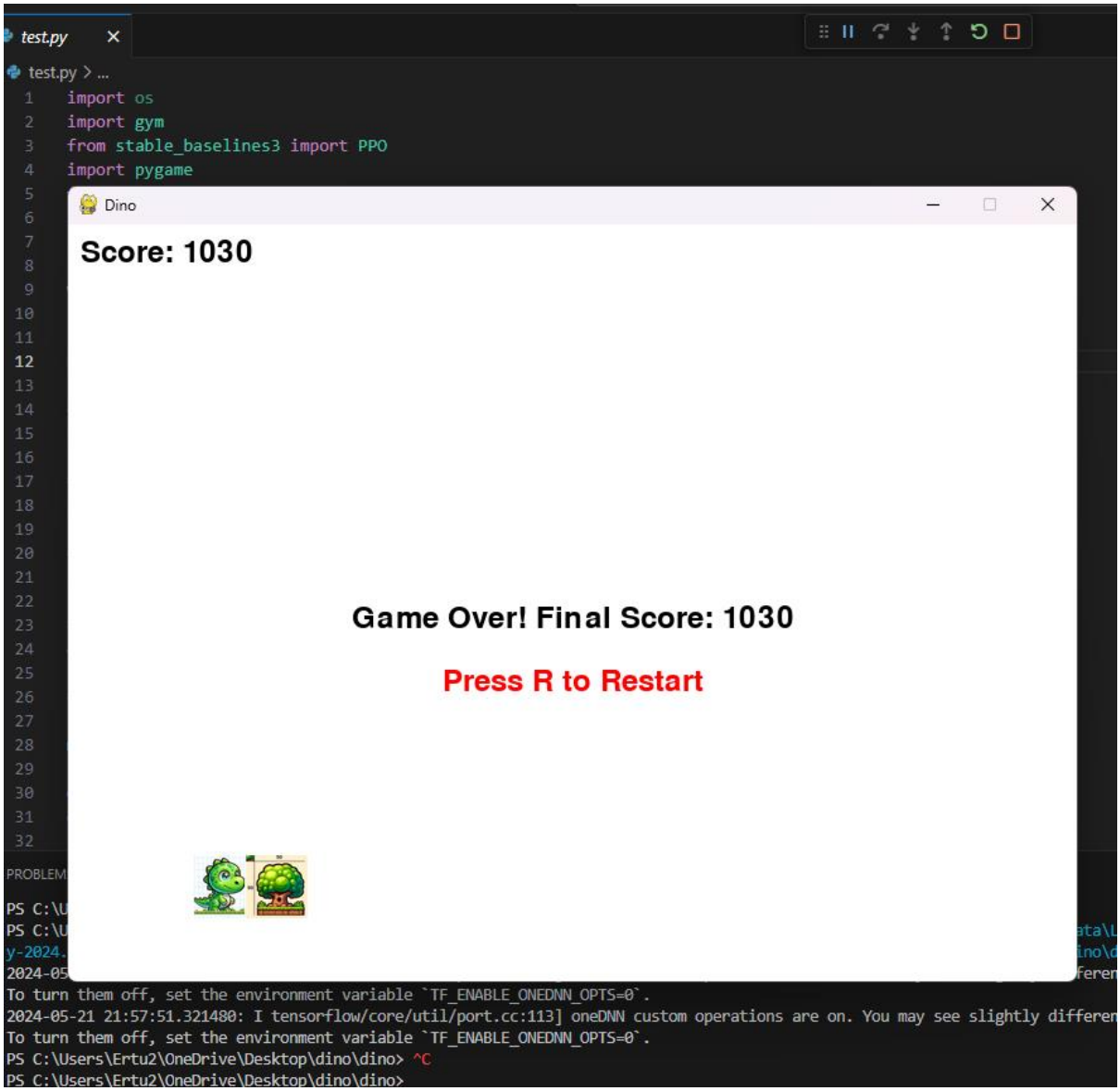
- self.dino_image and self.tree_image: Loads the images for the dinosaur and tree.

- self.reset(): Starts the game environment and sets it to the initial state.

2.3. Resetting the Game Environment (reset)

```
def reset(self):
    self.dino = pygame.Rect(100, GROUND_HEIGHT - self.dino_height, self.dino_width, self.dino_height)
    self.dino_vel_y = 0
    self.is_jumping = False
    self.jump_cooldown = 0
    self.trees = [pygame.Rect(SCREEN_WIDTH, GROUND_HEIGHT - self.tree_height, self.tree_width, self.tree_height)]
    self.score = 0
    self.game_over = False
    return self._get_obs()
```

- reset: Resets the game to the initial state, resetting the positions of the dinosaur and tree, score, and game state.



2.4. Step Function (step)

```
def step(self, action):
    reward = 0.1  # Base survival reward

    closest_tree = self.trees[0]
    distance_to_next_tree = closest_tree.x - self.dino.x

    if action == 1 and not self.is_jumping and self.jump_cooldown == 0:
        if distance_to_next_tree > 300:
            reward -= 5
```

```
    else:
        reward += 10
    self.dino_vel_y = -30
    self.is_jumping = True
    self.jump_cooldown = JUMP_COOLDOWN

if not self.game_over:
    if self.is_jumping:
        self.dino.y += self.dino_vel_y
        self.dino_vel_y += GRAVITY
        if self.dino.bottom > GROUND_HEIGHT:
            self.dino.bottom = GROUND_HEIGHT
            self.is_jumping = False
            self.dino_vel_y = 0

    for tree in self.trees:
        tree.x -= 10
        if tree.x < -self.tree_width:
            self.trees.remove(tree)
            self.trees.append(pygame.Rect(SCREEN_WIDTH, GROUND_HEIGHT - self.tree_height, self.tree_width, self.tree_height))
            self.score += 10
            reward += 10

    for tree in self.trees:
        if self.dino.collides(tree):
            self.game_over = True
            reward = -100

if self.jump_cooldown > 0:
    self.jump_cooldown -= 1

terminated = self.game_over
truncated = False
info = {}
 return self._get_obs(), reward, terminated, info
```

- step: Simulates a step in the game. It handles the jumping state of the dinosaur, movement of the trees, and collision detection. Reward and penalty mechanisms are defined here.

2.5. Observation Function (_get_obs)

```
def _get_obs(self):
    closest_tree = self.trees[0]
    distance_to_next_tree = closest_tree.x - self.dino.x
    return np.array([self.dino.y, closest_tree.x, closest_tree.y, self.dino_vel_y, distance_to_next_tree, self.jump_cooldown], dtype=np.float32)
```

- _get_obs: Creates and returns the observation space. This space includes the positions of the dinosaur and tree, the dinosaur's velocity, and jump cooldown.

2.6. Render and Close Functions

```
def render(self, mode='human'):
    self.screen.fill((255, 255, 255))
    self.screen.blit(self.dino_image, self.dino)
    for tree in self.trees:
        self.screen.blit(self.tree_image, tree)

    score_text = self.font.render(f'Score: {self.score}', True, (0, 0, 0))
    self.screen.blit(score_text, (10, 10))

    if self.game_over:
        text_game_over = self.font.render(f'Game Over! Final Score: {self.score}', True, (0, 0, 0))
        self.screen.blit(text_game_over, (SCREEN_WIDTH // 2 - text_game_over.get_width() // 2, SCREEN_HEIGHT // 2))
        text_restart = self.font.render('Press R to Restart', True, (255, 0, 0))
        self.screen.blit(text_restart, (SCREEN_WIDTH // 2 - text_restart.get_width() // 2, SCREEN_HEIGHT // 2 + 50))

    pygame.display.flip()
    self.clock.tick(FPS)
```

```
def close(self):
    pygame.quit()
```

- render: Updates the game screen, drawing the dinosaur and trees. Displays the score and game over messages.

- close: Closes Pygame and releases resources.

2.7. Environment Registration

```
gym.envs.registration.register(
    id='Dino-v0',
    entry_point='dino_env:DinoEnv',
)
```

- gym.envs.registration.register: Registers the Dino game environment with the Gym library, making it accessible through the Gym API.

3. Game Execution File (play.py)

This section explains the execution of the game and managing the main game loop.

3.1. Libraries and Constants

```
import pygame

# Initialize Pygame
pygame.init()

# Initialize the mixer module
pygame.mixer.init()
```

```
# Load the background music
pygame.mixer.music.load('assets/Neset-Ertas-Gel-Yanima-Gel.mp3')

# Play the music in an infinite loop
pygame.mixer.music.play(-1)

# Constants
SCREEN_WIDTH, SCREEN_HEIGHT = 800, 600
GROUND_HEIGHT = 550
GRAVITY = 2
FPS = 60

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
```

3.2. Game Class

The Game class contains the main game loop and basic game logic.

```
class Game:
    def __init__(self):
        # Screen
        self.screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
        pygame.display.set_caption("Dino")

        # Clock
        self.clock = pygame.time.Clock()

        # Load images
        self.dino_image = pygame.image.load('assets/dino.png')
        self.tree_image = pygame.image.load('assets/tree.png')
```

```
        self.dino_width, self.dino_height = self.dino_image.get_size()
        self.tree_width, self.tree_height = self.tree_image.get_size()

        # Dinosaur
        self.dino = pygame.Rect(100, GROUND_HEIGHT - self.dino_height, self.dino_width, self.dino_height)
        self.dino_vel_y = 0
        self.is_jumping = False

        # Trees
        self.trees = [pygame.Rect(SCREEN_WIDTH, GROUND_HEIGHT - self.tree_height, self.tree_width, self.tree_height)]
        self.tree_speed = 10

        # Score
        self.score = 0

        # Game state
        self.game_over = False

        # Font
        self.font = pygame.font.Font(None, 36)

        # Main game loop flag
        self.running = True
```

3.3. Game Functions

The reset, handle_events, update_game_logic, and draw_elements functions manage the game's main logic and interface.

```
def reset(self):
    self.game_over = False
    self.score = 0
```

```
    self.dino.y = GROUND_HEIGHT - self.dino_height
    self.trees = [pygame.Rect(SCREEN_WIDTH, GROUND_HEIGHT - self.tree_height, self.tree_width, self.tree_height)]
    self.dino_vel_y = 0
    self.is_jumping = False

def handle_events(self):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self.running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE and self.dino.bottom >= GROUND_HEIGHT and not self.game_over:
                self.dino_vel_y = -30
                self.is_jumping = True
            if event.key == pygame.K_r and self.game_over:
                self.reset()

def update_game_logic(self):
    if not self.game_over:
        # Dinosaur movement
        if self.is_jumping:
            self.dino.y += self.dino_vel_y
            self.dino_vel_y += GRAVITY
            if self.dino.bottom > GROUND_HEIGHT:
                self.dino.bottom = GROUND_HEIGHT
                self.is_jumping = False
                self.dino_vel_y = 0

        # Trees movement
        for tree in self.trees:
            tree.x -= self.tree_speed
            if tree.x < -self.tree_width:
                self.trees.remove(tree)
                self.trees.append(
                    pygame.Rect(SCREEN_WIDTH, GROUND_HEIGHT - self.tree_height, self.tree_width, self.tree_height))
                self.score += 10  # Increment score as trees go off screen
```

```
        # Collision detection
        for tree in self.trees:
            if self.dino.collides(tree):
                self.game_over = True  # Game over condition

    def draw_elements(self):
        self.screen.fill(WHITE)

        # Draw dinosaur
        self.screen.blit(self.dino_image, self.dino)

        # Draw trees
        for tree in self.trees:
            self.screen.blit(self.tree_image, tree)

        # Display score
        text = self.font.render(f'Score: {self.score}', True, BLACK)
        self.screen.blit(text, (10, 10))

        # Game Over screen
        if self.game_over:
            text_game_over = self.font.render(f'Game Over! Final Score: {self.score}', True, BLACK)
            self.screen.blit(text_game_over, (SCREEN_WIDTH // 2 - text_game_over.get_width() // 2, SCREEN_HEIGHT // 2))
            text_restart = self.font.render('Press R to Restart', True, RED)
            self.screen.blit(text_restart, (SCREEN_WIDTH // 2 - text_restart.get_width() // 2, SCREEN_HEIGHT // 2 + 50))

    def run(self):
        while self.running:
            self.handle_events()
            self.update_game_logic()
            self.draw_elements()

            # Update the display
            pygame.display.flip()

            # Frame rate
```

```
        self.clock.tick(FPS)

    pygame.quit()
```

- reset: Resets the game to its initial state.

- handle_events: Handles user inputs.

- update_game_logic: Updates the game logic.

- draw_elements: Draws the game elements.

- run: Runs the main game loop.

3.4. Starting the Game

```
if __name__ == "__main__":
    game = Game()
    game.run()
```

- if name == "main": Starts and runs the Game class.

4. Model Training (train_agent.py)

This section explains the training of the game and the reinforcement learning model.

4.1. Libraries and Environment

```
import os
import gym
import torch
from stable_baselines3 import PPO
from stable_baselines3.common.utils import get_device
from dino_env import DinoEnv
```

```
import warnings

warnings.filterwarnings("ignore")

# Create and wrap the environment
env = gym.make('Dino-v0')
```

- gym and PPO: Used to create the environment and apply reinforcement learning algorithms.

- torch: Provides device control for GPU usage.

4.2. Model Definition and Training

```
# Check if GPU is available and set the device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")

# Define the model with tuned hyperparameters
model = PPO(
  'MlpPolicy',
  env,
  verbose=0,
  device=device,
  learning_rate=3e-4,
  n_steps=4096,
  batch_size=4096,
  n_epochs=50,
  gamma=0.9999999
)

# Train the model
model.learn(total_timesteps=400000)

# Save the model
```

```
model.save('dino_model')


# Close the environment

env.close()
```

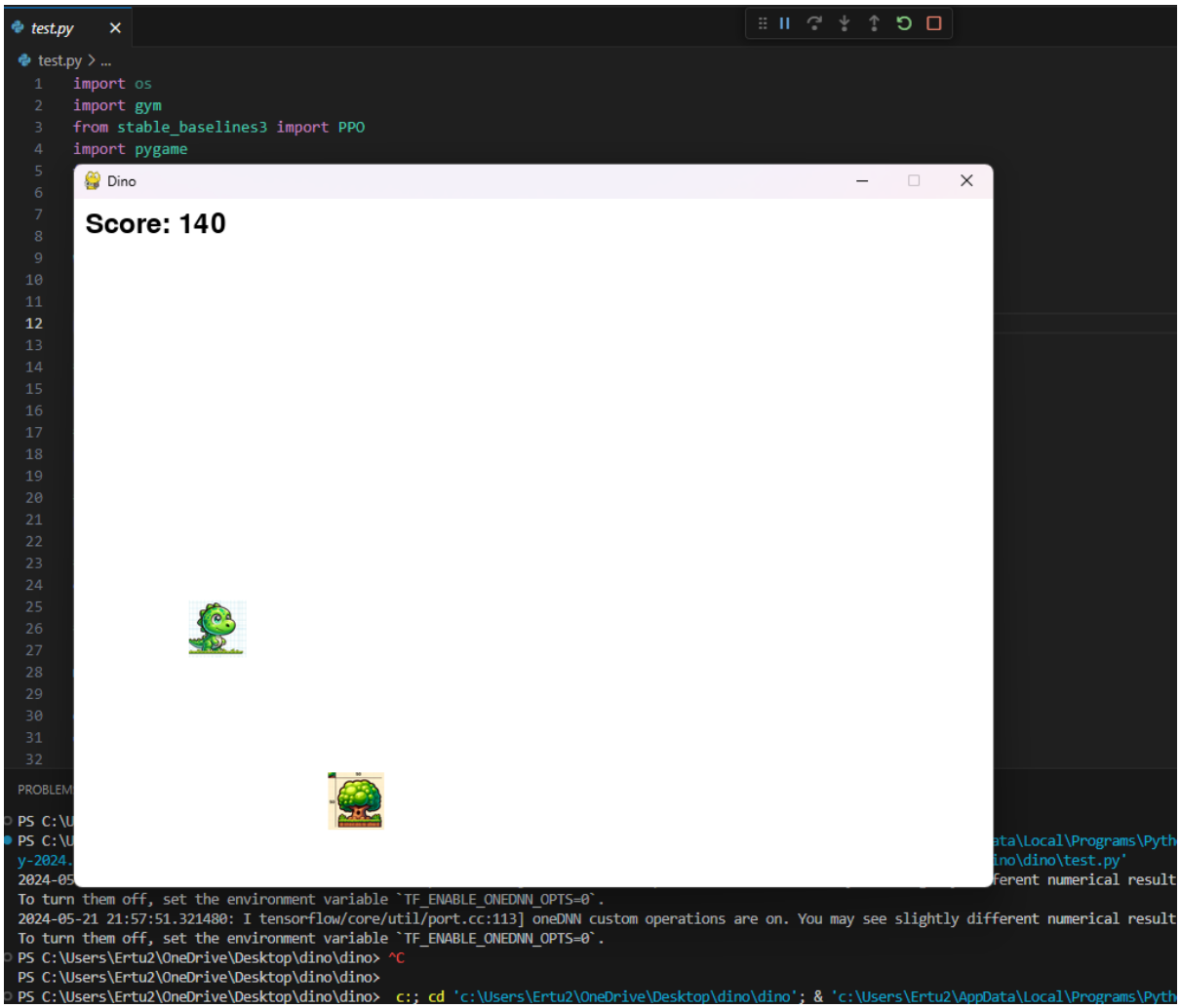
- device: Checks and sets the device for GPU or CPU usage.

- PPO: Defines and trains the model using the Proximal Policy Optimization algorithm.

- model.learn: Trains the model for a specified number of steps.

- model.save: Saves the trained model.

5. Model Testing (test.py)



This section explains the testing of the trained model and observing the game's performance.

5.1. Libraries and Environment

```
import os

import gym

from stable_baselines3 import PPO

import pygame

from dino_env import DinoEnv

import warnings


warnings.filterwarnings("ignore")


pygame.init()


# Initialize the mixer module

pygame.mixer.init()


# Load the background music

pygame.mixer.music.load('assets/Neset-Ertas-Gel-Yanima-Gel.mp3')


# Play the music in an infinite loop

pygame.mixer.music.play(-1)


# Load the environment

env = gym.make('Dino-v0')
```

- gym and PPO: Used to load the environment and the trained model.

- pygame: Used to handle game graphics and user interaction.

5.2. Model Loading and Testing

```
# Load the trained model
model = PPO.load("dino_model.zip")

obs = env.reset()
done = False

# Game over handling to display the final score and restart option
while True:
    env.render()
    action, _states = model.predict(obs)
    obs, reward, terminated, info = env.step(action)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            env.close()
            exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_r:
                obs = env.reset()

env.close()
```

- PPO.load: Loads the trained model.

- env.reset: Resets the environment to the initial state.

- while True: Game loop that renders the environment, takes the model's predictions, and performs steps.

6. Conclusion

This report has detailed the development, training, and testing processes of the Dino game. The game was developed using the Pygame library and trained with reinforcement learning algorithms in a Gym environment. The trained model successfully plays the game and achieves high scores. This work demonstrates how reinforcement learning algorithms can be used in game development and AI training.