DECISION MODELING MINI PROJECT

PROJECT REPORT

TITLE

DRAGON CHERRY FEAST: THE DRAGON'S TALE

TEAM MEMBERS:

HARSH PATIL (4505313)

VIDYA DINKAR (4505042)

DHEERAJ SINGH KARKI (4748118)

HARSH SINGH (4505045)

JANHAVI HINDURAO (4555309)

Table Of Content

Introduction

In recent years, the application of artificial intelligence (AI) in game development has significantly transformed the gaming industry. This project explores the integration of AI into a classic arcade-style game, reminiscent of Pac-Man, where the player controls a character, referred to as the "hungry dragon," in a grid-based environment. The objective is to navigate through the grid to collect cherries while avoiding enemies. Leveraging the Gym environment from OpenAI, this project provides a structured framework for developing and testing reinforcement learning algorithms. Additionally, Pygame is utilized for rendering the game visuals, offering an interactive and engaging user experience. The AI component enhances the gameplay by introducing decision modeling, where AI-driven enemies dynamically adapt to the player's movements.

The game's mechanics involve the hungry dragon moving within a 20x20 grid, collecting regular and special food items while evading six randomly positioned enemies. Each action and decision made by the player influences the game's outcome, making it a practical example of decision modeling in a controlled environment. Designed to provide an immersive experience where strategic thinking and quick decision-making are crucial, the player must navigate the hungry dragon using keyboard inputs to move up, down, left, or right. The game dynamically generates food items and special food that provide score increments and bonuses, while enemies move towards the dragon based on a simple AI algorithm, making their behavior unpredictable and challenging.

This project not only highlights the fun and engaging aspects of AI in gaming but also serves as an educational tool for understanding the principles of AI and decision modeling. By using reinforcement learning concepts, the game demonstrates how AI can be used to create responsive and intelligent non-player characters (NPCs) that enhance the overall gaming experience. In summary, "Dragon Cherry FEAST" showcases the potential of AI in creating dynamic, engaging, and challenging games, integrating AI into traditional game mechanics to offer a fresh and exciting gameplay experience. The project underscores the importance of AI in modern game development and provides a practical example of its application in a real-world scenario.

Problem Statement

The primary objective of this project is to develop a game that integrates artificial intelligence (AI) to enhance player engagement and challenge. Traditional games often rely on predefined behaviors for non-player characters (NPCs), which can lead to predictable and monotonous gameplay. This project aims to address this issue by implementing dynamic decision modeling through AI, creating an unpredictable and challenging environment for players.

In "Dragon Cherry FEAST," the player controls a hungry dragon navigating a 20x20 grid to collect food items while avoiding enemies. The challenge lies in designing an AI system that can effectively simulate intelligent behavior for the enemies, making the game more engaging and less predictable. The enemies need to adapt to the player's movements, requiring the player to think strategically and make quick decisions to succeed.

Additionally, the project seeks to demonstrate the practical application of reinforcement learning algorithms in a gaming context. The AI-driven enemies must be able to learn and adapt, providing a dynamic gameplay experience that evolves over time. This involves not only the technical challenge of implementing such algorithms but also ensuring that the game remains balanced and enjoyable for players.

The problem statement for this project encompasses the following key points:

1. Dynamic Enemy Behavior: Developing AI algorithms that enable enemies to adapt to the player's actions, creating a challenging and unpredictable gameplay experience.
2. Decision Modeling: Implementing decision modeling techniques to allow the AI to make intelligent choices based on the game state.
3. Player Engagement: Ensuring the game is engaging and fun, with a balance between challenge and playability.
4. Reinforcement Learning: Demonstrating the application of reinforcement learning in a game setting, showcasing its potential to create responsive and adaptive NPCs.

By addressing these challenges, "Dragon Cherry FEAST" aims to provide a compelling example of how AI can be integrated into game development to create more dynamic, challenging, and engaging experiences for players.

Objective

- Develop AI-Driven Enemies: Create AI algorithms to simulate intelligent enemy behavior, enhancing game difficulty and ensuring unpredictable gameplay.
- Implement Decision Modeling: Utilize decision modeling techniques to enable the AI to make strategic choices, improving the overall game dynamics and challenge.
- Enhance Player Engagement: Design the game to be highly engaging and fun, balancing challenge and playability to maintain player interest and enjoyment.
- Apply Reinforcement Learning: Integrate reinforcement learning to enable enemies to adapt to the player's actions, demonstrating practical AI applications in gaming.
- Create Dynamic Gameplay: Ensure the game environment evolves with the player's decisions, providing a unique and varied experience each time it is played.
- Educational Demonstration: Showcase the principles of AI and decision modeling in a practical context, serving as an educational tool for understanding AI in game development.

Scope

The scope of the "Dragon Cherry FEAST" project encompasses several key areas, from technical implementation to educational value and future enhancements:

- Game Development: The project involves developing a fully functional game where the player controls a character in a grid-based environment, collecting food items while avoiding enemies. The game will be built using Python, leveraging the OpenAI Gym environment for reinforcement learning and Pygame for graphical rendering.
- AI Integration: A primary focus of the project is the integration of AI to control enemy behaviors. This includes implementing decision modeling and reinforcement learning algorithms to create dynamic, adaptive enemies that respond intelligently to the player's actions.
- User Interaction: The game will be designed to be user-friendly, with simple controls (keyboard inputs) and clear visual feedback. The player's objective will be straightforward, promoting easy understanding and engagement.
- Educational Value: The project aims to serve as an educational tool, demonstrating the application of AI and decision modeling in game development. It will provide insights into how AI can enhance gameplay and create more engaging and challenging experiences.
- Scalability and Extendibility: The game is designed with future enhancements in mind. This includes the potential to add more complex AI behaviors, additional levels, varied challenges, and new game mechanics to keep the gameplay fresh and exciting.
- Performance and Testing: Ensuring the game runs smoothly and efficiently is crucial. The project will include performance optimization and thorough testing to identify and fix any bugs, ensuring a seamless user experience.

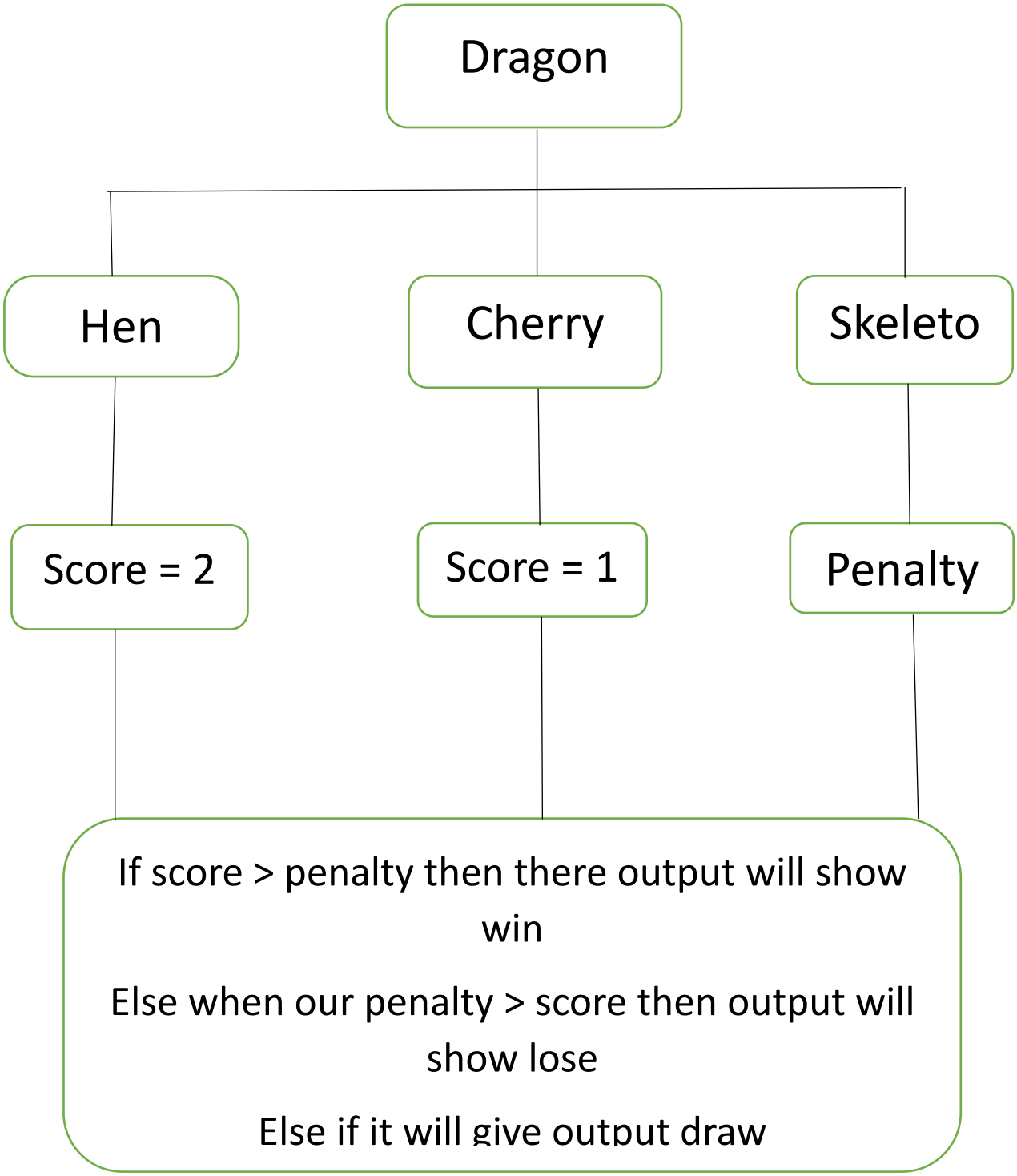
Overall, the scope of "Dragon Cherry FEAST" covers the creation of an engaging and educational AI-driven game, with a focus on demonstrating the practical applications of AI in enhancing gameplay dynamics and providing a foundation for future developments and extensions.

How AI is Used in the Project

1. Enemy Behavior Modeling: Artificial intelligence is utilized to simulate intelligent behavior for the enemies in "Dragon Cherry FEAST." Decision modeling techniques are employed to determine enemy movements, making them responsive and adaptable to the player's actions.
2. Reinforcement Learning: The game incorporates reinforcement learning algorithms to enable enemies to learn and adapt based on feedback from the game environment. This allows enemies to improve their strategies over time, creating a more challenging and dynamic gameplay experience.
3. Dynamic Decision Making: AI algorithms are programmed to make strategic decisions based on the current game state. This includes determining the optimal path for enemies to pursue the player, avoiding obstacles, and prioritizing targets such as food items.
4. Adaptive Gameplay: By integrating AI, the game environment becomes more dynamic and unpredictable. Enemies adjust their behavior in real-time, responding to changes in the player's position and game objectives. This adaptive gameplay ensures that each playthrough offers a unique and engaging experience.
5. Enhanced Player Challenge: The use of AI in "Dragon Cherry FEAST" increases the challenge level for players. Enemies become more intelligent and formidable adversaries, requiring players to strategize and adapt their tactics to succeed.
6. Educational Value: Beyond gameplay enhancements, the AI components in the project serve an educational purpose. Players gain insight into AI principles such as decision modeling and reinforcement learning, fostering a deeper understanding of AI concepts in a practical context.

   By leveraging AI techniques such as decision modeling and reinforcement learning, "Dragon Cherry FEAST" showcases the potential of AI to enhance game dynamics, provide engaging gameplay experiences, and serve as an educational tool for exploring AI principles in game development.

System Flow


Dragon
Hen
Cherry
Skeleto
Score = 2
Score = 1
Penalty
If score > penalty then there output will show win
Else when our penalty > score then output will show lose
Else if it will give output draw

1. Initialization: The game initializes by creating a grid-based environment, initializing the player (hungry dragon), enemies, and food items.
2. Game Loop: The game enters a loop where it continuously checks for player inputs and updates the game state accordingly.
3. Player Movement: Upon receiving player inputs, the game updates the player's position based on the chosen direction (up, down, left, or right).
4. Collision Detection: The game checks for collisions between the player and food items, special items, and enemies. If a collision occurs, appropriate actions are taken (e.g., scoring points, triggering special effects).
5. Enemy Movement: The enemies move according to their AI algorithms, which may involve pursuing the player, patrolling specific areas, or avoiding obstacles.
6. Scoring and Penalties: Points are awarded for collecting food items and completing objectives. Penalties may be incurred for colliding with enemies or failing to meet objectives within a specified time limit.
7. Game Over Check: The game checks for conditions that signal the end of the game, such as achieving a certain score, running out of time, or losing all health points.
8. End Game: When the game ends, a message is displayed to the player indicating whether they won or lost. The player may be given the option to restart the game or exit.
9. Optional Features: Additional features such as sound effects, animations, and user interface elements may be incorporated to enhance the gaming experience.
10. Repeat: The game continues to loop, allowing the player to play again or exit the game as desired.

    This system flow outlines the basic sequence of events that occur during gameplay, illustrating how the various components of the game interact to provide an engaging and dynamic experience for the player.

Future Scope

- Advanced AI Behaviors: Enhance enemy behaviors with more complex AI algorithms, incorporating advanced decision-making techniques and learning models for increased challenge and variety.
- Multiple Game Modes: Introduce new game modes with varying objectives, challenges, and environments, providing players with diverse and immersive gameplay experiences.
- Online Multiplayer Support: Implement online multiplayer functionality, allowing players to compete or cooperate with friends in real-time, adding a social and competitive element to the game.
- Dynamic Level Generation: Develop procedural level generation algorithms to create dynamic and infinitely replayable levels, offering fresh challenges and surprises with each playthrough.
- Cross-Platform Compatibility: Optimize the game for cross-platform compatibility, enabling players to enjoy "Dragon Cherry FEAST" on a variety of devices, including mobile phones, tablets, and gaming consoles.
- Community Content Creation: Introduce tools and support for community-driven content creation, allowing players to create and share their own levels, characters, and AI behaviors, fostering a vibrant and creative community around the game.
- AI vs. AI Battles: Enable AI-controlled characters to engage in battles against each other, allowing players to spectate and analyze AI behaviors, fostering experimentation and learning opportunities.

Conclusion

"Dragon Cherry FEAST" represents a successful integration of artificial intelligence (AI) into game development, showcasing its potential to enhance gameplay dynamics and provide engaging experiences for players. Through the implementation of AI-driven enemies, decision modeling, and reinforcement learning algorithms, the game offers a challenging and dynamic gameplay environment.

The project has not only demonstrated the practical application of AI in gaming but also served as an educational tool for understanding AI principles such as decision modeling and reinforcement learning in a real-world context. By providing insights into how AI can be used to create responsive and adaptive non-player characters (NPCs), the project has contributed to the broader discourse on AI in game development.

Moving forward, there are numerous opportunities for further enhancement and expansion of "Dragon Cherry FEAST." Future iterations of the game could include advanced AI behaviors, multiple game modes, online multiplayer support, dynamic level generation, cross-platform compatibility, community content creation, and AI vs. AI battles. These enhancements would further enrich the gameplay experience and broaden the appeal of the game to a wider audience.

In conclusion, "Dragon Cherry FEAST" exemplifies the potential of AI to transform traditional game mechanics, offering a glimpse into the future of interactive entertainment. By leveraging AI techniques to create dynamic and engaging gameplay experiences, the project has demonstrated the significant impact of AI on the gaming industry and paved the way for future innovations in game design and development.

References

1. OpenAI Gym. (n.d.). OpenAI Gym. Retrieved from https://gym.openai.com/
2. Pygame Documentation. (n.d.). Pygame Documentation. Retrieved from https://www.pygame.org/docs/
3. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.
4. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533. https://doi.org/10.1038/nature14236
5. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489. https://doi.org/10.1038/nature16961
6. Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The Arcade Learning Environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253–279. https://doi.org/10.1613/jair.3912

Appendix

Code-

```
import gym
from gym import spaces
import numpy as np
import pygame
import random
import time
from collections import deque


class PacmanEnv(gym.Env):
    def __init__(self):
        super(PacmanEnv, self).__init__()
        self.grid_size = (20, 20)  # Grid size
        self.screen_size = 600  # Screen size
        self.cell_size = self.screen_size // self.grid_size[0]  # Calculate cell size based on grid size


        # Initialize game attributes
        self.hungry_dragon = [10, 10]  # Start Pac-Man at the center
        self.food_pos = [random.randint(0, 19), random.randint(0, 19)]  # Random food position
        self.special_food_pos = None  # Special food position
        self.enemy_positions = [[random.randint(0, 19), random.randint(0, 19)] for _ in range(6)]  # Six random enemy positions
        self.done = False
        self.score = 0
        self.penalty_score = 0
        self.steps = 0
```

```
        self.max_steps = 20

        self.action_space = spaces.Discrete(4)  # Four possible actions
        self.observation_space = spaces.Box(low=0, high=4, shape=(20, 20), dtype=np.float32)

        # Initialize Pygame and sounds
        pygame.mixer.init()
        self.cherry_sound = pygame.mixer.Sound("cherry_sound.wav")
        self.special_food_sound = pygame.mixer.Sound("dragon_hurt.mp3")
        self.enemy_eaten_sound = pygame.mixer.Sound("dragon_hurt.mp3")
        self.enemy_eat_dragon_sound = pygame.mixer.Sound("ha-ha.mp3")
        pygame.mixer.music.load("Hitman.mp3")
        pygame.mixer.music.play(-1)  # Infinite loop

        pygame.init()
        pygame.font.init()
        self.screen = pygame.display.set_mode((self.screen_size, self.screen_size))
        pygame.display.set_caption('Dragon Cherry FEAST')

    def reset(self):
        self.hungry_dragon = [10, 10]
        self.food_pos = [random.randint(0, 19), random.randint(0, 19)]
        self.special_food_pos = None
        self.enemy_positions = [[random.randint(0, 19), random.randint(0, 19)] for _ in
range(6)]
        self.done = False
        self.score = 0
        self.penalty_score = 0
        self.steps = 0
        return self._get_obs()
```

```
    def _get_obs(self):
        obs = np.zeros(self.grid_size)
        obs[self.hungry_dragon[0], self.hungry_dragon[1]] = 1
        obs[self.food_pos[0], self.food_pos[1]] = 2
        if self.special_food_pos:
            obs[self.special_food_pos[0], self.special_food_pos[1]] = 3
        for pos in self.enemy_positions:
            obs[pos[0], pos[1]] = 4
        return obs


    def step(self, action):
        prev_food_pos = self.food_pos.copy()
        prev_special_food_pos = self.special_food_pos.copy() if self.special_food_pos else
None


        if action == 0:  # Up
            self.hungry_dragon[0] = max(0, self.hungry_dragon[0] - 1)
        elif action == 1:  # Down
            self.hungry_dragon[0] = min(self.grid_size[0] - 1, self.hungry_dragon[0] + 1)
        elif action == 2:  # Left
            self.hungry_dragon[1] = max(0, self.hungry_dragon[1] - 1)
        elif action == 3:  # Right
            self.hungry_dragon[1] = min(self.grid_size[1] - 1, self.hungry_dragon[1] + 1)


        reward = 0
        if self.hungry_dragon == prev_food_pos:
            reward = 10
            self.food_pos = [random.randint(0, 19), random.randint(0, 19)]
            self.score += 1
            self.steps += 1
            self.cherry_sound.play()
```

```
            if self.score % 4 == 0:
                self.special_food_pos = [random.randint(0, 19), random.randint(0, 19)]

        if self.special_food_pos and self.hungry_dragon == self.special_food_pos:
            reward = 2
            self.special_food_pos = None
            self.score += 2
            self.special_food_sound.play()

        for enemy_pos in self.enemy_positions:
            if self.hungry_dragon == enemy_pos:
                reward = -5
                self.penalty_score += 1
                self.steps += 1
                self.enemy_eaten_sound.play()
                self.enemy_eat_dragon_sound.play()

        if (self.score + self.penalty_score) >= 20:
            self.done = True

        return self._get_obs(), reward, self.done, {}

    def render(self, mode='human'):
        DARK_BLUE = (0, 0, 128)
        WHITE = (255, 255, 255)
        GREY = (192, 192, 192)
        GREEN = (0, 255, 0)
        RED = (255, 0, 0)
        BLACK = (0, 0, 0)
```

```
        self.screen.fill(DARK_BLUE)

        # Draw the grid with alternating white and grey squares
        for x in range(self.grid_size[0]):
            for y in range(self.grid_size[1]):
                color = WHITE if (x + y) % 2 == 0 else GREY
                pygame.draw.rect(self.screen, color, pygame.Rect(y * self.cell_size, x * self.cell_size, self.cell_size, self.cell_size))

        # Draw the grid lines
        for x in range(self.grid_size[0]):
            for y in range(self.grid_size[1]):
                pygame.draw.rect(self.screen, BLACK, pygame.Rect(y * self.cell_size, x * self.cell_size, self.cell_size, self.cell_size), 1)

        # Load and scale images
        pacman_image = pygame.image.load("dragon.png")
        cherry_image = pygame.image.load("cherry.png")
        special_food_image = pygame.image.load("special_food.png")
        enemy_image = pygame.image.load("skeleton.png")

        pacman_image = pygame.transform.scale(pacman_image, (self.cell_size, self.cell_size))
        cherry_image = pygame.transform.scale(cherry_image, (self.cell_size, self.cell_size))
        special_food_image = pygame.transform.scale(special_food_image, (self.cell_size, self.cell_size))
        enemy_image = pygame.transform.scale(enemy_image, (self.cell_size, self.cell_size))

        # Blit images
        self.screen.blit(pacman_image, (self.hungry_dragon[1] * self.cell_size, self.hungry_dragon[0] * self.cell_size))
        self.screen.blit(cherry_image, (self.food_pos[1] * self.cell_size, self.food_pos[0] * self.cell_size))
```

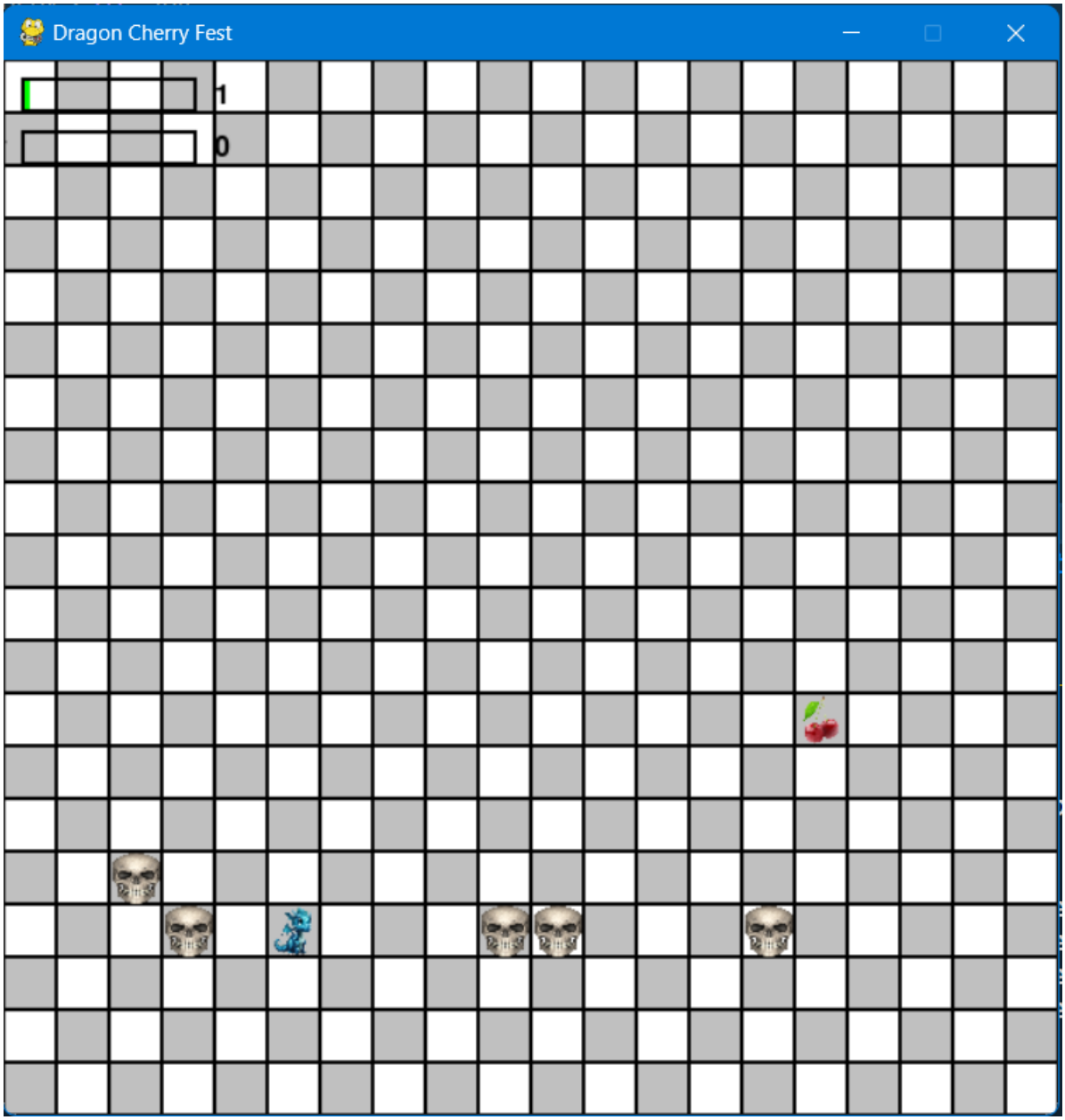
```
        if self.special_food_pos:
            self.screen.blit(special_food_image, (self.special_food_pos[1] * self.cell_size, self.special_food_pos[0] * self.cell_size))
        for enemy_pos in self.enemy_positions:
            self.screen.blit(enemy_image, (enemy_pos[1] * self.cell_size, enemy_pos[0] * self.cell_size))

        # Display score and penalty bars
        self._display_score_bar(GREEN, BLACK, 10, 10, self.score, self.max_steps, "Score")
        self._display_score_bar(RED, BLACK, 10, 40, self.penalty_score, self.max_steps, "Penalty")

        if self.done:
            self._display_end_message()

        pygame.display.flip()

    def _display_score_bar(self, fill_color, border_color, x, y, score, max_score, label):
        bar_width = 100
        bar_height = 20
        percentage = min(score / max_score, 1.0)
        filled_width = int(percentage * bar_width)
        pygame.draw.rect(self.screen, fill_color, pygame.Rect(x, y, filled_width, bar_height))
        pygame.draw.rect(self.screen, border_color, pygame.Rect(x, y, bar_width, bar_height), 2)
        font = pygame.font.Font(None, 24)
        label_text = font.render(f"{label}", True, border_color)
        self.screen.blit(label_text, (x - 70, y + 2))  # Adjusted position
        score_text = font.render(f"{score}", True, border_color)
        self.screen.blit(score_text, (x + bar_width + 10, y + 2))

    def _display_end_message(self):
```

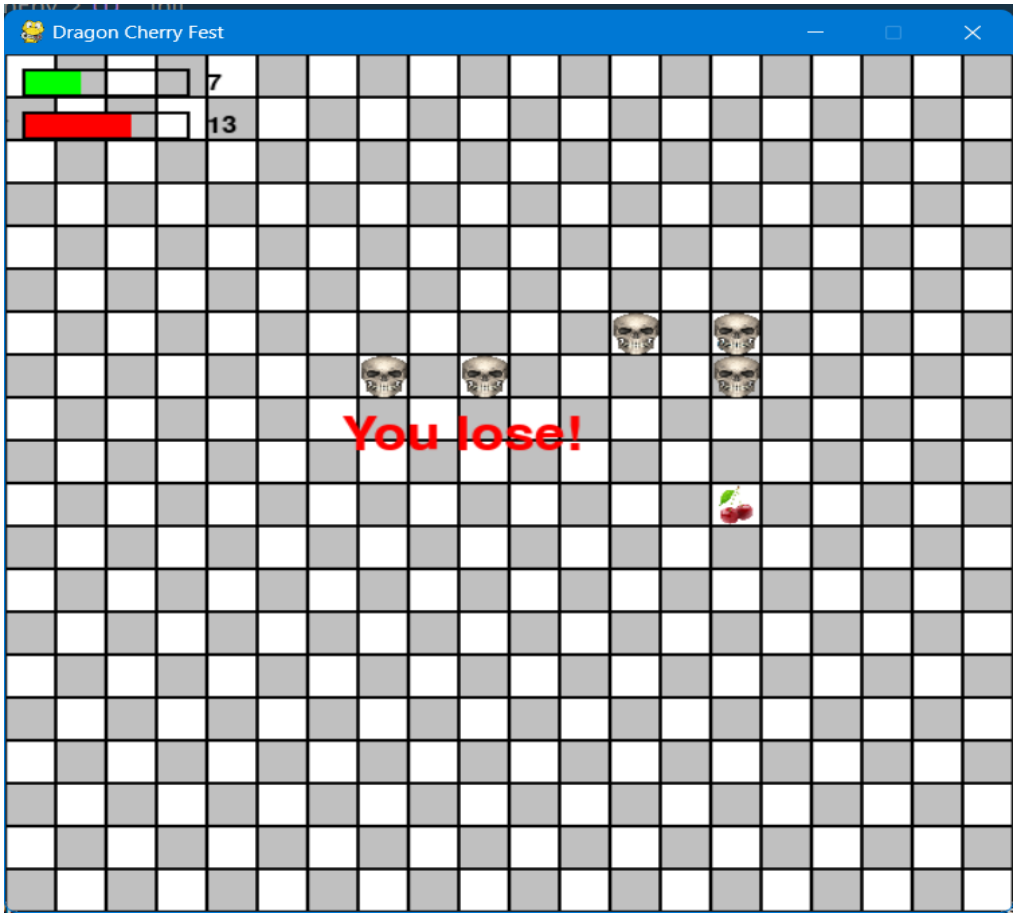
```
        message_font = pygame.font.Font(None, 48)
        if self.score > self.penalty_score:
            message_text = message_font.render("You win!", True, (0, 255, 0))
        elif self.score < self.penalty_score:
            message_text = message_font.render("You lose!", True, (255, 0, 0))
        else:
            message_text = message_font.render("It's a draw!", True, (255, 255, 0))
        self.screen.blit(message_text, (self.screen_size // 2 - 100, self.screen_size // 2 - 50))

    def bfs(self, start, goal):
        queue = deque([(start, [])])
        visited = set()
        while queue:
            (current, path) = queue.popleft()
            if current == goal:
                return path
            if current in visited:
                continue
            visited.add(current)
            x, y = current
            neighbors = [(x - 1, y), (x + 1, y), (x, y - 1), (x, y + 1)]
            valid_neighbors = [n for n in neighbors if 0 <= n[0] < self.grid_size[0] and 0 <= n[1]
< self.grid_size[1]]
            for neighbor in valid_neighbors:
                if neighbor not in visited:
                    queue.append((neighbor, path + [neighbor]))
        return []

    def move_enemies(self):
        for i, enemy_pos in enumerate(self.enemy_positions):
            target = (
```

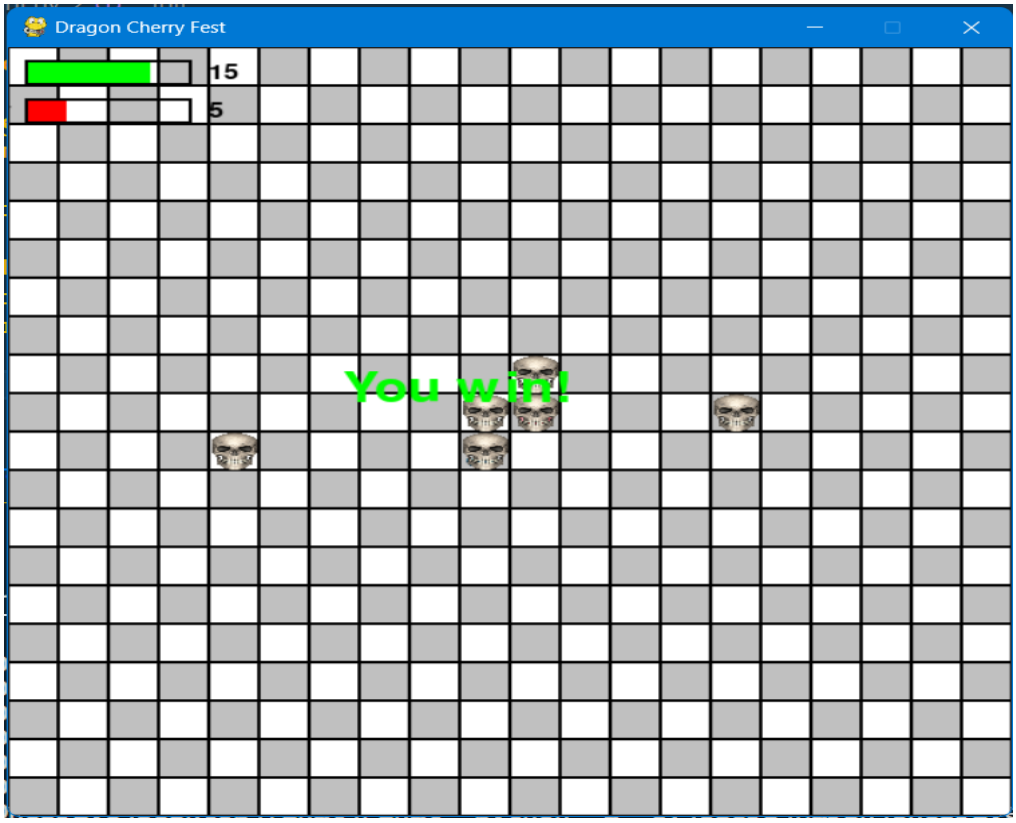
```
                self.hungry_dragon[0] + random.randint(-1, 1),
                self.hungry_dragon[1] + random.randint(-1, 1)
            )
            path = self.bfs(tuple(enemy_pos), target)
            if path:
                self.enemy_positions[i] = list(path[0])
            if self.enemy_positions[i] == self.hungry_dragon:
                self.penalty_score += 1
                self.steps += 1



def run_game():
    env = PacmanEnv()
    clock = pygame.time.Clock()
    running = True
    enemy_move_counter = 0
    end_message_displayed = False

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    env.step(0)
                elif event.key == pygame.K_DOWN:
                    env.step(1)
                elif event.key == pygame.K_LEFT:
                    env.step(2)
```

```
            elif event.key == pygame.K_RIGHT:
                env.step(3)

        if enemy_move_counter % 10 == 0:  # Enemies move twice as fast
            env.move_enemies()

        if (env.score + env.penalty_score) >= 20 and not end_message_displayed:
            env.done = True
            end_message_displayed = True
            env.render()
            pygame.display.flip()
            time.sleep(2)  # Wait for 2 seconds
            running = False

        enemy_move_counter += 1
        env.render()
        clock.tick(30)

    pygame.quit()


if __name__ == "__main__":
    run_game()
```

Output




Dragon Cherry Fest
7
13
You lose!


Dragon Cherry Fest
15
5
You win!