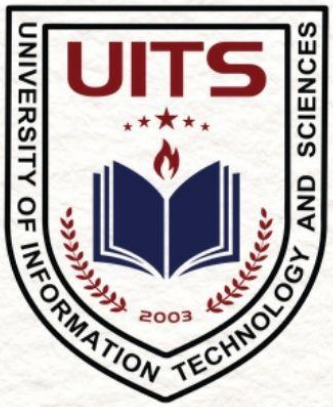
University of Infromation Technology & Sciences

Department of

Computer Science and Engineering



Project Title : Treasure Hunt Game

Team Name : ShowStoppers

Course Title: Artificial Intelligence Lab

Course Code : CSE-312

Submission Date : 19.10.24

Submitted To :-

Saima Siddique Tashfia

Lecturer, CSE, UITS

Submitted By :-

Name : Md. Rahmatullah Ashik

Id : 0432220005101007

Name : Tasnuba Akter

Id : 0432220005101014

Name : Ribben Sharma

Id : 0432220005101028

Name : Md. Abdhullah- Al - Numan

Id : 0432220005101039

Batch 52

Section : 5A

Semester: Autumn 24

Introduction

The goal of this project is to create an engaging 2D treasure hunt game where players or AI agents navigate a map to find hidden treasures. The game will incorporate advanced algorithms for navigation, pathfinding, and decision-making to enhance the experience.

Key Features

1. Single-Player Mode:
The player competes against time to locate the treasure on a predefined map. Obstacles and hints are placed strategically to make the game challenging.
2. Two-Player Mode:
Players compete against each other to reach the treasure first. Includes adversarial AI for single-player mode where the player competes against an AI-controlled agent.
3. Dynamic Map Generation:
Randomly generates maps for every new game to ensure variety.
4. Pathfinding Algorithms:
A* Algorithm for optimal pathfinding. BFS/DFS for alternative search strategies.
5. Interactive Hints System:
Players can discover hints like directions or distances to the treasure, adding strategic elements.

Game Design and Concept

The game is divided into three distinct levels with increasing difficulty: Easy Level: Jungle Explorer, Medium Level: Desert Dare, and Hard Level: Mountain Madness.

The player moves through a grid where obstacles and treasures are randomly placed. The goal is to find the treasure while avoiding obstacles that could end the game prematurely.

Grid-Base Gameplay

The game utilizes a grid structure, which organizes the screen into smaller cells. This grid is used to place the player, obstacles, and treasure in specific positions.

The player moves one grid cell at a time using the arrow keys, ensuring that each step is aligned with the grid layout.

Algorithm Used

1. Pathfinding Algorithms:
A Algorithm:* For efficient navigation to find the shortest path to the treasure.
BFS/DFS: For exploration and obstacle traversal.
2. Adversarial Search: Minimax Algorithm with Alpha-Beta Pruning: Used in two-player mode for AI decision making.
3. Random Walk: For randomized treasure placement on the map.
4. Dynamic Scoring: Implements a scoring system to evaluate players' performance based on efficiency and time.

Technology Used

Programming Language: Python
Pygame: For 2D game development.
PyCharm: For coding and debugging.
Algorithms: Pathfinding (A*, BFS, DFS), Adversarial Search (Minimax).
Graphics and Assets: Basic sprites for characters, treasure, and obstacles.

Project Phases and Timeline

Phase 1: Planning and Design (Week 1) Define game rules, map structure, and algorithm requirements. Create sketches or mockups for the game layout.
Phase 2: Development (Weeks 2-3) Implement the game map and treasure placement. Integrate A* Algorithm and BFS/DFS for player navigation. Add obstacles and hints system.
Phase 3: Adversarial AI Development (Week 4) Implement adversarial AI using Minimax with Alpha-Beta Pruning. Test and refine the two-player mode.
Phase 4: Testing and Optimization (Week 5) Conduct gameplay testing to ensure algorithms perform correctly. Optimize game performance and fix bugs.
Phase 5: Deployment (Week 6) Finalize the game with UI enhancements. Prepare documentation and user guide.

Expected Outcome

The project will result in a fully functional 2D treasure hunt game that demonstrates the practical application of search and decision-making algorithms. It will be both entertaining and educational, showcasing algorithmic efficiency in game design.

Potential Enhancements (Future Scope)

Add multiplayer support for online gaming.
Introduce dynamic difficulty adjustment based on player performance.
Use machine learning for AI behavior improvement.

Source Code

```
import pygame
import random

# Initialize Pygame
pygame.init()

# Screen dimensions
WIDTH, HEIGHT = 800, 600
GRID_SIZE = 40

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
TREASURE_COLOR = (255, 223, 0)
ORANGE = (255, 165, 0)

# Initialize screen
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Treasure Hunt Game")

# Player settings
player_x, player_y = 0, 0
player_color = BLUE

# Grid settings
```

```
cols, rows = WIDTH // GRID_SIZE, HEIGHT // GRID_SIZE

# Levels configuration
levels = {
    "Jungle Explorer": 30,
    "Desert Dare": 45,
    "Mountain Madness": 60
}
current_level = "Jungle Explorer"

# Generate random obstacles and treasure
def generate_grid(num_obstacles):
    obstacles = []
    # Ensure that player starting position (0, 0) is not occupied by obstacles
    while len(obstacles) < num_obstacles:
        obstacle = (random.randint(0, cols - 1), random.randint(0, rows - 1))
        # Make sure the obstacle is not at (0, 0) and not overlapping with other
obstacles
        if obstacle != (0, 0) and obstacle not in obstacles:
            obstacles.append(obstacle)

    # Ensure treasure doesn't overlap with obstacles or player position
    treasure = (random.randint(0, cols - 1), random.randint(0, rows - 1))
    while treasure == (0, 0) or treasure in obstacles:
        treasure = (random.randint(0, cols - 1), random.randint(0, rows - 1))

    return obstacles, treasure

# Initialize the first level
obstacles, treasure = generate_grid(levels[current_level])
```

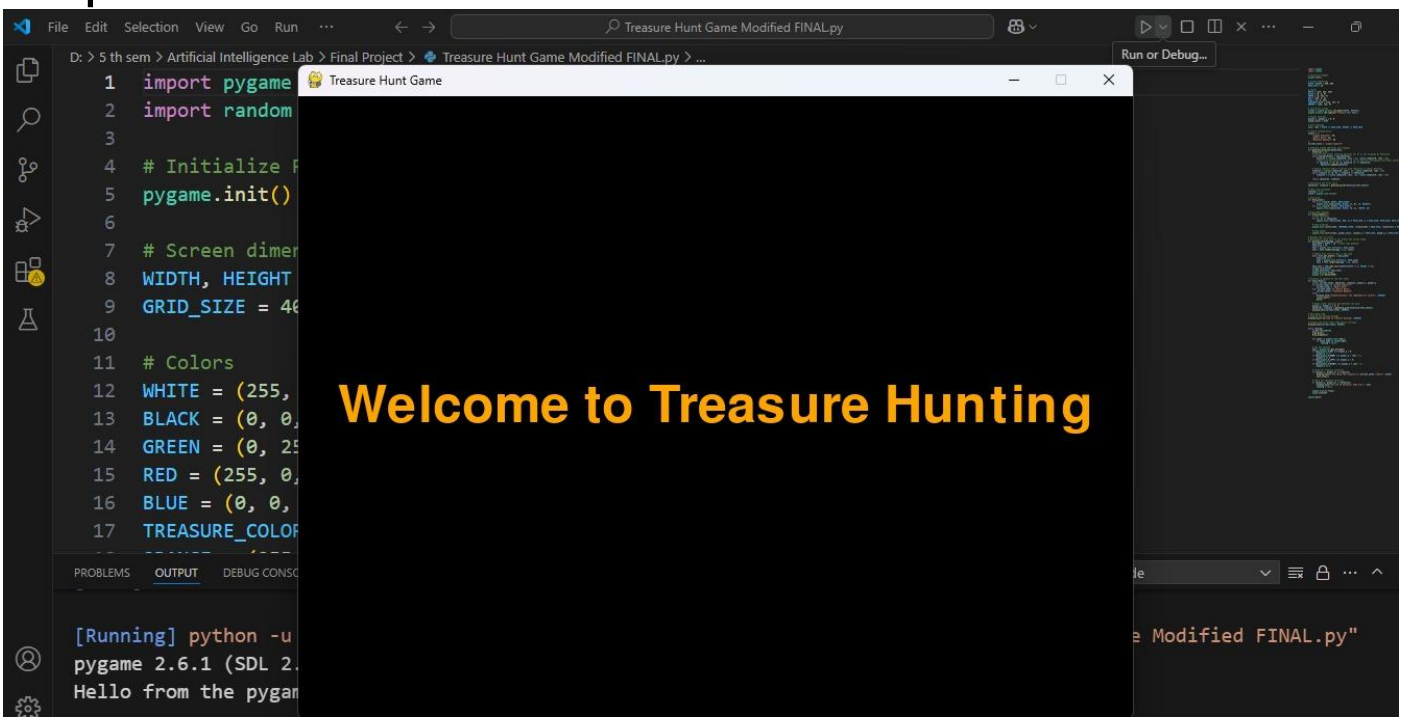
```
# Game loop variables
running = True
clock = pygame.time.Clock()

# Draw grid
def draw_grid():
    for x in range(0, WIDTH, GRID_SIZE):
        pygame.draw.line(screen, BLACK, (x, 0), (x, HEIGHT))
    for y in range(0, HEIGHT, GRID_SIZE):
        pygame.draw.line(screen, BLACK, (0, y), (WIDTH, y))

# Draw game elements
def draw_elements():
    # Draw obstacles
    for (x, y) in obstacles:
        pygame.draw.rect(screen, RED, (x * GRID_SIZE, y * GRID_SIZE, GRID_SIZE, GRID_SIZE))

    # Draw treasure
    pygame.draw.rect(screen, TREASURE_COLOR, (treasure[0] * GRID_SIZE, treasure[1] * GRID_SIZE, GRID_SIZE, GRID_SIZE))

    # Draw player
    pygame.draw.rect(screen, player_color, (player_x * GRID_SIZE, player_y * GRID_SIZE, GRID_SIZE, GRID_SIZE))

# Display text on screen
# Dynamically scale text to fit within the screen width
def display_text(message, color):
```

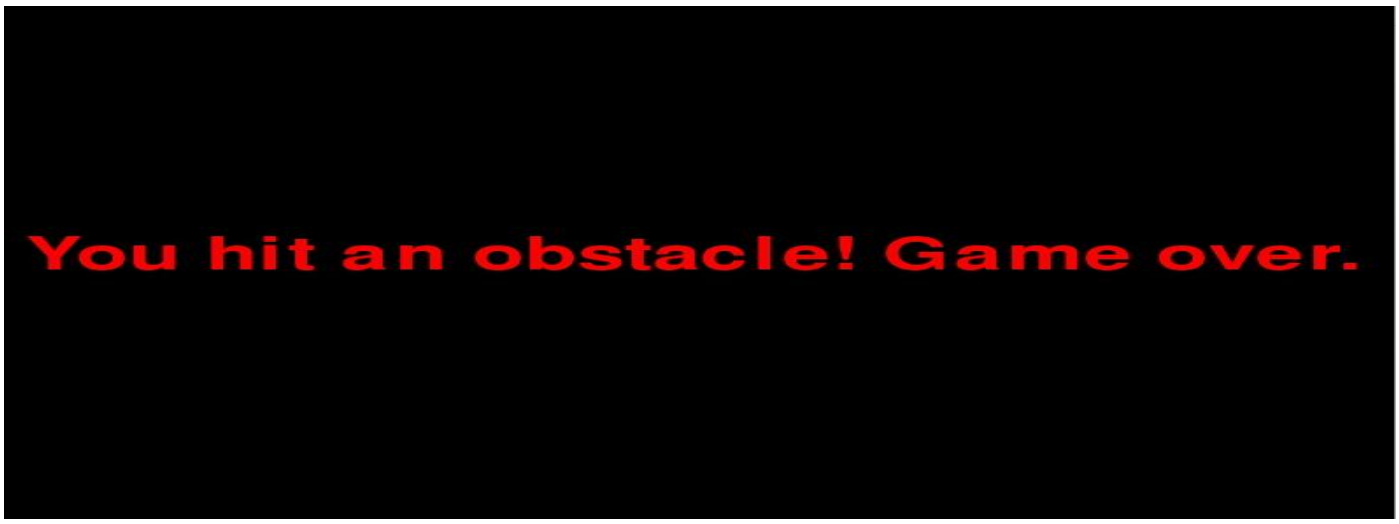
```
    max_width = WIDTH - 20  # Leave some padding
    font_size = 74
    font = pygame.font.Font(None, font_size)
    text = font.render(message, True, color)

    # Reduce font size if text is too wide
    while text.get_width() > max_width:
        font_size -= 2
        font = pygame.font.Font(None, font_size)
        text = font.render(message, True, color)

    text_rect = text.get_rect(center=(WIDTH // 2, HEIGHT // 2))
    screen.fill(BLACK)
    screen.blit(text, text_rect)
    pygame.display.flip()
    pygame.time.delay(2000)

# Function to advance to the next level
def next_level():
    global current_level, obstacles, treasure, player_x, player_y
    if current_level == "Jungle Explorer":
        current_level = "Desert Dare"
    elif current_level == "Desert Dare":
        current_level = "Mountain Madness"
    else:
        display_text("Congratulations! You completed all levels!", ORANGE)
        pygame.quit()
        exit()

    # Reset player position and generate new grid
```

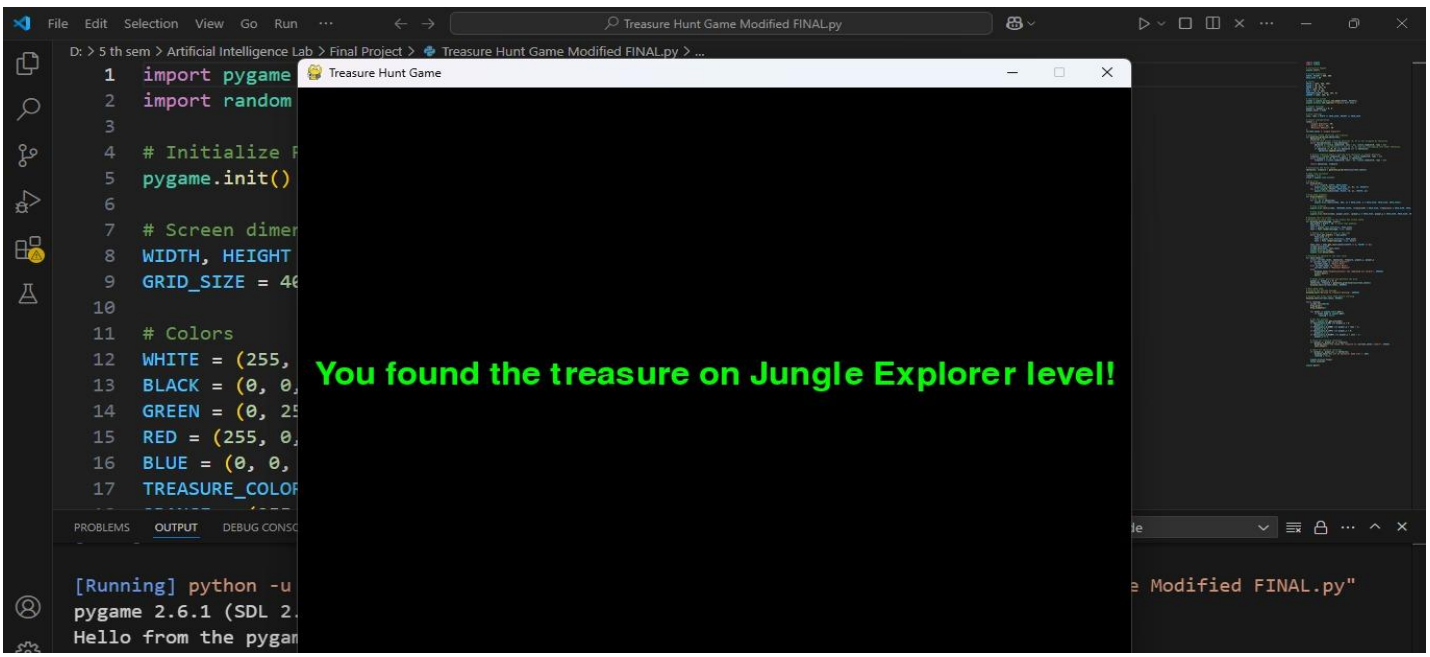
```
    player_x, player_y = 0, 0
    obstacles, treasure = generate_grid(levels[current_level])
    display_text(current_level, ORANGE)

# Main game loop
# Display the welcome message
display_text("Welcome to Treasure Hunting", ORANGE)

# Display the first level name before starting
display_text(current_level, ORANGE)

while running:
    screen.fill(WHITE)
    draw_grid()
    draw_elements()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Get key presses
    keys = pygame.key.get_pressed()
    if keys[pygame.K_UP] and player_y > 0:
        player_y -= 1
    if keys[pygame.K_DOWN] and player_y < rows - 1:
        player_y += 1
    if keys[pygame.K_LEFT] and player_x > 0:
        player_x -= 1
    if keys[pygame.K_RIGHT] and player_x < cols - 1:
        player_x += 1
```

```
    # Check for treasure collision
    if (player_x, player_y) == treasure:
        display_text(f"You found the treasure on {current_level} level!", GREEN)
        next_level()

    # Check for obstacle collision
    if (player_x, player_y) in obstacles:
        display_text("You hit an obstacle! Game over.", RED)
        running = False

    pygame.display.flip()
    clock.tick(10)

pygame.quit()
```

Output


Treasure Hunt Game
Welcome to Treasure Hunting


You hit an obstacle! Game over.


Treasure Hunt Game
You found the treasure on Jungle Explorer level!

Level Progression

The game includes a level progression system. As the player progresses, the difficulty increases by adding more obstacles in the next level.

Upon finding the treasure, the player advances to the next level, and the game resets with new obstacles and a new treasure position.

Once all levels are completed, the player is congratulated for finishing the game.

Game Over and Completion

If the player hits an obstacle, the game ends. However, if they successfully find the treasure, they advance to the next level.

Once the player completes all the levels, a congratulatory message is shown, and the game exits gracefully.

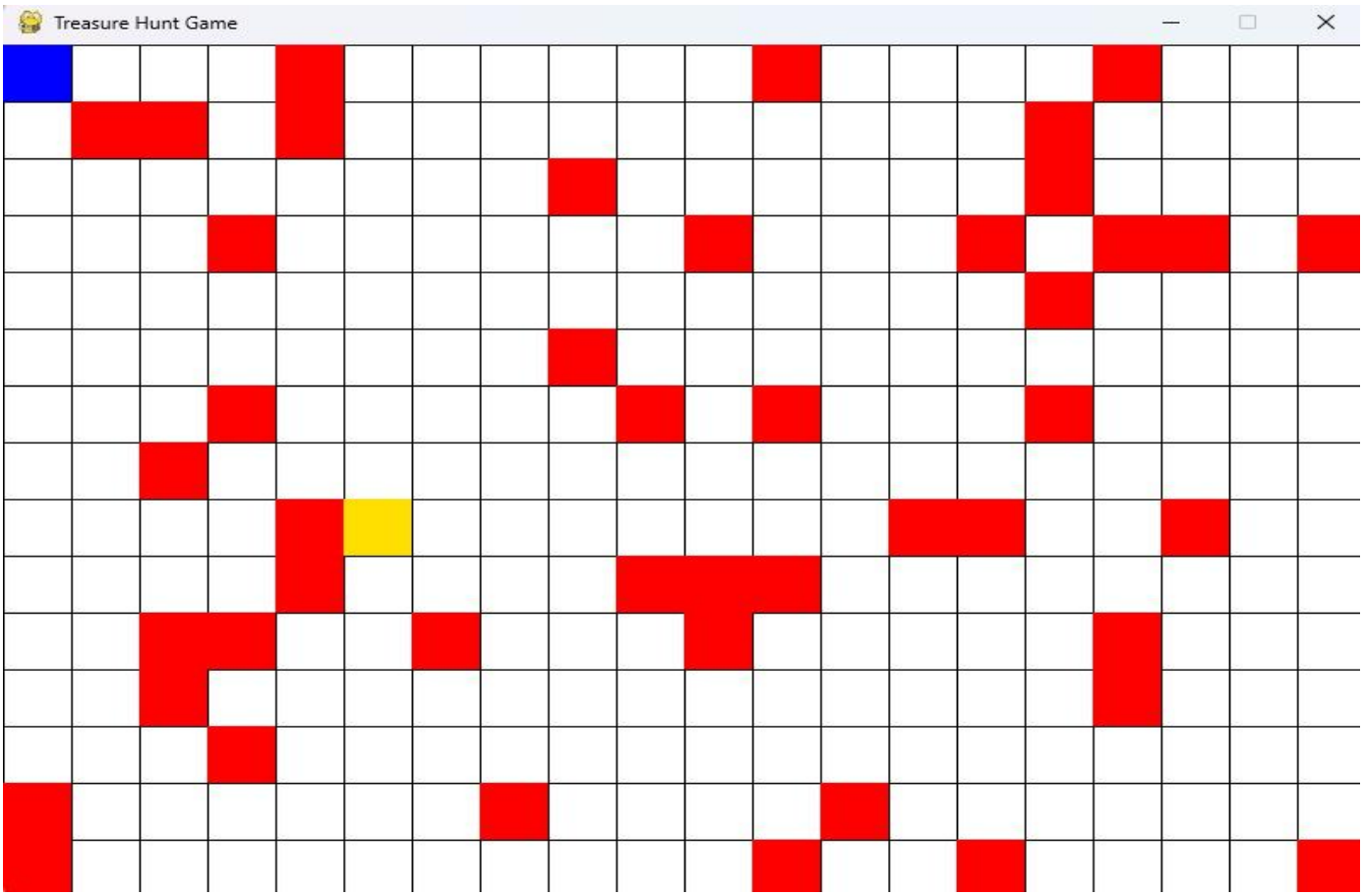
User Interaction and Feedback

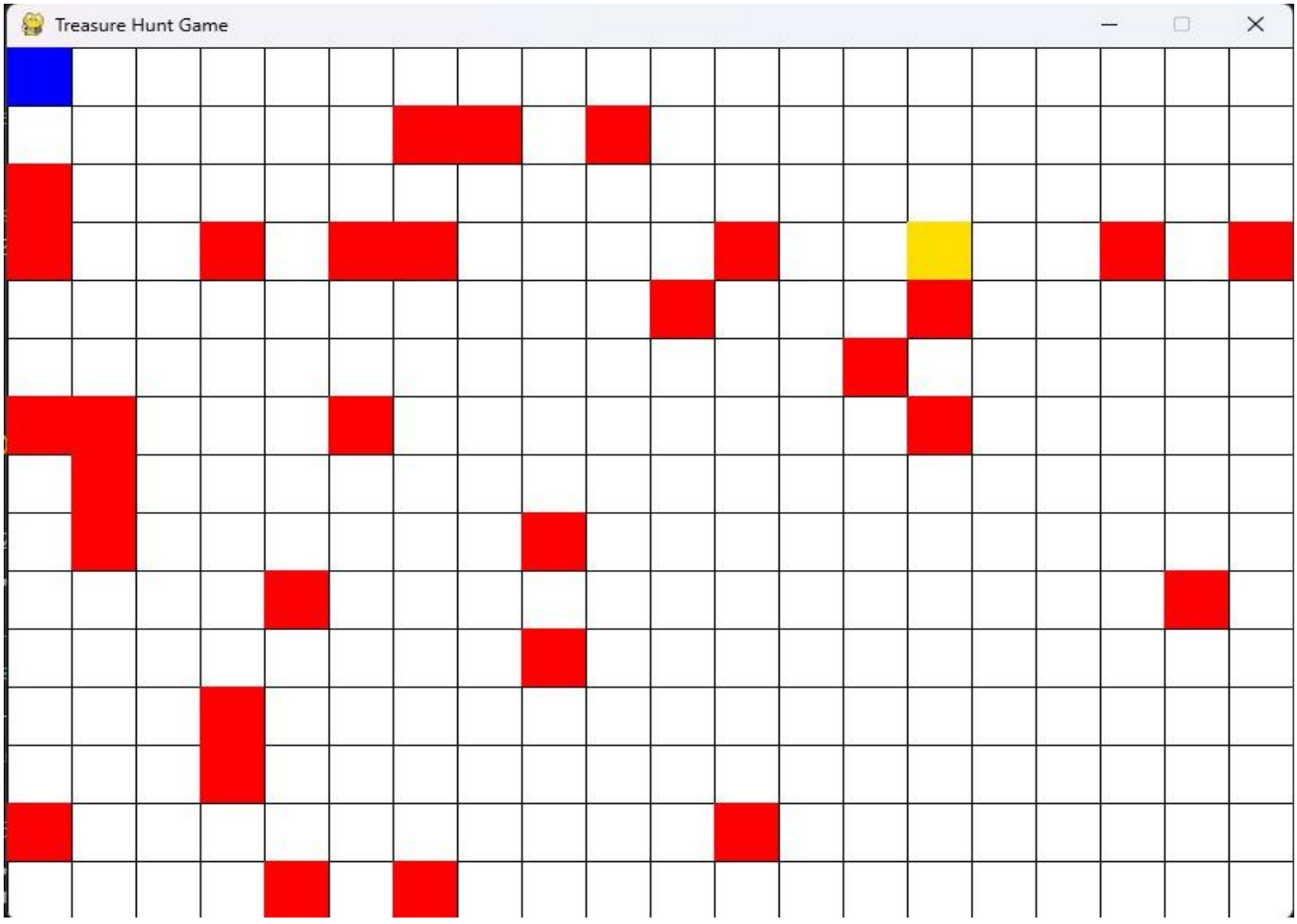
The player receives immediate feedback after each action. If they hit an obstacle, a "Game Over" message is displayed. If they find the treasure, they are congratulated, and the next level begins.

Text is dynamically displayed, and the font size is adjusted to ensure readability regardless of the message's length.

The game includes intuitive controls using the arrow keys for movement and clear visual feedback through colors (e.g., red for obstacles, blue for the player, yellow for treasure).

Game Overview





User Experience

The project demonstrates how to provide clear feedback to the player through text messages, visual elements, and simple controls.

Conclusion

This project successfully implements a basic yet engaging treasure hunt game with simple mechanics and increasing difficulty. By utilizing Pygame's graphics and event-handling capabilities, it offers a fun and interactive experience. The game structure, randomization, and level progression are well-suited for a beginner-level game development project and provide a strong foundation for future enhancements, such as adding sound effects, more levels, or a scoring system.

References

- Pygame – Making game : https://www.pygame.org/docs/tut/MakeGames.html
- Github Link: https://github.com/williamribben10/CSEs
- Treasurer hunt – Wikipedia: https://en.wikipedia.org/wiki/Treasurer_hunt