CMPE160 ASSIGNMENT 3 REPORT

**Introduction:**  In this project I implemented a grid-based shortest path finding algorithm to find the path of a knight trying to collect all the coins on his way. While doing so i also used StdDraw library to visualize those paths. Environment of this game consists of;

1- Knight Figure: Corresponds to knight collecting the coins.

2- Coins: Objectives which knight is trying to collect them all.

2- Grass Tile: Cheapest way to pass over which is 1-5 units.

3- Sand Tile: Expensive way to pass over which is 8-10 units.

4- Impassable Tile: The way which blocks the knight to pass through.

5- Trail Trace: Little dots which corresponds to path of the knight.

The cost of passing those tiles varies between the intervals of them respectively. Knight is moving from one coin to another while keeping the cost of this path cheapest individually until he collects all the possible coins.

**Classes:** I have implemented 8 classes in total for this project.

1- Main Class:

Class managing the overall game flow (reading inputs visualizing etc.). Here is UML diagram:

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

2- Bonus Class:

Class for bonus part implementation of the project inlcuding psvm. Here is UML diagram: metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

3- Tile Class:

Class representing the tiles in the game. It holds information about the tile's position, type (e.g., grass, sand, or obstacle), and its neighbors (adjacent tiles). It also contains properties for pathfinding, such as distance from the start tile, whether it has been visited, and its previous tile (for reconstructing the path).

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

4- FileHandler Class:

Class for implementing file operations including reading input files and writing the output file with given parameters.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

5- Map Class:

This class is responsible for managing the game map, which is made up of tiles. It handles loading map data from files, storing tile information and creating connections between neighboring tiles.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

6- PathFinder Class:

This class is responsible for implementing the pathfinding logic, particularly the Dijkstra's algorithm. It finds the shortest path from a start tile to a destination tile while considering travel costs.

metin, ekran görüntüsü, yazı tipi, çizgi içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

7- GameVisualizer Class:

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.This class handles the visualization aspects of the game, such as drawing the map, animating the knight’s movement, and displaying objectives, paths, and other game elements on the screen using StdDraw.

8- ShortestRoute Class:

Class solving the **Traveling Salesman Problem (TSP)** to find the shortest route that visits all objectives on the map and returns to the start. It uses Dijkstra’s algorithm to precompute distances between objectives, then applies dynamic programming to determine the optimal visiting order. The result is written to a bonus output file, including each movement step, total cost, and total number of steps.

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Algorithm:**

Algorithm Used: Dijkstra’s Algorithm

Purpose:  
To find the shortest path from the starting point to each objective on the map, one at a time.

How it works:

1. The map contains tiles with different travel costs.
2. The algorithm starts at the initial tile and explores neighbors using a priority queue (min-heap), always picking the tile with the lowest total cost so far.
3. It continues exploring until it reaches the target objective.
4. This process is repeated for each objective in the order they appear in the input file.

**Bonus:**

Algorithm Used:

1. Preprocessing: Dijkstra’s Algorithm
2. Main TSP Solver: Held-Karp Dynamic Programming Algorithm (Exact solution for TSP)

How it works:

1. Preprocessing:
   * First, compute and store the shortest paths and distances between all pairs of objectives using Dijkstra's algorithm.
   * This creates a distance matrix and path matrix.
2. Held-Karp Algorithm (Dynamic Programming TSP):
   * Uses a bitmask to represent visited objectives.
   * dp[mask][i] = minimum cost to visit all objectives in mask ending at objective i.
   * Iterates over all subsets of objectives to find the minimum route that visits all objectives and returns to the start.
3. Path Reconstruction:
   * Backtracks through the parent array to get the optimal visiting order.
4. Output:
   * Uses the path matrix to reconstruct actual paths and writes steps to bonus.txt with step count and cost.

**Some figures inside game:**

ekran görüntüsü, kalıp, desen, düzen, kare, metin içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Figure 1: 15x10 Game Screen

ekran görüntüsü, kalıp, desen, düzen, dikdörtgen, kare içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Figure 2: 5x5 Game Screen

ekran görüntüsü, kalıp, desen, düzen içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Figure 3: Bonus Part Game Screen (Shortest Route)

ekran görüntüsü, kalıp, desen, düzen, kare, yeşil içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Figure 4: 20x20 Game Screen