Data Structures
Spring 2023 FAST-NU, Lahore
Assignment 3

CLO 3

Section: 4C

Total Marks: 250

Deadline: 29rd April, 2023 (Saturday).

Question No. 1 (250 marks)

Develop an Adventure Game that involves navigating a maze filled with obstacles and enemies to reach a treasure at the end.

The player starts at a designated location in the forest and must find their way to the end of the forest while solving puzzles, collecting items, and defeating enemies. The player can move in different directions using a stack data structure, with each move costing health points. The player encounters puzzles, with each puzzle requiring a different solution.

The player starts with a certain amount of health points (10) and must make their way through the maze by moving up, down, left, or right. Each move costs the player some health points. The maze is randomly generated each time the game is played.

Along the way, the player will encounter enemies that must be defeated using a **stack-based combat system**. The stack will hold the enemy's attack moves and the player's defensive moves, and the winner will be determined by a set of rules that take into account each move's power and type.

To aid the player in their quest, they will have a **linked list-based inventory system** that can hold items such as health potions, keys, and weapons. The inventory system must be designed to allow for easy addition and removal of items while maintaining a sorted order.

At certain points in the maze, the player will encounter doors that can only be opened with a key. The player must use a **queue-based key management system** to keep track of their keys and open the appropriate doors.

The game should use an **AVL tree** to keep track of the player's high scores and progress.

Subparts

• **Maze Generation:** Develop an algorithm that randomly generates a maze for the game. The maze should be created using a two-dimensional array.

• **Movement:** Implement a function to allow the player to move up, down, left, or right in the maze. The function should update the player's location and decrease their health points.

• **Combat System:** Develop a stack-based combat system that allows the player to battle enemies. The stack should hold the player's defensive moves and the enemy's attack moves, and the winner should be determined by a set of rules that take into account each move's power and type.

• **Linked List Inventory:** Implement a linked list-based inventory system that allows the player to hold items such as health potions, keys, and weapons. The system should be designed to allow for easy addition and removal of items while maintaining a sorted order.

• **Key Management System:** Develop a queue-based key management system that allows the player to keep track of keys found in the maze. The system should allow the player to add and remove keys as needed and should be able to open doors that require keys.

• **Balancing Health, Inventory, and Key Management:** Develop a strategy for balancing the player's health points, inventory, and key management systems. The game should be challenging but not impossible, so the player must make strategic decisions about when to use items, when to engage in combat, and when to save keys for later.

• **AVL Tree High Scores:** Implement an AVL tree to keep track of the player's high scores and progress in the game. The tree should be able to store the player's name, score, and other relevant information. Use file to save the data.

• **Binary Search Tree Navigation:** Implement a binary search tree to store the maze's nodes. The player's movement through the maze should be optimized using the binary search tree to find the shortest path to the treasure. Use file to save the data.

• **Binary Search Tree Leaderboard:** Add a leaderboard feature to the game that uses a binary search tree to store the top scores. The leaderboard should be updated after each game, and the player's score should be compared to the existing scores to determine their rank. Use file to save the data.

• **Binary Search Tree Item Search:** Allow the player to search for specific items in their inventory using a binary search tree. The search should be optimized to minimize the time required to find items, even when the inventory is large.

• **Binary Search Tree Door Locking**: Add locked doors to the maze that can only be opened by solving a puzzle. Use a binary search tree to store the puzzle solutions and check the player's answers to see if they are correct.

Use all the mentioned data structures for the implementation. When the game will start, you need to ask the user for the size of the maze and then create it accordingly. The number of rows and columns will be the same (it will be a square maze). Take the mod of the size with 5, it will be the number of obstacles in the maze, the number of locked doors, and the number of keys. The number of defensive moves and attacking moves will be the number of keys * 2. The level or the intensity of the moves will be diferent for all. Let's say there are 10 moves, their strength or intensity will be set as 1,2,3,4,.....,10. The user will have the defensive moves of all stregths, and same is the case for the enemies in the battle field.

ONLY USE THE DATA STRUCTURES MENTIONED.