Treasure Hunt Game Report

Emir Kaan Oğşarim 2221251042

Ahmed Akif Apari 2221251048

Introduction:

What is the purpose of the project?

Designed as part of the semester project of the Data Structures course, the aim of the project, which is based on the Treasure Hunt Game, includes the manual design and implementation of Single Linked-List and Binary Search Tree data structures. The report aims to provide users and github readers with a comprehensive understanding of the project. It is aimed to enrich the project by using JavaFX technology.

How is the overall game setup?

This project is a game developed to apply the concepts of data structures and algorithms.

The game features a player moving through a map created using a linked list data structure, where each cell has randomly assigned properties.
It consists of two different levels:

- In the first level, the player moves steadily forward.
- In the second level, movements can be both forward and backward depending on the cell effects.

After each move, the player's score is updated, and at the end of the game, scores are saved to a file and stored in a Binary Search Tree (BST) structure.

Data structures used

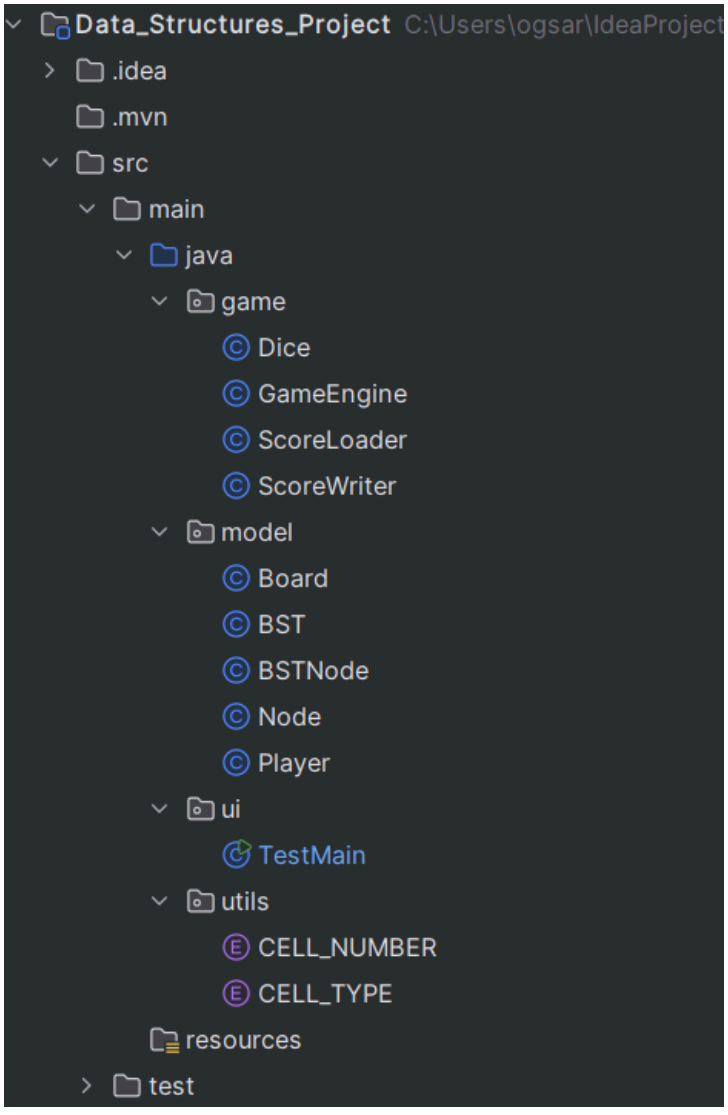
This project aims to implement two fundamental data structures:

- **Linked List:**
  The game map is modeled using a linked list structure. Each cell (node) is linked to the next one (via the next reference), forming a sequential structure with embedded information.

- **Binary Search Tree (BST):**
  At the end of the game, each player's score is inserted into a BST. The BST manages access to scores, sorting, and finding the minimum/maximum score.

Class Structures and Tasks

The project files are designed to be organized and follow the SOLID principles. Each operation is distributed into its representative packages, and the main skeleton is formed through the import of these packages across classes. Below (a.1) are screenshots of the core packages and classes of the project.



a.1

game/Dice.java:

Our class is created with the Singleton design pattern. The uncontrolled creation of the Dice object is restricted with the getInstance method. When Dice is called, that is, when the dice object is created, it has a constructor that will create only 1 Random()

object in memory. Any other object created will use the object previously created with getInstance().

game/GameEngine.java:

The primary purpose of this class is to prevent code clutter caused by a series of operations running in the main method.
It contains two main methods:

◦ startGame():

The startGame() method is the main operation that initiates and manages the game's flow across two levels. It encapsulates the entire gameplay experience for the player, including board setup, player creation, gameplay execution, score recording, and Binary Search Tree (BST) updates.

This method helps keep the main method clean by handling all game logic within a single, organized unit.

Step-by-Step Breakdown

1. **Initialize the Score System (BST):**
   - A new Binary Search Tree (BST) object is created to store scores.
   - (Commented out) Optionally, scores can be loaded from Score.txt using a score loader method.
2. **Start Level 1:**
   - A new Board object is created and initialized for **Level 1** using createBoard(1).
3. **Create and Register the Player:**
   - The system prompts the user to enter their name.
   - A Player object is created with the input name, the Level 1 board, and the level number.
4. **Gameplay – Level 1:**
   - The playUntilEnd(player1) method executes gameplay until the player reaches the end of Level 1.
   - Upon completion, the player's score is:
     - Inserted into the BST.
     - Written to a score file using ScoreWriter.writeSingleScore().
5. **Display Score Summary:**
   - The current **maximum** and **minimum** scores in the BST are retrieved and displayed.
   - An in-order traversal of the BST prints all nodes in sorted order.
6. **Level 1 Completion Message:**
   - A message is printed to show that Level 1 has been completed along with the player's score.
7. **Continue to Level 2?**
   - The player is asked if they want to continue to Level 2.

- Input is captured and evaluated.

If Player Chooses to Continue:

1. **Start Level 2:**
   - A new Board is created for **Level 2** with createBoard(2).
2. **Reuse Player:**
   - A new Player object is instantiated for Level 2 using the same name.
3. **Gameplay – Level 2:**
   - The player plays through Level 2 using playUntilEnd(player2).
   - The score is:
     - Added to the BST.
     - Written to the score file.
4. **Display Final Score Summary:**
   - Maximum and minimum scores are re-displayed.
   - A final in-order BST traversal shows all player scores.
   - A thank-you message is shown after game completion.

If Player Chooses Not to Continue:

- A simple thank-you message is displayed.

playUntilEnd()

The playUntilEnd(Player player) method manages the active gameplay loop for a single level.
It continues to execute the player's turns until the player reaches the end of the board or chooses to quit the game.

This method encapsulates the core turn-based logic, ensuring smooth interaction between the player and the game environment.

Step-by-Step Breakdown

1. **Gameplay Loop:**
   - A while loop runs continuously **as long as the player has not reached the end** of the board.
   - The condition !player.isAtEnd() ensures the loop stops only when the player finishes the level.
2. **Prompt User Action:**
   - Before each move, the program:
     - Displays the player's name.
     - Shows their current position (getCurrentNode().getId()).
     - Prompts the player to press ENTER to roll the dice or type Q to quit.
3. **Handle Quit Command:**
   - If the player types Q (case-insensitive), a message is printed confirming the player has exited.

- The method returns immediately, breaking the loop and ending the current level.

4. **Execute Turn:**
   - If the player chooses to continue (by pressing ENTER), the system:
     - Prints a message indicating the dice roll.
     - Calls player.move() to move the player based on the dice and cell effects.
     - Prints a separator for clarity between turns.

Purpose and Benefits:

- This method keeps the game interactive and player-driven.
- By allowing early exits, it enhances user control.
- It simplifies the main gameplay logic by delegating turn handling to a self-contained method.

game/ScoreLoader.java:

The primary purpose of this class is to load previously recorded player scores from an external file (score.txt) and insert them into a Binary Search Tree (BST).
This allows the game to maintain persistent high score data across multiple sessions, supporting features like ranking, score tracking, and comparisons.

It contains one main method:

loadScores(BST bst):

The loadScores() method is responsible for reading the score file line by line, parsing the data into meaningful components (score, player name, level), and inserting them into the BST. This operation ensures that old scores are preserved and accessible when the game starts.

This method helps separate file input logic from the core gameplay code, promoting modularity and clean architecture.

Step-by-Step Breakdown

Check File Existence:

- A File object is created pointing to score.txt.
- If the file does not exist:
  - A message is printed to inform the user.
  - The method exits early without modifying the BST.

Read and Parse the File:

- The file is opened using a BufferedReader within a try-with-resources block to ensure proper file handling and automatic closure.
- The method reads the file line by line, assuming each line contains a score entry.

Process Each Line:

- Each line is expected to follow a specific format, such as:
  20 (Alice, level 1)
- The line is split using the (character to isolate the score from the rest of the player information.
- The remaining part is further split using) and , to extract:
  - The player's name.
  - The level as a string (e.g., "level 1").

Extract and Convert Data:

- The score string is trimmed and parsed as an integer.
- The player's name is trimmed and extracted directly.
- The level string is cleaned by removing "level " and converted into an integer.

Insert into BST:

- The parsed score, player name, and level are inserted into the given BST object using bst.insert().

Error Handling:

- Any IOException that may occur during reading is caught and printed.
- This ensures the method fails gracefully without crashing the game.

Purpose and Benefits:

- Provides a reliable mechanism for restoring past game results.
- Enhances user experience by preserving historical scores.
- Supports game analytics (min/max score, ranking) by maintaining a populated BST.
- Encourages separation of concerns by isolating file-reading logic from the game engine.

game/ScoreWriter.java:

The primary purpose of this class is to handle writing player scores to an external file (score.txt) to ensure score persistence across game sessions.
This class prevents duplicate entries and supports both appending single scores and (optionally) writing entire BST data to the file.

It contains two main methods (one commented-out optional method and one active method):

writeSingleScore(int score, String user, int level):

The writeSingleScore() method appends a single player score to score.txt if it does not already exist.
This ensures that scores are recorded only once and supports ongoing session updates without overwriting previous data.

Step-by-Step Breakdown

Prepare the Score Line:

- A new string is created representing the player's score in the format:
  score (username, level x)

Check File Existence:

- If score.txt does **not** exist:
  - The file is created.
  - The new score line is directly written using a FileWriter.

Check for Duplicates:

- If the file **does** exist:
  - It is read line by line using a BufferedReader.
  - If the exact score line is already present, the method returns early and skips writing.

Append the New Score:

- If no duplicates are found:
  - The new line is appended to the end of the file using a FileWriter in append mode.

loadExistingScores():

A private utility method used to load all existing lines from score.txt into a list.

Functionality:

- Reads each line from the file.
- Stores each existing score as a string in a List<String>.

Purpose and Benefits:

- Maintains a consistent, persistent record of player performance.
- Prevents duplicate score entries through careful checking.
- Supports both incremental score updates and full overwrites via two different strategies (single or full write).

- Keeps file handling logic clean and encapsulated away from game logic.

model/Board.java:

The primary purpose of this class is to construct the game board using a singly linked list (Node objects) and to assign randomized cell types and movement effects based on the selected game level.

The board's dynamic, linked structure allows for flexible cell generation and individual cell behavior.

createBoard(int level):

The createBoard() method is responsible for initializing the game board by creating a sequence of linked nodes (cells). Each node is assigned a random type based on the difficulty level and may include special effects.

Step-by-Step Breakdown

User Input – Board Size:

- Prompts the player to input a number of cells.
- Ensures the minimum board size is **30**; if not, repeatedly prompts until valid.

Generate Linked List Nodes:

- Iterates from 1 to the selected number of cells.
- For each node:
  - A CELL_TYPE is generated using randomCellType(level).
  - If the cell type is MOVE_EFFECT, a random movement value between -3 and +3 is generated (with 50% chance of being negative).
  - A Node is instantiated with the id, cell type, and optional move effect.
  - The node is appended to the end of the linked list (head → tail).

Track Board Length:

- Each time a node is added, a counter (linkedListNodeCounter) is incremented for tracking board size.

randomCellType(int level):

This private method determines the type of each cell by randomly generating a "luck percentage" and using probability ranges tailored to the game level.

Level 1 Probabilities:

- 30% → TREASURE
- 15% → TRAP
- 5% → GRAND_TREASURE
- 50% → EMPTY

Level 2+ Probabilities:

- 25% → TREASURE
- 15% → TRAP
- 10% → GRAND_TREASURE
- 20% → MOVE_EFFECT *(only available in Level 2+)*
- 30% → EMPTY

This makes Level 2 more dynamic and unpredictable, introducing cells that can push the player forward or backward.

printBoardTest():

A helper/debug method used to visually inspect the current state of the board.

Functionality:

- Iterates through the linked list.
- Prints each node's ID and cell type in order.
- Useful for testing the correctness of board generation.

getHead():

Package-private accessor used to retrieve the first node of the linked list (the starting point of the board). This is commonly used by the Player class or game engine to begin traversal from the board's start.

Purpose and Benefits:

- Modularizes and encapsulates the board setup logic for easy maintenance.
- Offers flexibility by supporting variable board sizes and dynamic level-based difficulty.
- Randomization keeps gameplay fresh and replayable.
- Provides a clear separation between board data and game logic.

model/Player.java:

The Player class represents an individual player in the game. It is responsible for tracking the player's position, handling dice-based movement, and applying cell effects (like gaining or losing points or moving additional steps). It also keeps the player's name, score, and current state during gameplay.

Constructor: Player(String playerName, Board board, int level)

- Initializes the player with:
  - A **name**,
  - A reference to the game **board**,
  - The **starting node** set to the head of the board's linked list,
  - An initial **score** of 0,
  - A singleton **Dice** instance for rolling,
  - The **current game level** (1 or 2).

This encapsulates the player's game context at the time of creation.

Key Methods

boolean isAtEnd()

- Checks whether the player is at the **final cell** of the board by verifying if there is no next node.
- Used by the game loop to determine if the level is complete.

void move()

Handles the **entire process** of moving the player during their turn.

Step-by-Step Process:

1. Rolls the dice using Dice.rollDice().
2. Moves forward **N steps**, where N is the dice result.
3. Updates the currentNode reference accordingly.
4. After movement, it **prints position info** and then **calls applyCellEffect()** to apply the cell's logic (score changes, additional moves, etc.).

Edge Case Handling:

- If the player reaches the end of the board mid-movement, the function terminates early.

void applyCellEffect()

Applies game logic based on the **type of the current cell**. The effect depends on the level.

Level 1 Cell Effects:

- TREASURE → +10 points.
- TRAP → -10 points.
- GRAND_TREASURE → +50 points.
- EMPTY → No effect.

Level 2 Cell Effects (all of Level 1, plus):

- MOVE_EFFECT: Additional logic where the player:
  - Moves forward or backward by moveEffect steps.
  - If moveEffect is **positive**, player moves forward normally (with bounds checking).
  - If moveEffect is **negative**, calculates the **target cell** and resets currentNode to that node (again with safety checks to avoid invalid negative indices).

This effect introduces dynamic, unpredictable behavior for Level 2 gameplay.

int getScore()

Returns the current score of the player.

String getPlayerName()

Returns the player's name.

Node getCurrentNode()

Returns the player's current position on the board (the node reference).

Purpose and Benefits

- Encapsulates all logic and data for a player's session.
- Provides modular turn-based behavior through move() and applyCellEffect().
- Supports flexible level-based gameplay differentiation.
- Ensures that board traversal and scoring are handled seamlessly and consistently.

model/Node.java:

The Node class represents a **single cell** in the game's **linked list-based board**. Each node contains the cell's metadata (such as type, ID, and move effect) and a reference to the **next node**, forming the board structure.

Constructor: Node(int id, CELL_TYPE cellType, int moveEffect)

Initializes a new board cell with:

- id: The **sequential identifier** of the cell (e.g., 1st, 2nd, ...).
- cellType: Enum value representing the type of the cell (e.g., TREASURE, TRAP, etc.).
- moveEffect: Only relevant for MOVE_EFFECT type cells, indicating how many extra steps (positive or negative) to move when landed on.

- next: Initially set to null and must be connected manually using setNextNode().

Key Methods

void setNextNode(Node next)

- Sets the **next node** in the board's linked list.
- Used during board creation to **link one node to the next**.

Node getNextNode()

- Returns the **next node** connected in the list.
- Used by the player or engine to **traverse** the board.

int getId()

- Returns the cell's **unique ID**.
- Used for displaying or tracking the player's position.

CELL_TYPE getCellType()

- Returns the **type of the cell** (e.g., TREASURE, TRAP, EMPTY, etc.).
- Used by Player class to determine what effect to apply when a player lands on the cell.

int getMoveEffect()

- Returns the **move effect value** associated with MOVE_EFFECT type cells.
- Can be positive (move forward) or negative (move backward).
- Ignored for other cell types.

Purpose and Benefits

- Forms the **core data structure** of the board via a singly linked list.
- Encapsulates **both logic and data** related to a single cell.
- Allows easy **expansion**, **customization**, and **traversal** for game logic.

model/BSTNode.java:

The BSTNode class represents a **node in a Binary Search Tree (BST)** used to store player scores. It organizes data based on scores to allow efficient searching, insertion, and traversal for high score operations.

Constructor: BSTNode(int score, String playerName, int level)

Initializes a node with:

- score: The **numeric score** achieved by the player.

- playerName: The **name of the player** associated with this score.
- level: The **game level** in which this score was obtained.
- left, right: Child nodes in the BST, initialized as null.

Fields

Field	Type	Description
score	int	Player's score used as the **sorting key** in the BST
playerName	String	Name of the player
level	int	Level at which the score was recorded
left	BSTNode	Left child (smaller scores)
right	BSTNode	Right child (greater or equal scores)

Getter Methods

- getScore() → Returns the score.
- getPlayerName() → Returns the player's name.
- getLevel() → Returns the level number.

Purpose and Benefits

- Serves as a **storage unit** within the BST used by the game to manage and display high scores.
- Enables **efficient sorting and retrieval** of scores through BST properties.
- Designed for use in conjunction with the BST class that manages the tree structure.

model/BST.java

The BST class represents a **Binary Search Tree** data structure for storing and managing game scores. Each node contains a player's score, name, and level, and nodes are organized based on score values to allow efficient operations like insertion, search, and traversal.

Constructor

public BST()

Initializes an empty BST by setting the root node to null.

Public Methods

void insert(int score, String username, int level)

- Inserts a new score entry into the BST.
- Handles duplicates by inserting equal scores to the **right subtree**.
- Calls a recursive helper method.

void inOrderTraversal(BSTNode root)

- Performs an **in-order traversal** of the tree (left → root → right).
- Prints all stored scores in ascending order.

void searchTraversal(BSTNode root, String username)

- Traverses the tree and prints **all entries** for a given username.
- Uses **in-order** logic to maintain order.

BSTNode getRoot()

- Returns the root of the tree.

BSTNode getMaxNode()

- Returns the **node with the highest score** (rightmost node in the tree).

BSTNode getMinNode()

- Returns the **node with the lowest score** (leftmost node in the tree).

void getAllScores(BSTNode root, StringBuilder scores)

- Performs an in-order traversal and appends each node's score info to a StringBuilder.
- Used for exporting or displaying all scores.

Private Methods

BSTNode insertRecursive(BSTNode root, int score, String username, int level)

- Recursively finds the correct location to insert a new node.
- Maintains BST ordering based on the score value.

ui.TestMain

- Typically used in conjunction with ScoreWriter to save or display player performance.
- Allows features like **leaderboards**, **per-player history**, or **score filtering** by level.
- Ensures **log(n)** average time complexity for insertion and retrieval (in a balanced state).

1. **Scanner Initialization**

- A Scanner is used to take **input from the player** (for menu selections and username queries).

2. **Score System Setup**

- A BST (Binary Search Tree) object is created to store **player scores**.
- ScoreLoader.loadScores() is called to **load historical scores** into the BST. (Presumably from a hardcoded source or file.)

3. **Menu Loop**

- A **do-while loop** allows the user to return to the main menu after completing an action (either playing or viewing scores).
- It checks whether the user wants to **continue or exit**.

4. **Menu Options**

- The user can:
  - **Start the game** by selecting option 1.
  - **Search for scores** by selecting option 2.

Game Start Option

- When the user chooses to play, a GameEngine object is created and the startGame() method is called to begin gameplay.

Scoreboard Search Option

```
System.out.println("Enter an username: ");

String userSearch = sc.nextLine();

bst.searchTraversal(bst.getRoot(), userSearch);
```

- The user inputs a username.
- The BST is searched for that username using an **in-order traversal** (searchTraversal()), and matching scores are printed.

Purpose of This Class

- Acts as a **simple console menu** to access game functions.
- Integrates:
  - **Game logic** via GameEngine
  - **Score management** via BST and ScoreLoader

- Keeps the program modular by not directly handling game logic or score storage in the UI class.

Game Flow

This program implements a **turn-based, level-based board game** using a **Linked List** as the board and a **Binary Search Tree (BST)** for score tracking. The game flow proceeds through the following main stages:

1. **Game Start**

- The user selects or inputs a level (Level 1 or Level 2).
- The user is prompted to enter the **number of cells** for the board (minimum of 30).

2. **Board Initialization**

- The Board.createBoard(int level) method:
  - Creates a singly linked list of Node objects.
  - Each Node is assigned a random CELL_TYPE (e.g., TREASURE, TRAP, EMPTY, MOVE_EFFECT, GRAND_TREASURE) based on level difficulty.
  - If the cell type is MOVE_EFFECT, a random effect (-3 to +3) is generated.

3. **Player Setup**

- A Player object is created with:
  - A name,
  - A reference to the generated board,
  - The selected level.
- The player's starting position is set to the **head node** of the board.
- Score starts at **0**.

4. **Gameplay Loop**

- The game proceeds in **turns**, controlled by a loop (likely from the GUI or main class, not shown here):
- On each turn:
  - The player **rolls a dice** (using a Singleton Dice object).
  - The player **moves forward** by the number of steps indicated by the dice roll.
  - After landing on a new node, the **cell type effect is applied**:
  - **Level 1:**
    - TREASURE: +10 points
    - TRAP: -10 points
    - GRAND_TREASURE: +50 points
    - EMPTY: no change
  - **Level 2 (adds MOVE_EFFECT):**
    - Same as above

    - MOVE_EFFECT: Player is either moved forward or backward again by 1–3 steps based on effect value

5. **End Condition**

- The game ends when:
  - Player.isAtEnd() returns true (the player reaches the end of the board).
- Final score is displayed or logged.

6. **Scoring and Saving**

- The player's name, score, and level are inserted into a **Binary Search Tree (BST)** for:
  - In-order score display,
  - Leaderboard generation,
  - Searching for specific players.

7. **Leaderboard**

- After the game, the BST allows:
  - Printing all scores in order (inOrderTraversal()),
  - Searching by name (searchTraversal()),
  - Displaying the highest or lowest score (getMaxNode(), getMinNode()),
  - Exporting scores to a file using loadScores().

Class Interactions Summary

Component	Purpose	Key Role
Board	Game board with a Linked List of cells	Randomly generates board with effects
Node	Represents a cell on the board	Holds cell type and optional move effect
Player	Game participant	Rolls dice, moves, collects points
Dice	Singleton dice roller	Generates random move values
BST & BSTNode	Binary Search Tree of scores	Stores and displays score history

Conclusion and Evaluation

Conclusion

This board game project successfully demonstrates core object-oriented programming principles through the integration of custom data structures and game logic. By using a **Linked List** for the game board and a **Binary Search Tree (BST)** for storing player scores, the project creates an engaging and scalable game system that dynamically adapts to varying difficulty levels.

Key features include:

- **Randomized board generation** with multiple cell effects (TREASURE, TRAP, etc.)
- A **modular player system** with score tracking and level-based behavior
- A **singleton dice mechanic** to simulate chance
- An organized **score leaderboard system** using BST traversal and search

This implementation showcases the effective use of recursion, abstraction, encapsulation, and single-responsibility design.

Evaluation

Strengths

- **Modular Design**: The system is cleanly split into logical classes (Board, Player, Node, BST, etc.), making it easy to expand or maintain.
- **Level-Based Complexity**: Different cell types and behaviors between levels provide replay value and increasing difficulty.
- **Data Structure Usage**:
  - The **Linked List** represents board traversal in a natural and educational way.
  - The **BST** organizes score data for quick lookup, max/min detection, and leaderboard listing.
- **Code Readability**: Descriptive naming and comments aid in understanding and future development.

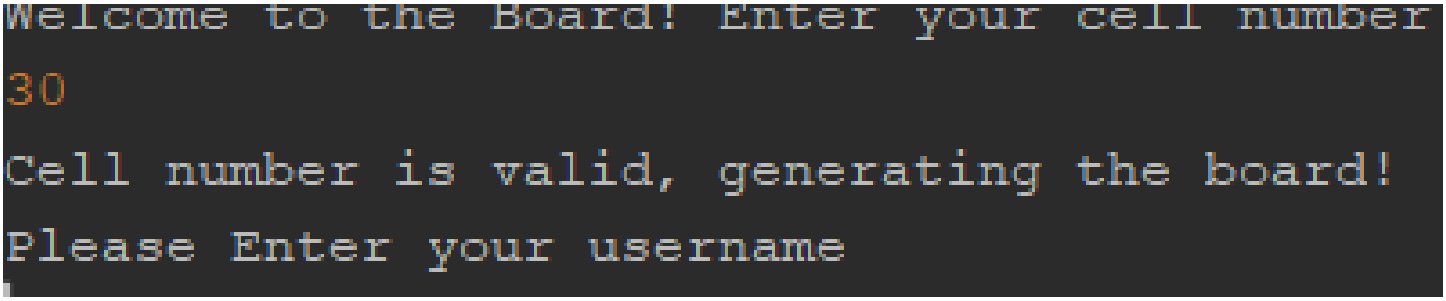
Limitations

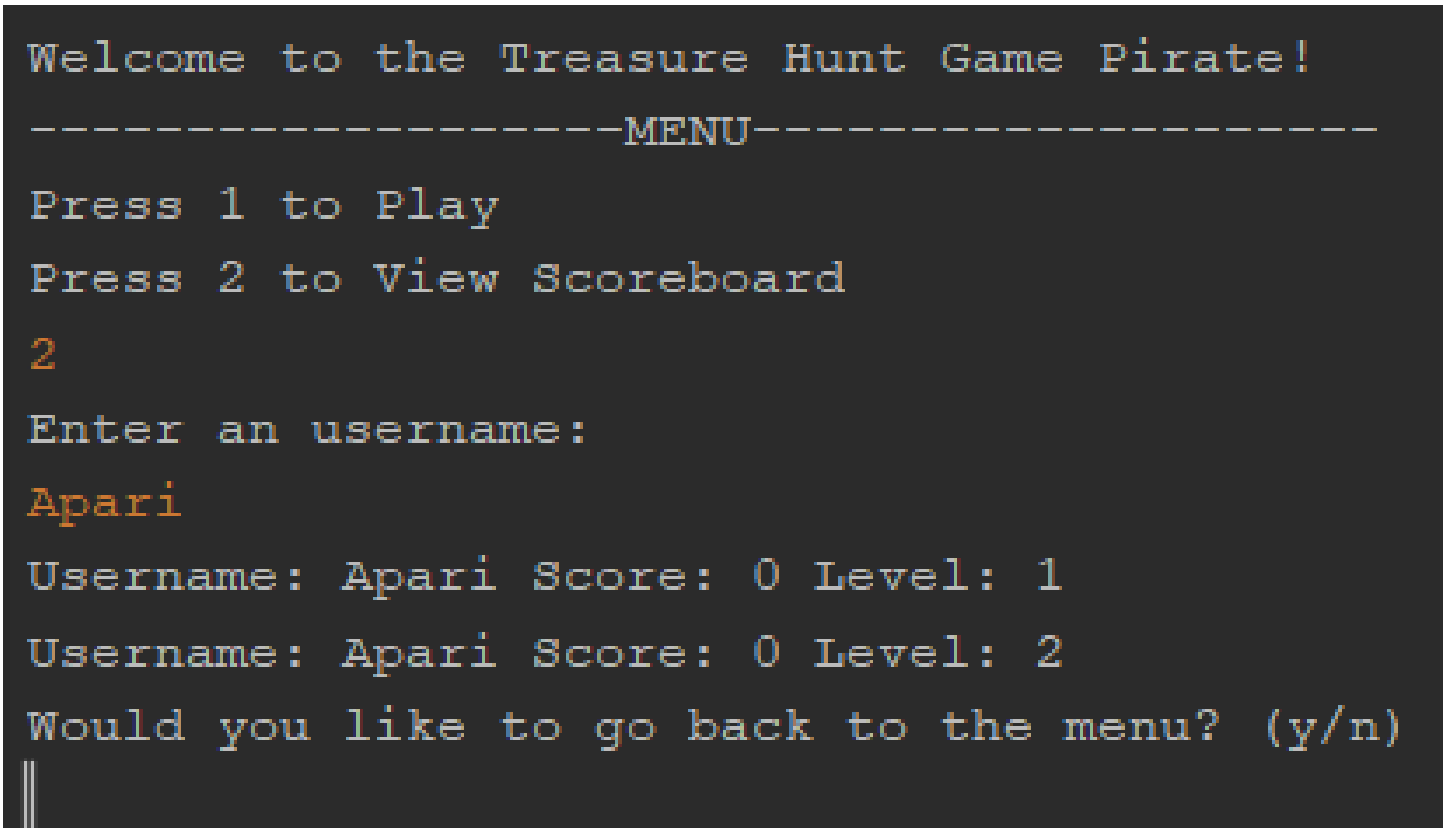
- **Console Dependency**: Board size input and output are currently console-based, limiting user experience. A GUI would greatly improve interaction.
- **Single Player**: The current implementation handles only one player at a time. Extending it to a multiplayer format could improve competitiveness.
- **No Persistence**: Scores are stored only during runtime. Implementing file/database saving would allow for long-term statistics.
- **Limited Game End Feedback**: The end-game state (win/lose, final message) could be more engaging or celebratory for players.

Opportunities for Improvement

- Add a **graphical user interface (GUI)** using Java Swing or JavaFX for better visual feedback and usability.
- Introduce **multiplayer support** with turn management.
- Save game progress and scores to a file (e.g., .txt or .csv) or use serialization.
- Include a **high score screen** and difficulty customization options.
- Add unit tests for robustness and future code changes.

Screenshots of the Game





```
Apari
Apari - Current position: 1th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 6 !
Apari moves 6 steps
Apari is at the 2. cell (TRAP)
                |
                V
Apari is at the 3. cell (EMPTY)
                |
                V
Apari is at the 4. cell (EMPTY)
                |
                V
Apari is at the 5. cell (TREASURE)
                |
                V
Apari is at the 6. cell (TREASURE)
                |
                V
Apari is at the 7. cell (TREASURE)
                |
                V
Apari found a treasure! +10 points. Current score: 10
-----------------------------------------------
Apari - Current position: 7th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari is rolling the dice..
Rolling the dice!!
It's 6 !
Apari moves 6 steps
Apari is at the 8. cell (EMPTY)
               |
               V
Apari is at the 9. cell (EMPTY)
               |
               V
Apari is at the 10. cell (EMPTY)
               |
               V
Apari is at the 11. cell (TRAP)
               |
               V
Apari is at the 12. cell (TREASURE)
               |
               V
Apari is at the 13. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 10
-----------------------------------------------
Apari - Current position: 13th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 13th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 2 !
Apari moves 2 steps
Apari is at the 14. cell (EMPTY)
              |
              V
Apari is at the 15. cell (TRAP)
              |
              V
Oopsie! Apari fell into a trap! -10 points. Current score: 0
-----------------------------------------------
Apari - Current position: 15th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 15th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 1 !
Apari moves 1 steps
Apari is at the 16. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 16th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 16th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 6 !
Apari moves 6 steps
Apari is at the 17. cell (EMPTY)
               |
               V
Apari is at the 18. cell (EMPTY)
               |
               V
Apari is at the 19. cell (EMPTY)
               |
               V
Apari is at the 20. cell (EMPTY)
               |
               V
Apari is at the 21. cell (TREASURE)
               |
               V
Apari is at the 22. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 22th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 22th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 6 !
Apari moves 6 steps
Apari is at the 23. cell (TREASURE)
               |
               V
Apari is at the 24. cell (TRAP)
               |
               V
Apari is at the 25. cell (EMPTY)
               |
               V
Apari is at the 26. cell (EMPTY)
               |
               V
Apari is at the 27. cell (TRAP)
               |
               V
Apari is at the 28. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 28th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 28th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 2 !
Apari moves 2 steps
Apari is at the 29. cell (EMPTY)
               |
               V
Apari is at the 30. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Current MAX: Apari: 0  -  Current MIN: Apari: 0

=== BST snapshot after Level-1 ===
Username: Apari Score: 0 Level: 1
Apari finished level 1 with score of: 0
Do you want to continue to Level 2? (Yes: 1 / No: 0)
```

```
=== BST snapshot after Level-1 ===
Username: Apari Score: 0 Level: 1
Apari finished level 1 with score of: 0
Do you want to continue to Level 2? (Yes: 1 / No: 0)
1
Level 2 is starting..
Welcome to the Board! Enter your cell number:
30
Cell number is valid, generating the board!
Apari - Current position: 1th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 1th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 4 !
Apari moves 4 steps
Apari is at the 2. cell (MOVE_EFFECT)
               |
               V
Apari is at the 3. cell (TRAP)
               |
               V
Apari is at the 4. cell (TREASURE)
               |
               V
Apari is at the 5. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 5th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 5th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 4 !
Apari moves 4 steps
Apari is at the 6. cell (TRAP)
                |
                V
Apari is at the 7. cell (EMPTY)
                |
                V
Apari is at the 8. cell (EMPTY)
                |
                V
Apari is at the 9. cell (TRAP)
                |
                V
Oopsie! Apari fell into a trap! -10 points. Current score: -10
-----------------------------------------------
Apari - Current position: 9th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 9th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 2 !
Apari moves 2 steps
Apari is at the 10. cell (TREASURE)
                |
                V
Apari is at the 11. cell (TREASURE)
                |
                V
Apari found a treasure! +10 points. Current score: 0
-----------------------------------------------
Apari - Current position: 11th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 11th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 1 !
Apari moves 1 steps
Apari is at the 12. cell (EMPTY)
                |
                V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 12th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 12th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 4 !
Apari moves 4 steps
Apari is at the 13. cell (TREASURE)
               |
               V
Apari is at the 14. cell (MOVE_EFFECT)
               |
               V
Apari is at the 15. cell (TRAP)
               |
               V
Apari is at the 16. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 16th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 16th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 3 !
Apari moves 3 steps
Apari is at the 17. cell (TREASURE)
               |
               V
Apari is at the 18. cell (GRAND_TREASURE)
               |
               V
Apari is at the 19. cell (EMPTY)
               |
               V
Apari is in an empty cell! Current score: 0
-----------------------------------------------
Apari - Current position: 19th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 19th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 4 !
Apari moves 4 steps
Apari is at the 20. cell (TREASURE)
               |
               V
Apari is at the 21. cell (TREASURE)
               |
               V
Apari is at the 22. cell (TREASURE)
               |
               V
Apari is at the 23. cell (MOVE_EFFECT)
               |
               V
Yahoo! Apari got a positive move effect! Move +2 forward from your current cell!
-----------------------------------------------
Apari - Current position: 25th To roll the dice: ENTER, to quit the game: Q:
```

"

```
Apari - Current position: 25th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 5 !
Apari moves 5 steps
Apari is at the 26. cell (MOVE_EFFECT)
               |
               V
Apari is at the 27. cell (TRAP)
               |
               V
Apari is at the 28. cell (EMPTY)
               |
               V
Apari is at the 29. cell (GRAND_TREASURE)
               |
               V
Apari is at the 30. cell (MOVE_EFFECT)
               |
               V
Apari got a negative move effect! Moving to cell 29
Apari moved backward to cell 29
-----------------------------------------------
Apari - Current position: 29th To roll the dice: ENTER, to quit the game: Q:
```

```
Apari - Current position: 29th To roll the dice: ENTER, to quit the game: Q:
Apari is rolling the dice..
Rolling the dice!!
It's 2 !
Apari moves 2 steps
Apari is at the 30. cell (MOVE_EFFECT)
              |
              V
Apari is in the ending point!
-----------------------------------------------
Current MAX: Apari: 0  -  Current MIN: Apari: 0

=== BST snapshot after Level-2 ===
Username: Apari Score: 0 Level: 1
Username: Apari Score: 0 Level: 2
Apari finished level 2 with score of: 0
Thanks for completing the game!
Would you like to go back to the menu? (y/n)
```

```
Welcome to the Treasure Hunt Game Pirate!
--------------------MENU----------------------
Press 1 to Play
Press 2 to View Scoreboard
2
Enter an username:
Apari
Username: Apari Score: 0 Level: 1
Username: Apari Score: 0 Level: 2
Would you like to go back to the menu? (y/n)
```