Treasure Hunt Adventure Game
Design Document

Author: Margaux Cruzan

Document Revision History

Date	Version	Description of changes	Author
4/18/2025	0.0	Initial draft	Bashira Akter Anima
4/28/2025	1	First draft	Margaux Cruzan

1. Introduction

The programing problem we are trying to solve is how we can create a text-based game in which a player moves around a dynamically updating board and tries to avoid obstacles (traps and opponents) while collecting treasure to win. The intended audience of this document is people with intermediate knowledge of C++ learning about object-oriented programming, specifically virtual inheritance, templating, aggregation, and overloading. With this project, we are trying to solve how we can use inherited classes and polymorphism in our created classes to run our game in the most efficient way and how we can move the player around the board as well as interact with objects as the board dynamically updates.

1.1 Design Objectives

The functionality of the game that should be hidden from the end user is the setup of the board (the actual board including the location of each trap and treasure), the opponent's random movements and the random generation of the board. The end user does, however, needs to be able to access the displayed board (hiding the location of the objects) , the status of the game (if the player has won or lost), the player's stats after each turn (health lost or treasure found). The general features of the design will include the needed classes (for the board, the player, and what the player will interact with, these are described in more detail in section 3), the interactions of these classes including how the player will interact with objects, how the board will handle generating itself and placing the objects, and the user interface design of the game (the board display, the prompts for the user's input, and the output messeges).

2. Design Overview

2.1 Constraints and Assumptions

In order for the program to work as intended, the board must update after each turn, win and loss conditions must be checked after each turn and object placement and player movements must happen within the boards of the board. The constraints given in the assignment instructions are that treasures must not be surrounded by traps, vectors are not allowed, at least 4 classes must be used, 1 operator overload must be used, 1

template function must be used, and at least 1 abstract base class must be used. While designing my program, I ran into the issue of making sure the player objects cannot move out of bounds, as well as making sure that while randomly generating the board, objects are not placed on top of each other (1 object per space).

3. Structural Design

3.1 Design Explanation and Rationale

I chose to design my program using 6 classes,

-Board
-Trap
-Treasure
-Player (Abstract Base Class)
-User (Child of Player)
-Opponent (Child of Player)

These classes interact in various ways. The Board class holds the information of objects on the 5x5 grid. It is composed by Trap and Treasure objects, as these objects cannot exist without the existence of the board. The Player class holds information of players in the game. It is an abstract base class for the User and Opponent classes.

This is because both user and opponent share similar attributes and methods and share similarities in their movement behavior. Player and Board share an aggregated relationship. This is because the board contains player, but the board doesn't necessarily need to exist for player to exist.

3.2 Class Diagram

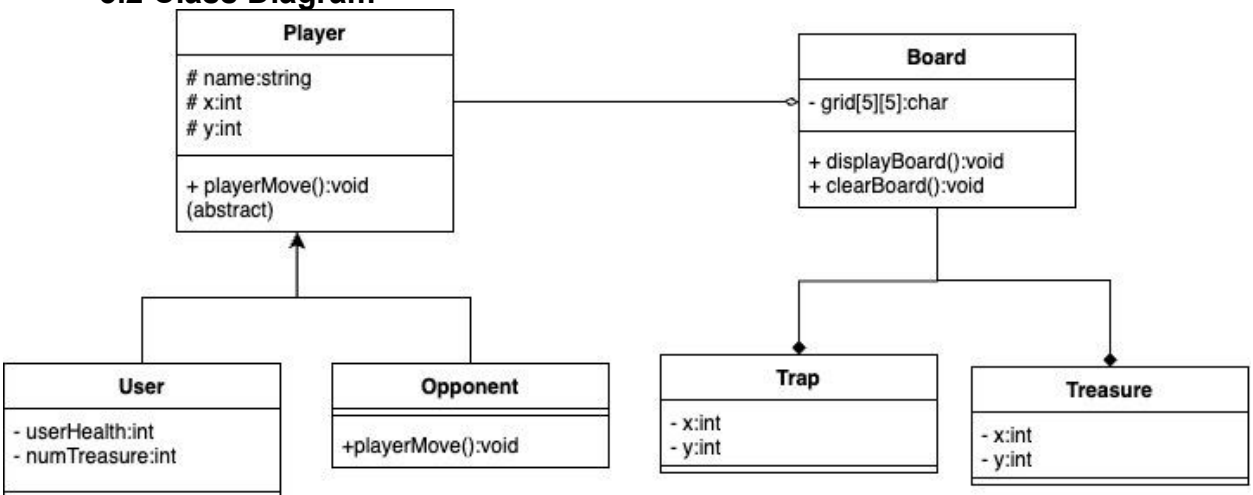

Figure 1: Class diagram for Treasure Hunt Game. Player is an Abstract Base Class and is inherited by the User and Opponent class. Board composes the Trap and Treasure classes, and Board aggregates player.

3.3 Class Descriptions

3.3.1 Class: Player

- Purpose: Class that defines the common attributes and behaviors of the Player and Opponents classes.

- Constraints: Play is an abstract base class, so a Player object cannot be explicitly created. It must also have at least one purely virtual method that is overridden by its child classes.

3.3.1 Attribute Descriptions

1. Attribute: name

Type: string
Description: Stores the name of the player.
Constraints: Initialized in child classes.

2. Attribute: x

Type: int
Description: Stores player's x coordinate.
Constraints: Must be within the bounds of the board.

3. Attribute: y

Description: Stores player's y coordinate.
Constraints: Must be within the bounds of the board.

3.3.1 Method Descriptions

1. Method: playerMove
   Return type: Void
   Parameters: none
   Return value: none
   Pre-condition: Implemented in child classes.

Post-condition: Player's position is changed.
Attributes used: x and y
Description: Defines how the player moves. Each child class will implement differently
Methods called: none

3.3.2 Class: User

- Purpose: Class that defines the user, who moves around the board using WASD to and can collect treasure or lose health from obstacles
- Constraints: Inherits from Player and must implement playerMove.

3.3.2 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Stores the name of the user.
   Constraints: Inherited from Player.
2. Attribute: x
   Type: int
   Description: Stores user's x coordinate.
   Constraints: Inherited from Player.
3. Attribute: y
   Description: Stores user's y coordinate.
   Constraints: Inherited from Player.
4. Attribute: userHealth

Type: int
Description: stores the user's health
Constraints: user loses if 0

5. Attribute: numTreasures

Type: int
Description: stores the user's number of treasures

Constraints: user wins if 3

3.3.2 Method Descriptions

1. Method: playerMove
   Return type: Void
   Parameters: none
   Return value: none
   Pre-condition: none

Post-condition: Player's position is changed.
Attributes used: x and y
Description: Moves player 1 space in direction corresponding to WASD input.
Methods called: none

2. Method: checkWin

Return type: bool
Parameters: none
Return value: true if player won, false if not
Pre-condition: none
Post-condition: none
Attributes used: numTreasures
Description: will check if numTreasures is 3, if yes, returns true and the player won. If not, returns false and game continues
Methods called: none

3. Method: checkLose

Return type: bool
Parameters: none
Return value: true if player lost, false if not
Pre-condition: none
Post-condition: none
Attributes used: userHealth
Description: will check is userHealth is 0, if yes, returns true and the player lost. If not, returns false and game continues.
Methods called: none

3.3.3 Class: Opponent

- Purpose: Class that defines the opponent, who moves randomly around the board at each turn and can remove health from the user.
- Constraints: Inherits from Player and must implement playerMove.

3.3.3 Attribute Descriptions

1. Attribute: name
   Type: string
   Description: Stores the name of the opponent.
   Constraints: Inherited from Player.
2. Attribute: x
   Type: int
   Description: Stores opponent's x coordinate.
   Constraints: Inherited from Player.
3. Attribute: y
   Description: Stores opponent's y coordinate.
   Constraints: Inherited from Player.

3.3.3 Method Descriptions

1. Method: playerMove
   Return type: Void
   Parameters: none
   Return value: none

Pre-condition: none
Post-condition: Opponent's position is changed.
Attributes used: x and y
Description: Moves opponent to a random location on the board after each turn.
Methods called: none

3.3.4 Class: Board

- Purpose: Class that stores information of objects on the board.
- Constraints: Cannot have more than one object per space.

3.3.4 Attribute Descriptions

1. Attribute: grid

Type: 2D character array
Description: Stores the layout of the board and placement of objects on it
Constraints: Must be 5x5

3.3.4 Method Descriptions

1. Method: displayBoard

Return type: void
Parameters: none
Return value: none
Pre-Condition: none
Post-Condition: current board is printed to screen
Attributes used: grid
Description: Iterates through the grid array and each space is printed
Methods called: none

2. Method: clearBoard

Return type: void
Parameters: none
Return value: none
Pre-Condition: none
Post-Condition: Each space on board is empty, indicated by '.'
Attributes used: grid
Description: Iterates through the grid array and replaces each space with a '.' if not already, using checkSpace
Methods called: checkSpace

3. Method: checkSpace

Return type: char
Parameters: int x and int y
Return value: character indicating object
Pre-Condition: space is a '.', 'T', 'X', or 'O'
Post-Condition: none
Attributes used: grid
Description: takes the coordinates of a space and returns what object, if any, is there.
Methods called: none

3.3.5 Class: Trap

- Purpose: Class that represents a trap on the board and removes players health.
- Constraints: Interacted with when player is on its coordinates.

3.3.5 Attribute Descriptions

1. Attribute: x

Type: int
Description: Stores trap’s x coordinate.
Constraints: must be within boards of the board

2. Attribute: y
Description: Stores trap's y coordinate.
Constraints: must be within bounds of the board.

3.3.5 Method Descriptions
None, except for getters, setters, and constructors.

3.3.6 Class: Treasure

- Purpose: Class that represents a treasure on the board and adds to players treasure count when collected
- Constraints: Interacted with when player is on its coordinates.

3.3.6 Attribute Descriptions

3. Attribute: x
Type: int
Description: Stores treasure’s x coordinate.
Constraints: must be within boards of the board
4. Attribute: y
Description: Stores treasure’s y coordinate.
Constraints: must be within bounds of the board.

3.3.6 Method Descriptions
None, except for getters, setters, and constructors.

3.4 Main and Helpers Descriptions

1. Method: displayMenu

Return Type: none
Parameters: none
Return value: none
Class objects used: none
Description: displays the initial main menu displayed to user
Methods Called: none

2. Method: generateGrid

Return Type: none
Parameters: Reference to current board
Return value: none
Class objects used: Board
Description: fills to board with traps, treasure, and empty spaces, represented by cooresponding characters.
Methods Called: checkSpace()

3. Method: isValid

Return Type: bool
Parameters: Reference to current board, coordinates of space.
Return value: returns true if space is available, false if not.
Classes used: Board
Description: Checks one space on the game board to see if it is available to place an object (checks if character there is a ‘.’).
Methods Called: none

4. Method: getUserInput

Return Type: templated date type

Parameters: none

Return value: the user's input

Classes used: none

Description: called when the program needs to get the user's input. It returns the appropriate date type, either int or char, based on user's input.

Methods Called: none

5. Method: removeFromBoard

Return Type: none

Parameters: Reference to current board, coordinates of object

Return value: none

Classes used: Board

Description: when a trap or treasure is revealed to the user, the corresponding character is replaced with '.'

Methods Called: none

6. Method: displayRevealed

Return Type: none

Parameters: Reference to displayed board, coordinates of object

Return value: none

Classes used: Board

Description: when a treasure or trap is interacted with, '.' on the displayed board is replaced with the corresponding character.

Methods Called: none