Homework 1: Treasure Hunter

Due Date

Friday, February 19 @ 2 hours after 11:59pm (2am Saturday morning)

Objectives

- Practice with pointers and references
- Review of objects, inheritance, and logic
- Exposure to operator overloading, ternary operator, sets, and iterators

Resources

Pointers: https://www.youtube.com/watch?v=6pmWojisM_E (https://www.youtube.com/watch?v=6pmWojisM_E)

Treasure Hunter Simulator

Story:

You are an adventurous risk-taking treasure hunter willing to fight to the death for points. You're filled with a blind rage fueled by greed to reach your destination as fast as possible with as many points as possible. So ask yourself, are you willing to embark on the endless treasure hunt?

The Simulator:

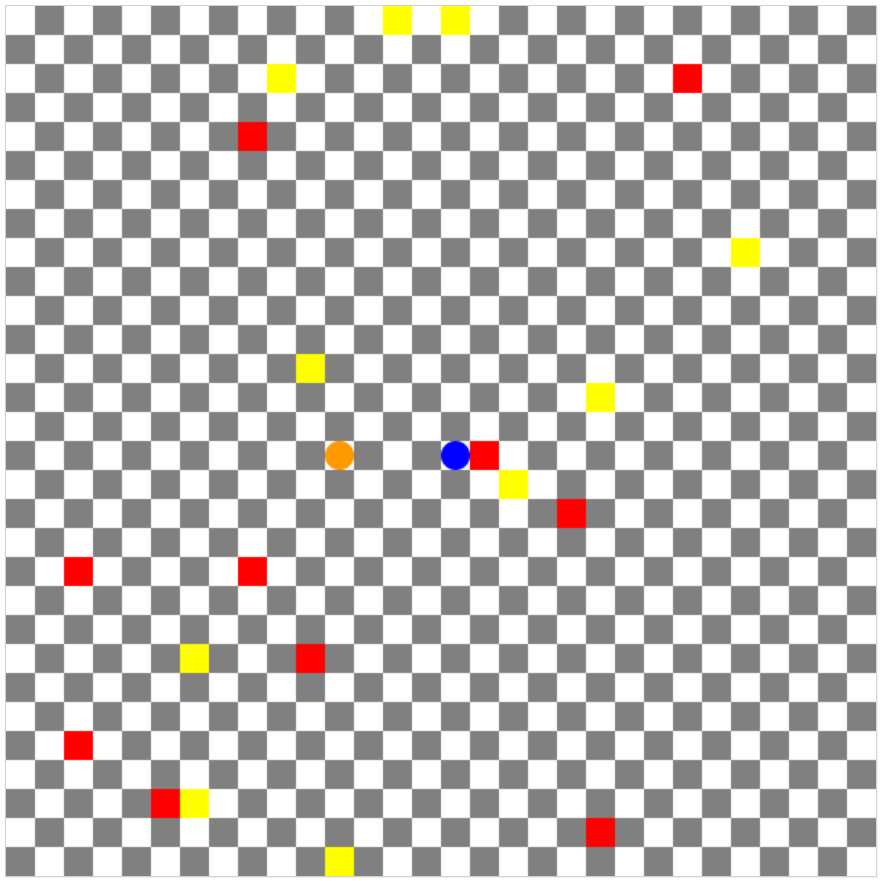
You will be writing a program that simulates the functionality of the game. You will only be writing code for the backend. The simulator should take a string that emulates user input and updates the treasure hunter as needed.

This is a roguelike, single-player game, 2D top-down game. In this game, the player moves on a 2D grid with the goal of accumulating points while navigating towards the goal. There are also squares on the grid that contain enemies and treasures that have properties defined in the Game Mechanics section below.

Here is a prototype to give an idea of what the game would look like to a player. You do not need to make a GUI for this homework. This is only provided to make the rules of game clear:

focus the game (click on it)
"wasd" for movement (this will be "uldr" for the assignment)
k for b
l for a
blue circle = treasure hunter
yellow square = treasure
orange circle = exit
red square = enemy

score: 50



Functionality Requirements

Detecting the Konami Code (1 point)

The user can input a cheat code to be invincible. The cheat code is the Konami code: "up, up, down, down, left, right, left, right, b, a" represented as "uuddlrlrba" for this game.

We will be testing this method to see if it properly detects the Konami Code:

```
bool GameLogic::containsKonamiCode(std::string inputs);
```

This should take in a string, as parameter inputString, and return true if the Konami code "uuddlrlrba" is found within the given inputs. Otherwise, it should return false.

Examples:

```
containsKonamiCode("uuddlrlrba") == true
containsKonamiCode("aaabdlrlruuddlrlrba") == true
containsKonamiCode("aaabdlrlruuddllrlrba") == false
```

Game Mechanics (2 points)

For this part of the assignment we will be testing the functionality of your TreasureHunter class. We will instantiate an object of type TreasureHunter through a constructor that takes a GameBoard, complete with goal, enemy, and treasure locations, and the starting point of the hunter. We will then call the computeScore function with a string representing a sequence of user inputs. Your goal is to incorporate the rules of the game and return the final score a player would receive with the given inputs on the given board.

- Movement
  - The treasure hunter starts out with 50 points.
  - The player moves up, down, left, right, and attacks by inputting "u", "d", "l", "r", and "a" respectively.
  - If the treasure hunter moves closer to the end goal destination in either the *x* or *y* direction, then 1 point is added to the score.
  - If the treasure hunter moves farther away from the end goal in *x* or *y*, then 2 points are taken away from the score.

    - If the player inputs anything other than "udlrab", then the treasure hunter doesn't move and loses 1 point.
    - Moves are made one at a time; no diagonal movements.
    - Only lower-case characters are considered for valid input.
    - If the treasure hunter moves to a space with treasure on it, then 10 points are added to the score and the treasure is removed from the board.
- Fighting
    - If the treasure hunter is on the same space as an enemy, then the player can't move until an "a" has been inputted for an attack. The treasure hunter has a 1-hit KO, so the player can move afterwards.
    - If the Konami Code was entered before fighting the enemy, then the player can instantly KO the enemy and move as if the enemy wasn't there.
- Win Condition
    - The player wins and earns 100 points when the treasure hunter has reached the end goal destination. The score should be returned as the amount of points the treasure hunter has once the end goal is reached.
- Lose Condition
    - The player loses when the treasure hunter has no points left. The score should be returned as 0.

We will be testing this method to see if the correct number of points the Treasure Hunter has after moving with the given inputString:

```
int TreasureHunter::computeScore(std::string inputString);
```

You will get 1 point for a mostly correct solution and 2 points for a completely correct solution.

Efficiency (1 point)

Your game simulation must run efficiently in both time and memory usage when using multiple boards. All methods will be tested extensively with large inputs to determine efficiency. To test efficiency we will be changing the game board many times by calling the following functions:

```
void changeBoard(GameBoard&);
void changeBoard(GameBoard*);
```

Code

Download and complete these C++ files with your solutions.
hw1.zip (homeworkCode/hw1.zip)
Do not edit the GameBoard.h and GameBoard.cpp files.

A run.cpp is included which contains a main method. This is provided to help you understand how to test your code, but this is not an exhaustive test of the functions. You should expand this function with you're own testing for ensure that your code handles every possible input.
Do not submit run.cpp. It will not be graded.

To compile the simulation, type in the terminal: make
To run the simulation, type in the terminal: ./TreasureHunter
You will only see cout statements printed to the terminal.

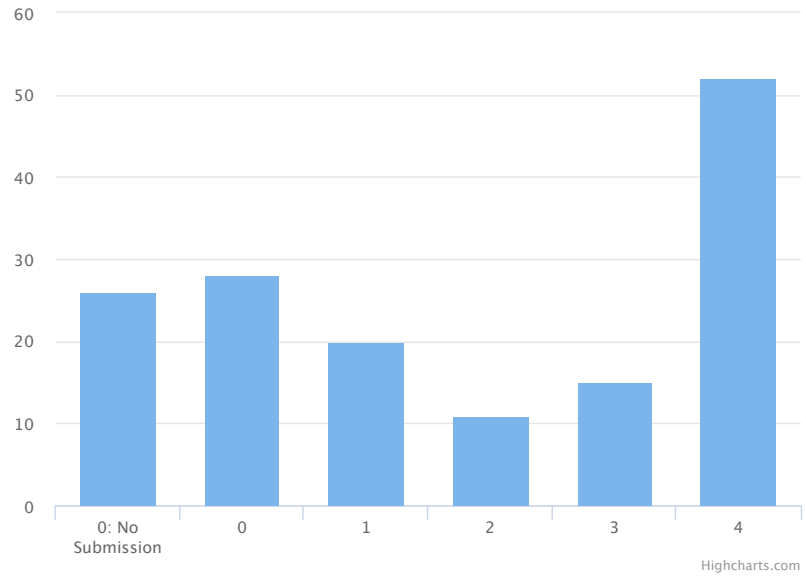
Submission

You should be submitting 4 files: GameLogic.h GameLogic.cpp TreasureHunter.h TreasureHunter.cpp
These file should be submitted individually on the submission website: https://www-student.cse.buffalo.edu/planetexpress/ (https://www-student.cse.buffalo.edu/planetexpress/)

Solution

hw1-solution.zip (homeworkSolutions/hw1-solution.zip)

Results


60
50
40
30
20
10
0
0: No Submission
0
1
2
3
4
Highcharts.com