Programming II

Assignment 0

Review of Programming 1

1 Programming Concepts

- Standard I/O
- Selection Constructs
- Loop Constructs
- References
- Functions
- Arrays

2 Task Summary

This programming task is a review of Programming I concepts up to and including declaring, defining, and invoking functions. In the spirit of this course (and to introduce some basic game programming techniques), we will be implementing a very simple interactive game, which utilizes the Game Loop Pattern. For this assignment you will learn and demonstrate how to:

- Write and read data from the standard input and output streams,
- Write functions using pass-by-value and pass-by-reference
- Utilize a loop construct
- Utilize multiple types of selection construct
- Declare and define variables
- Manipulate arrays
- Implement the Game Loop Pattern

3 Game Loop Architecture

The classic game experience involves a single player interacting with a graphical interface. Typically, the game will process any input, update the state, and then render to the screen. This pattern occurs very frequently in Game Programming and we call this a Game Loop.

The Game Loop is a loop (typically a while loop) and each iteration is called a frame (btw, frames per second is how fast this loop can run). Every frame has to process any input that has been submitted (via mouse or keyboard for example), update the state from this input and render the result.

The state, mentioned previously, is the internal representation of the game and not the graphical representation.

The GUI (Graphical User Interface), mentioned throughout this document, is built by the render stage and is display on the screen.

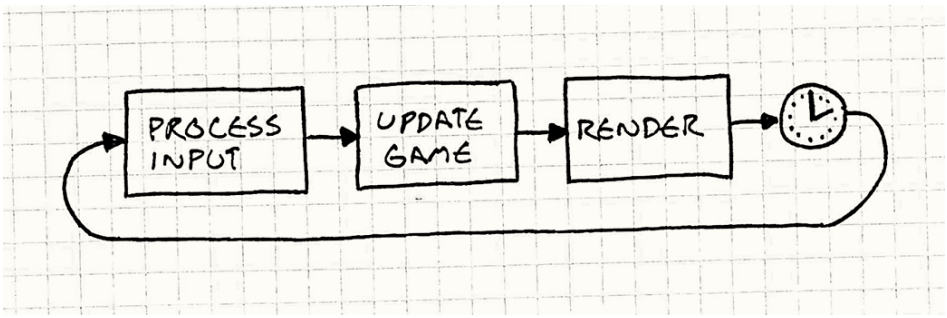

Figure 1: A Simple Game Loop Architecture
Source: http://gameprogrammingpatterns.com/game-loop.html

The Game Loop Pattern divides the code to required to render a frame into manageable steps. In this programming task, the process input stage will accept a character (e.g. 'w') from the player indicating they would like to walk forward. The update game stage will move the player forward and the render stage will show the result.
In this homework you will also be writing code that maintains a text-based GUI and updates a very simple game state.

4 Game Overview

This game contains 36 locations arrayed in a 6x6 grid. The locations are connected along the cardinal directions according to the following graphical layout. The user can move throughout this grid searching for a jewel that is located at a secret location and is hidden from the user.

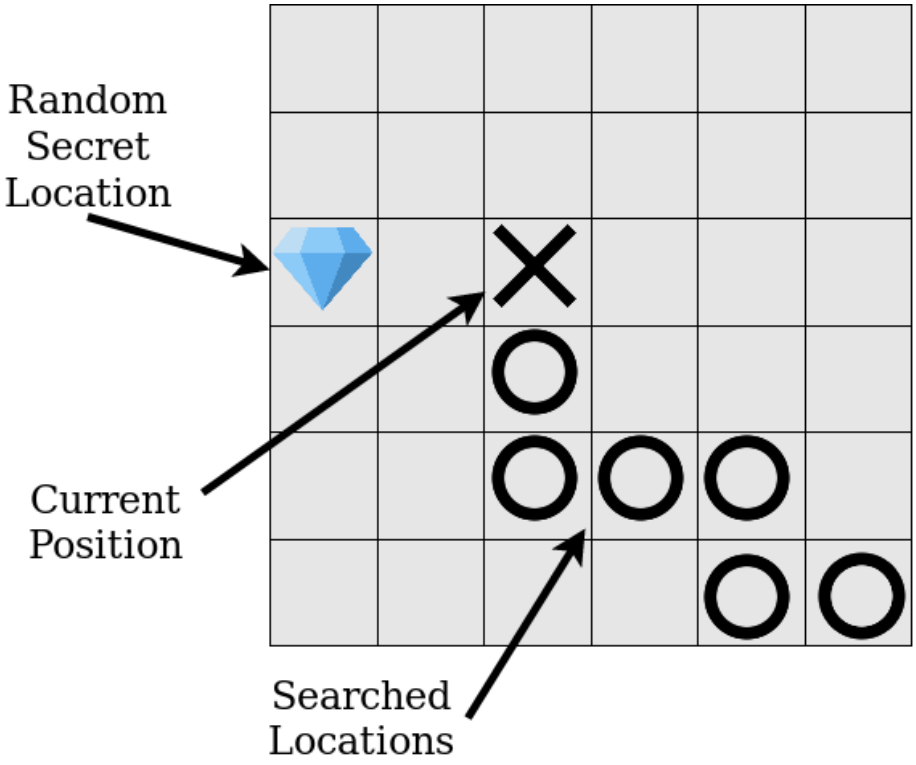

Figure 2: A representation of the board

5 Rules of the Game

1. The player always begins the game in the lower right hand corner of the grid (x = 5, y = 5).
2. The secret location containing a jewel is selected randomly $x \in \{0, 1, 2, 3, 4, 5\}$ and $y \in \{0, 1, 2, 3, 4, 5\}$.
3. The player can change his/her location along either the x or y axis by keyboard input (wasd character style) as shown in the figure.
4. The goal of the game is for the player to find the secret location by searching the grid.
5. When the jewel is found, the player wins.
6. The player cannot move outside the bound of the grid. If the player tries to move beyond the borders of the grid, then the action is ignored.

6 Graphical User Interface

6.1 Startup Screen

This game contains a startup screen supplied by the instructor. This allows the instructor to remember who you are while grading and helps provide a more professional look and feel to your application.

Although the code is provided, look over it and make sure the information is correct.

```
FIND THE SECRET!
Your name here (2020)
CPSC 2377, Programming II, Homework 1
UALR, Computer Science Dept.
INSTRUCTIONS:
Find the secret location to win the game!
        (North)
           w
           |
(West) a --+-- d (East)
           |
           s
        (South)
Press any key to continue . . .
```

Figure 3: Example of the startup screen

6.2 During Gameplay

You are required to draw the GUI which consists of the message "Current Search History" followed by the game board (Figure 2). The game board is encased by a border consisting of dashes, pipes, and plus signs. Inside the game board, an x represents the current position of the player and a o represents the path traveled so far. Following the game board is a message "Select action (wasd):" This indicates to the player the controls that can be used and how to use them.

```
Current Search History
        +-----+
        |     |
        |  Xo |
        |   o |
        |   o |
        |   oo|
        +-----+
Select action (wasd):
```

Figure 4: Example of the screen during gameplay

6.3 After Gameplay

After the player has found the secret location, we render a special message "YOU FOUND THE SECRET LOCATION" at the top of the screen. Once again we render the game board with the same border with o representing the path the player took. However, instead of rendering an x, we render a J representing the jewel. The last message is changed to reflect that there are no futher actions.

```
YOU FOUND THE SECRET LOCATION
        +-----+
        |     |
        | Joo |
        |   o |
        |   o |
        |   oo|
        +-----+
Press any key to continue . . .
```

Figure 5: After the player found the secret location

7 Tasks

The instructor has provided you with the main game loop as well as all the necessary function prototypes and state variables. Your job is to fill in the function definitions to correctly implement the GUI and Game Engine as described by the rules and game play scenarios given above.

Below is a recommended list of tasks to complete the assignment.

7.1 Task 0: Download Base Files

1. Click on the link to the repository in Blackboard and a copy of the repository will be created for you.
2. You will use GitHub Desktop to clone the repository to your computer

7.2 Task 1: Implement Base Game Functionality

Initially, you should not worry about the secret position or adhering to the rules. First layout the code necessary to render the game board and move about it. In this stage, you may consider not to render the full board, just what is necessary to prove you can move. This requires modifying `getAction`, `displayGameBoard`, `changeGameState`, and `initGameState`.

7.3 Task 2: Implement Full Functionality

Once you can move around the grid then you need to modify the `displayGameState`, `changeGameState`, `gameIsDone`, and `displayGameDone` functions to correctly implement the game rules. The display should have a text-based boundary and the current and past positions of the player should be recorded as X and o characters, respectively. You will also successfully detect when the player has won the game and generate a correct displayGameDone function in which the secret location is revealed via the J character.

7.4 Task 3: Submit completed Assignment

Be sure to push your code to your GitHub repository often! Do a push by the due date, and if you make more progress, you can push again for a 10% reduction within two days past the due date. Be sure to submit a ¡5 minute video explaining your code. This is submitted in Blackboard using Kaltura!

8 Grading Rubric

Your job is to design and implement a C++ procedural paradigm program that performs the tasks according to this specification (and any supplemental specifications provided to clarify program tasks). As would be true in a real game development scenario, you will be evaluated primarily on your codes ability to implement all of the requirements correctly. A secondary consideration, but also important, will be the structure, readability, and documentation of your code.

9 How to go about writing this code

1. Read this programming task specification carefully.
2. Download and build the project.
3. Run the base code.
4. Examine the source code. Particularly, study which variables the functions (those required for the programming task) take as arguments and compare them with what these functions are to do.
5. Write down the steps that each function should perform as simple as possible.
6. Use this plan to write the code.
7. Write the code incrementally, compiling often. Save your work!
8. Save your progress often by committing your code and pushing to github.
9. Ask questions early and often!