**Spike:** Task 3.P

**Title:** Gridworld

**Author:** Mitchell Wright, 100595153

**Goals / deliverables:**

Produce a working Gridworld game, according to the provided specification sheet and develop and understanding of a simple game loop (Update/Render).

Items created during task:

* Code, see: \03 - Spike – Gridworld\GridWorld\

**Technologies, Tools, and Resources used:**

List of information needed by someone trying to reproduce this work

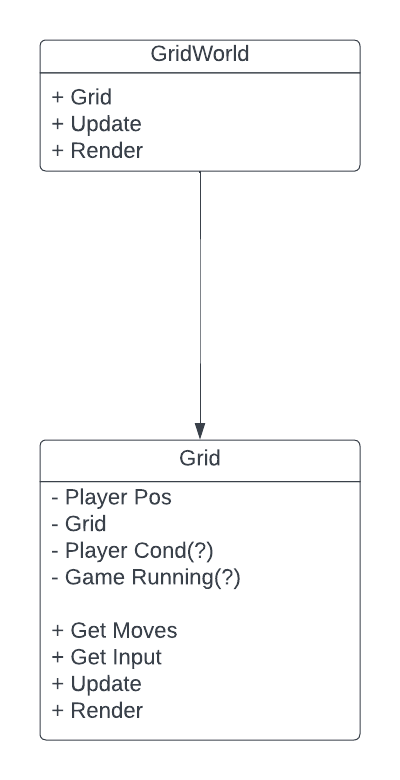
* Visual Studio 2022
* SourceTree
* GitHub
* Lecture 2.1 – Game Loops & Software Architecture

**Tasks undertaken:**

* Design Game Architecture
* Implement Grid
* Get User Input
* Implement Update
* Implement Render
* Commit to Git

**What we found out:**

1. Design Game Architecture:



This is a very simple game, so to match, there is a very simple UML diagram with the basics I’d require to properly implement the game.

1. Implement Grid

Implementation of the actual grid involved creating a simple 2D char array and copying over each line of the grid from the spec sheet.

A screenshot of a computer code

Description automatically generated

As interaction between tiles is not necessary for Gridwold, the array seemed like an easy choice, as neighbours are only +/-1 on either coordinate. While it would have been possible to create a whole tile class, etc, that would have been overkill for this project.

1. Get User Inputs:

Getting user input was a slight pain as I was taking chars as an input type. Using cin or getchar() would lead to issues should multiple characters be typed into the console.

A screen shot of a computer program

Description automatically generated

Instead, I tried \_getwche(). This meant that only a single character could be type at a time, completely bypassing that issue.

Once the input has been taken from the console, it is then converted into a movement direction and used to move the ‘player’.

A computer screen shot of a program code

Description automatically generated

Please note the lack of player class, as since this is not necessary for the game to function, I left it out to save on unnecessary complexity. The player in this case would have only ever been made up of only integer values.

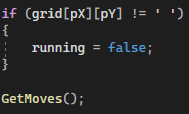
1. Implement Update:

Grid has a public update function, which is looped through in main while the game is running.

A screenshot of a computer program

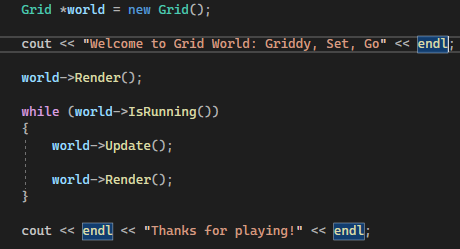
Description automatically generated

Other than input, the update function is relatively sparse. A function that gets a list of possible moves is used, as well as a quick check to see if the game should still be running.



1. Implement Render:

Similarly to update, rendering is also called in the main loop, but is also called once initially outside of the loop.



Rendering within Grid basically just requires the output of the current moves available to the player, a double chevron for style, or if the game is not running anymore, the win/loss message.

A computer screen shot of a program code

Description automatically generated

1. Commit to Git:

Here’s the commit history for this one.

