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

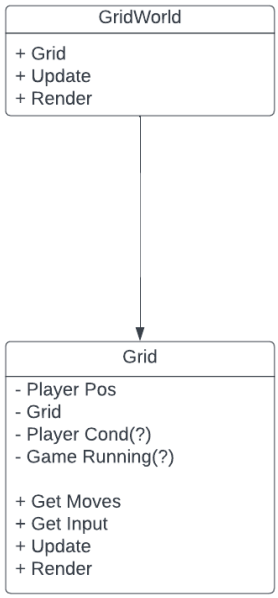
- Visual Studio 2022
- SourceTree
- GitHub
- Lecture 2.1 – Game Loops & Software Architecture

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.

2. 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.

```
class Grid
{
private:
    char grid[8][8] = {
        { '#', '#', '#', '#', '#', '#', '#', '#' },
        { '#', 'G', ' ', 'D', '#', 'D', ' ', '#' },
        { '#', ' ', ' ', ' ', '#', ' ', ' ', '#' },
        { '#', '#', '#', ' ', '#', ' ', 'D', '#' },
        { '#', ' ', ' ', ' ', '#', ' ', ' ', '#' },
        { '#', ' ', '#', '#', '#', '#', ' ', '#' },
        { '#', ' ', ' ', ' ', ' ', ' ', ' ', '#' },
        { '#', '#', ' ', '#', '#', '#', '#', '#' },
    };
```

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.

3. 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.

```
void Grid::Update()
{
    char input = ' ';

    while (input == ' ')
    {
        //Get player input
        input = _getwche();

        cout << endl;

        //Validate input
        input = tolower(input);
```

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’.

```
if (find(moves.begin(), moves.end(), input) != moves.end() || input == 'q')
{
    switch (input)
    {
    case 'n':
        pX--;
        break;
    case 'e':
        pY++;
        break;
    case 's':
        pX++;
        break;
    case 'w':
        pY--;
        break;
    case 'q':
        running = false;
        break;
    default:
        cout << "Error: Invalid input not flagged correctly." << endl;
    }
}
else
{
    input = ' ';
    cout << "Invalid input." << endl;
}
```

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.

4. Implement Update:

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

```
while (world->IsRunning())
{
    world->Update();
```

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.

```
if (grid[pX][pY] != ' ')
{
    running = false;
}

GetMoves();
```

5. Implement Render:

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

```
Grid *world = new Grid();

cout << "Welcome to Grid World: Griddy, Set, Go" << endl;

world->Render();

while (world->IsRunning())
{
    world->Update();

    world->Render();
}

cout << endl << "Thanks for playing!" << endl;
```

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.

```
void Grid::Render()
{
    char dir = ' ';

    if (running)
    {
        cout << "You are able to move ";

        //for loop through possible move directions
        for (int i = 0; i < moves.size(); i++)
        {
            dir = toupper(moves[i]);
            cout << dir;
            if (i + 1 != moves.size())
                cout << ", ";
        }
        cout << ":" << endl;

        cout << ">> ";
    }
    else
    {
        switch (grid[pX][pY])
        {
        case 'G':
            cout << "Congratulations, you win I guess." << endl;
            break;
        case 'D':
            cout << "Oof, you have not won. Damn shame." << endl;
            break;
        default:
            cout << "An error has occured. You have entered into unknown territory and the game cannot continue." << endl;
            break;
        }
    }
}
```

6. Commit to Git:

Here's the commit history for this one.

Description	Date	Author
• main origin/main origin/HEAD **Moved GW to correct folde**	**16 Aug 2023 15:44**	**Mitchell Wright <100595153@student.swin.edu.au>**
Added Spec Sheet. Created VS project. Created basic skeleton for GW.	9 Aug 2023 15:38	Mitchell Wright <100595153@student.swin.edu.au>
Completed report.	9 Aug 2023 15:01	Mitchell Wright <100595153@student.swin.edu.au>
Properly implemented gitignore. Added task 2 files.	9 Aug 2023 14:57	Mitchell Wright <100595153@student.swin.edu.au>
Added new gitignore. Added report for task 2.	9 Aug 2023 14:54	Mitchell Wright <100595153@student.swin.edu.au>
Finalised report	2 Aug 2023 14:51	Mitchell Wright <acmetonto@hotmail.com>
Added T1 report	2 Aug 2023 14:44	Mitchell Wright <acmetonto@hotmail.com>
Altered README.md	2 Aug 2023 14:42	Mitchell Wright <acmetonto@hotmail.com>
Initial commit. Added file structure skeleton	2 Aug 2023 14:29	Mitchell Wright <acmetonto@hotmail.com>
Initial commit	2 Aug 2023 14:20	100595153 <141204538+100595153@users.noreply.github.com>