Spike Summary Report

Spike: Task_03
Title: Gridworld
Author: Thomas Horsley, 103071494

Goals & Deliverables

Aim: The overall goal of the spike was to develop an understanding surrounding game loops, update and render functionality and game data manipulation.

Deliverables:

- Code satisfying the aim
- Pre-programming documentation (see below)
- Spike Summary Report

🧠

Spike Summary Report

Technology, Tools and Resources

Tech and Tools

💡 The project was scripted in C++17 using the VSCode IDE version 1.76.
UML's and charts are made with *www.Lucidchart.com*

Optionally (though recommended), source control is handled using Git.

VSCode Plugins/Extensions

- C/C++
  Author: Microsoft
  Version: v2023.4.1
- Colorful Comments (I always recommend)
  Author: Parth Rastogi
  Version: 1.0
- Code Runner
  Author: Jun Han
  Version: v0.12.0

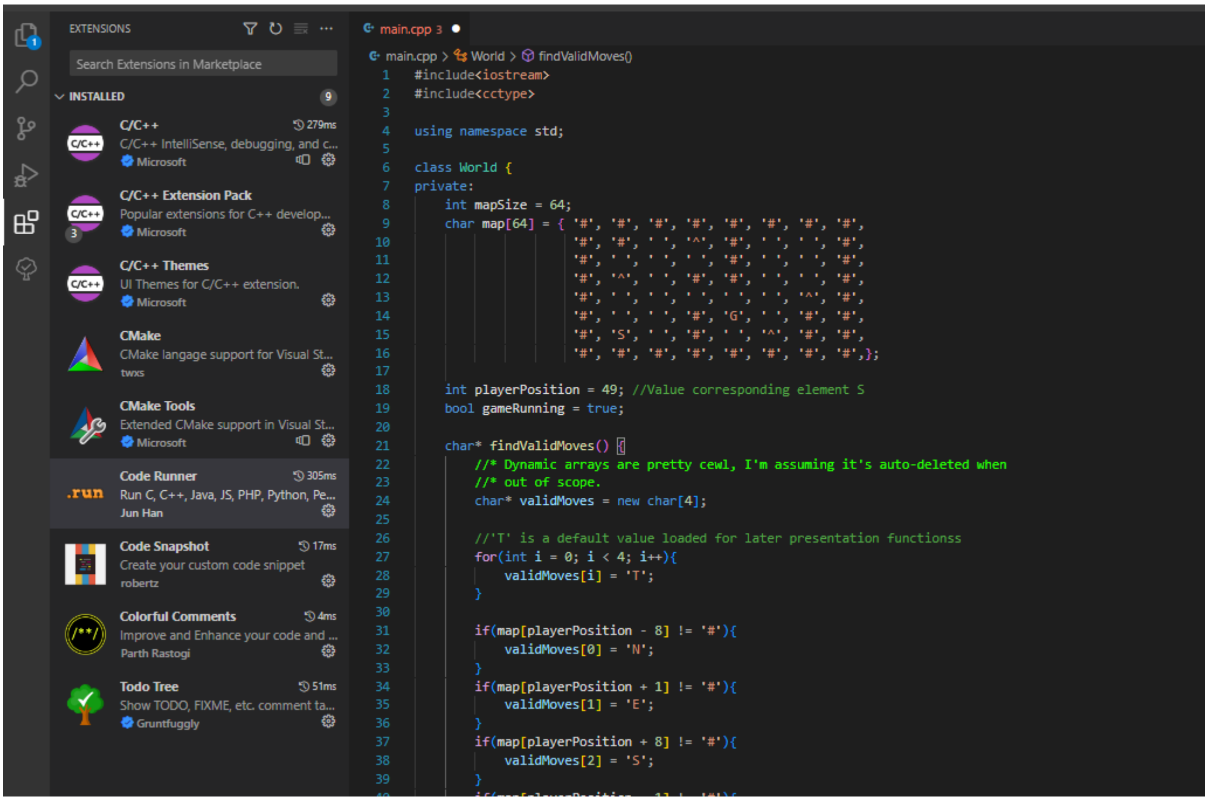
Resources

- Echo360 Lectures "*Topic 1.2 - How to C++"*
- Echo360 Lectures "*Topic 2.1 - Game loops & Software Architecture of Games"*
- C++ POINTERS - How to create/change arrays at runtime?
  See: https://www.youtube.com/watch?v=axsplPtoQF0&list=LL&index=1&t=409s

Tasks Undertaken

Installation

1. Download and Install VSCode v1.79 (or above)
   a. https://code.visualstudio.com/
2. Download and Install G++ using MINGW
   a. https://sourceforge.net/projects/mingw/
3. Install the previously mentioned extensions.


EXTENSIONS
Search Extensions in Marketplace
INSTALLED 9
C/C++ 279ms
C/C++ IntelliSense, debugging, and c...
Microsoft
C/C++ Extension Pack
Popular extensions for C++ develop...
Microsoft
C/C++ Themes
UI Themes for C/C++ extension.
Microsoft
CMake
CMake langage support for Visual St...
twxs
CMake Tools
Extended CMake support in Visual St...
Microsoft
Code Runner 305ms
Run C, C++, Java, JS, PHP, Python, Pe...
Jun Han
Code Snapshot 17ms
Create your custom code snippet
robertz
Colorful Comments 4ms
Improve and Enhance your code and ...
Parth Rastogi
Todo Tree 51ms
Show TODO, FIXME, etc. comment ta...
Gruntfuggly
main.cpp 3
main.cpp > World > findValidMoves()

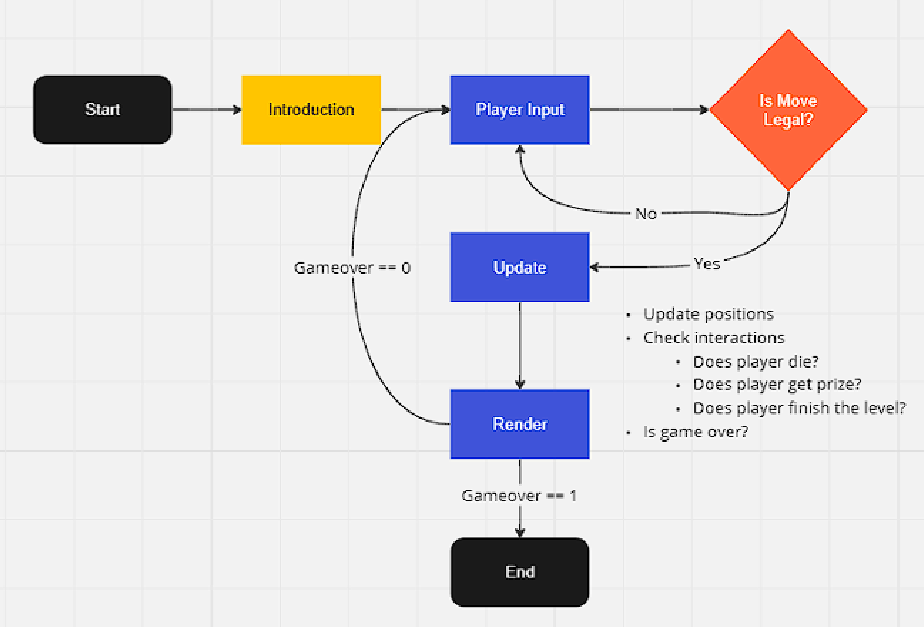
```cpp
#include<iostream>
#include<cctype>

using namespace std;

class World {
private:
    int mapSize = 64;
    char map[64] = { '#', '#', '#', '#', '#', '#', '#', '#',
                     '#', '#', ' ', '^', '#', ' ', ' ', '#',
                     '#', ' ', ' ', ' ', '#', ' ', ' ', '#',
                     '#', '^', ' ', '#', '#', ' ', ' ', '#',
                     '#', ' ', ' ', ' ', ' ', ' ', '^', '#',
                     '#', ' ', ' ', '#', 'G', ' ', '#', '#',
                     '#', 'S', ' ', '#', ' ', '^', '#', '#',
                     '#', '#', '#', '#', '#', '#', '#', '#',};

    int playerPosition = 49; //Value corresponding element S
    bool gameRunning = true;

    char* findValidMoves() {
        //* Dynamic arrays are pretty cewl, I'm assuming it's auto-deleted when
        //* out of scope.
        char* validMoves = new char[4];

        //'T' is a default value loaded for later presentation functionss
        for(int i = 0; i < 4; i++){
            validMoves[i] = 'T';
        }

        if(map[playerPosition - 8] != '#'){
            validMoves[0] = 'N';
        }
        if(map[playerPosition + 1] != '#'){
            validMoves[1] = 'E';
        }
        if(map[playerPosition + 8] != '#'){
            validMoves[2] = 'S';
        }
```

Spike Summary Report

Planning & Implementation

Game Loop Flow Diagram

1. For this I find a great tool is www.Lucidchart.com



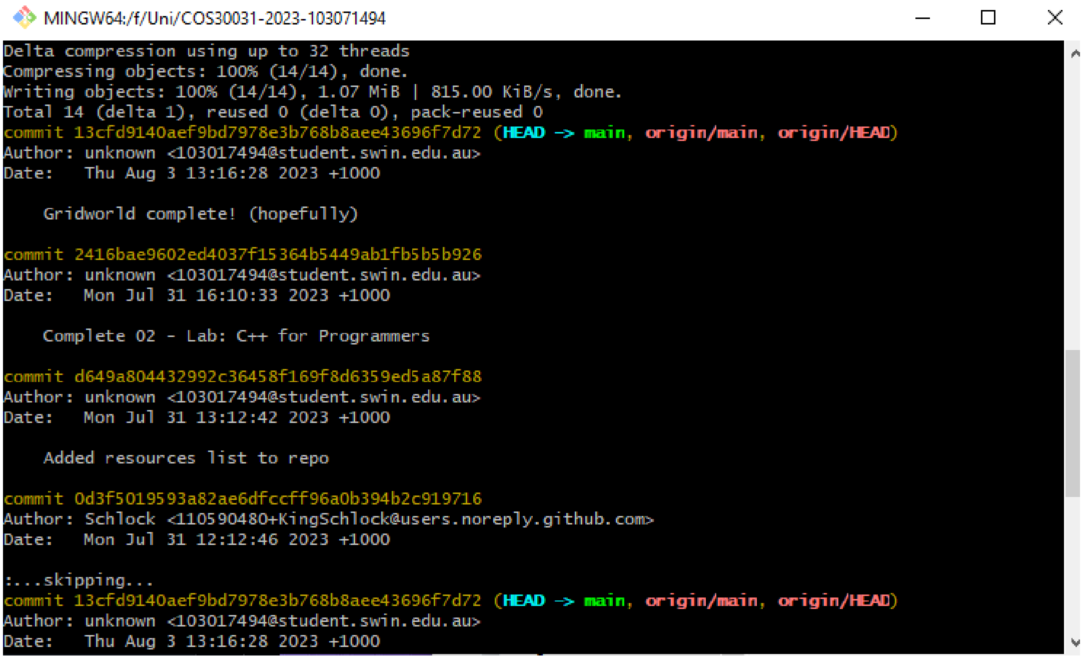
For simple projects like this it’s often beneficial to have the input handling managed in the update function. This approach was taken in my code.

Planning of Class World { };

- The world will be represented through an array of 64 chars spliced into an 8x8 map.
    - This map will be static
    - The map includes characters #, ^, G and S.
    - The player will not be it’s own entity as the only data we’re concerned with is it’s position on the map. Therefore it can be represented through coordinates only.
- As the World holds the map data it is able to check for collisions easily, this can be achieved through char comparison.
- The update method will hold a simple event handler function capable of knowing when the player dies or completes the level.

Spike Summary Report

Git Commit History



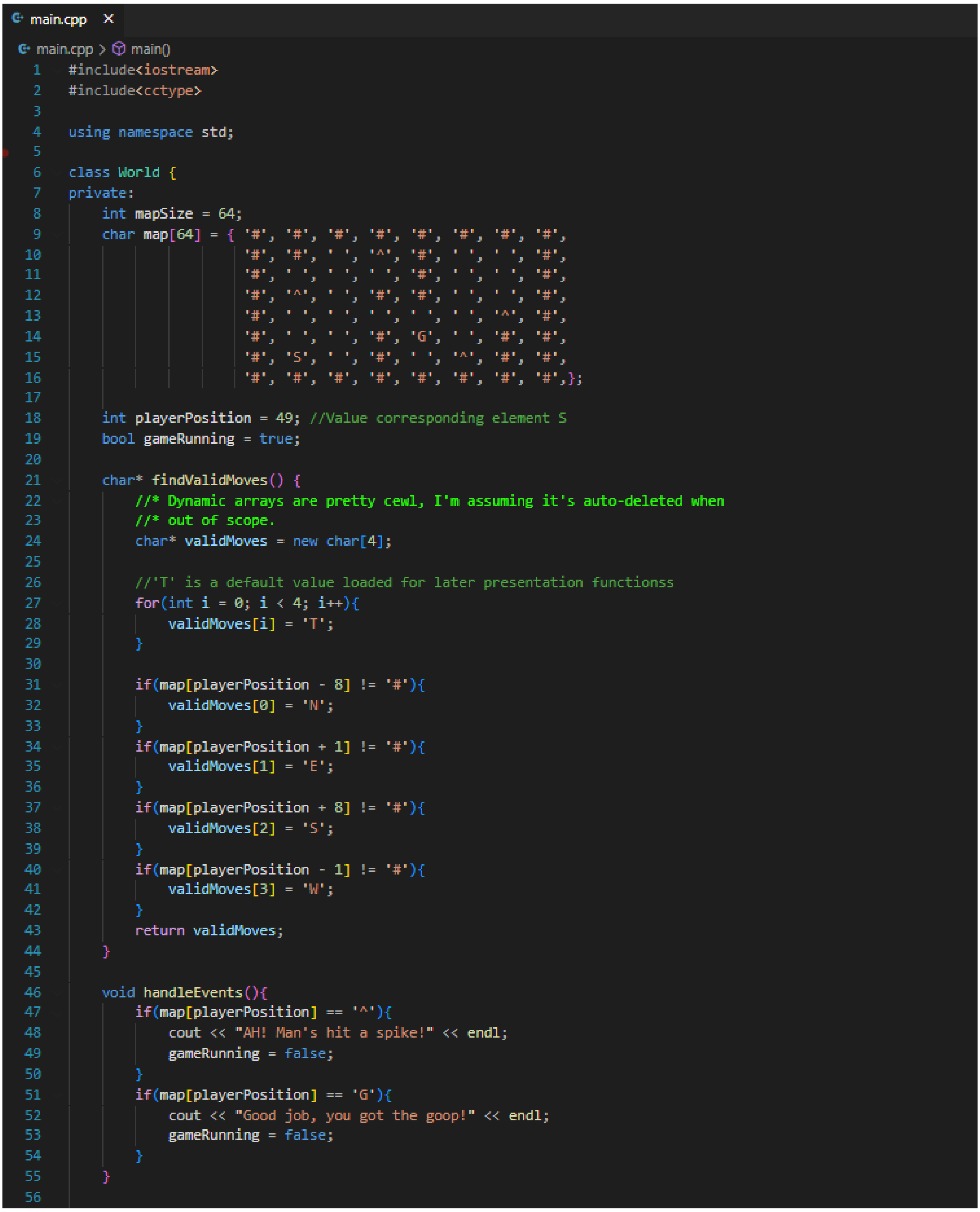
MINGW64:/f/Uni/COS30031-2023-103071494

```
Delta compression using up to 32 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 1.07 MiB | 815.00 KiB/s, done.
Total 14 (delta 1), reused 0 (delta 0), pack-reused 0
commit 13cfd9140aef9bd7978e3b768b8aee43696f7d72 (HEAD -> main, origin/main, origin/HEAD)
Author: unknown <103017494@student.swin.edu.au>
Date:   Thu Aug 3 13:16:28 2023 +1000

    Gridworld complete! (hopefully)

commit 2416bae9602ed4037f15364b5449ab1fb5b5b926
Author: unknown <103017494@student.swin.edu.au>
Date:   Mon Jul 31 16:10:33 2023 +1000

    Complete 02 - Lab: C++ for Programmers

commit d649a804432992c36458f169f8d6359ed5a87f88
Author: unknown <103017494@student.swin.edu.au>
Date:   Mon Jul 31 13:12:42 2023 +1000

    Added resources list to repo

commit 0d3f5019593a82ae6dfccff96a0b394b2c919716
Author: Schlock <110590480+KingSchlock@users.noreply.github.com>
Date:   Mon Jul 31 12:12:46 2023 +1000

:...skipping...
commit 13cfd9140aef9bd7978e3b768b8aee43696f7d72 (HEAD -> main, origin/main, origin/HEAD)
Author: unknown <103017494@student.swin.edu.au>
Date:   Thu Aug 3 13:16:28 2023 +1000
```

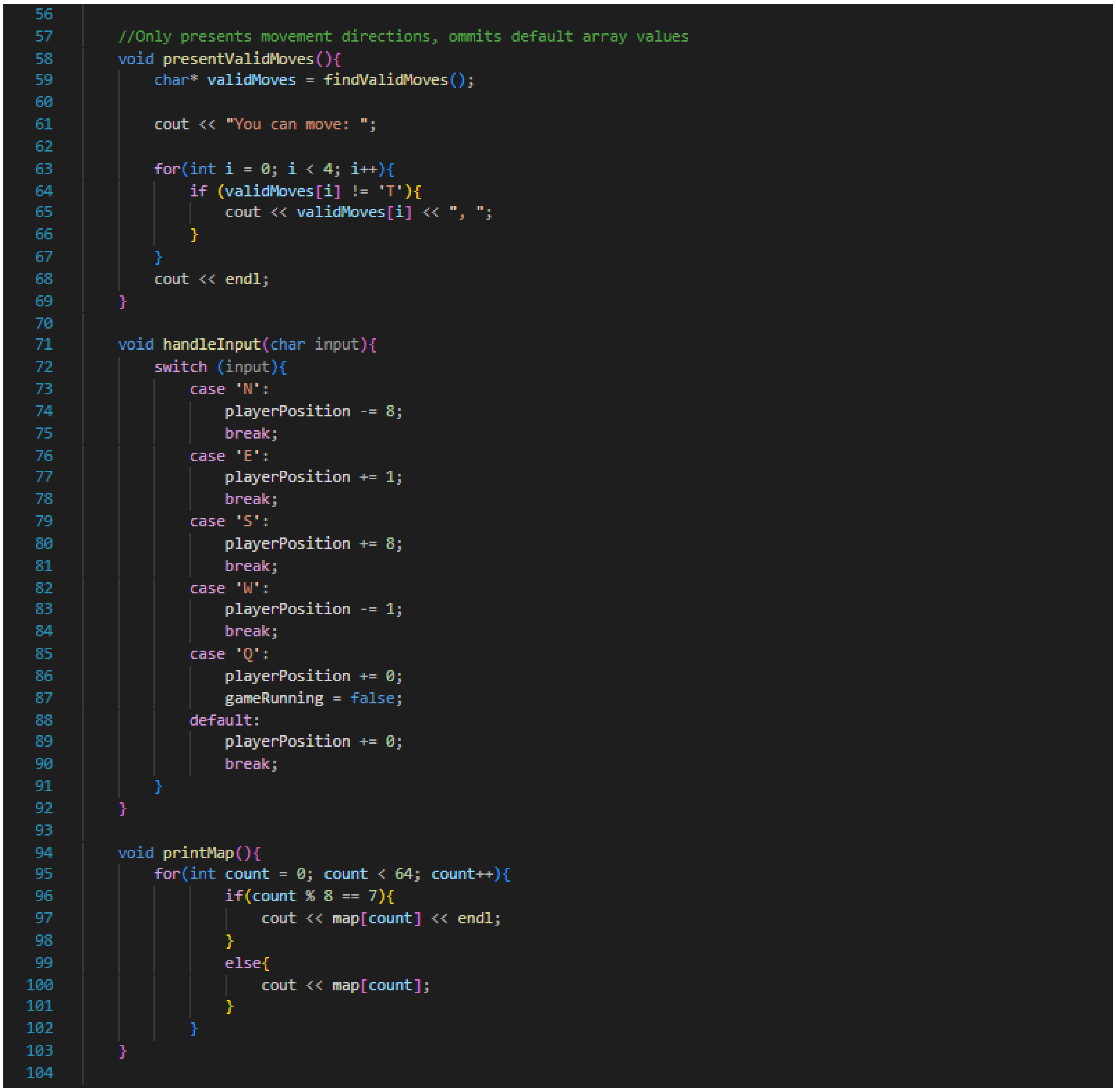
🧠

Spike Summary Report

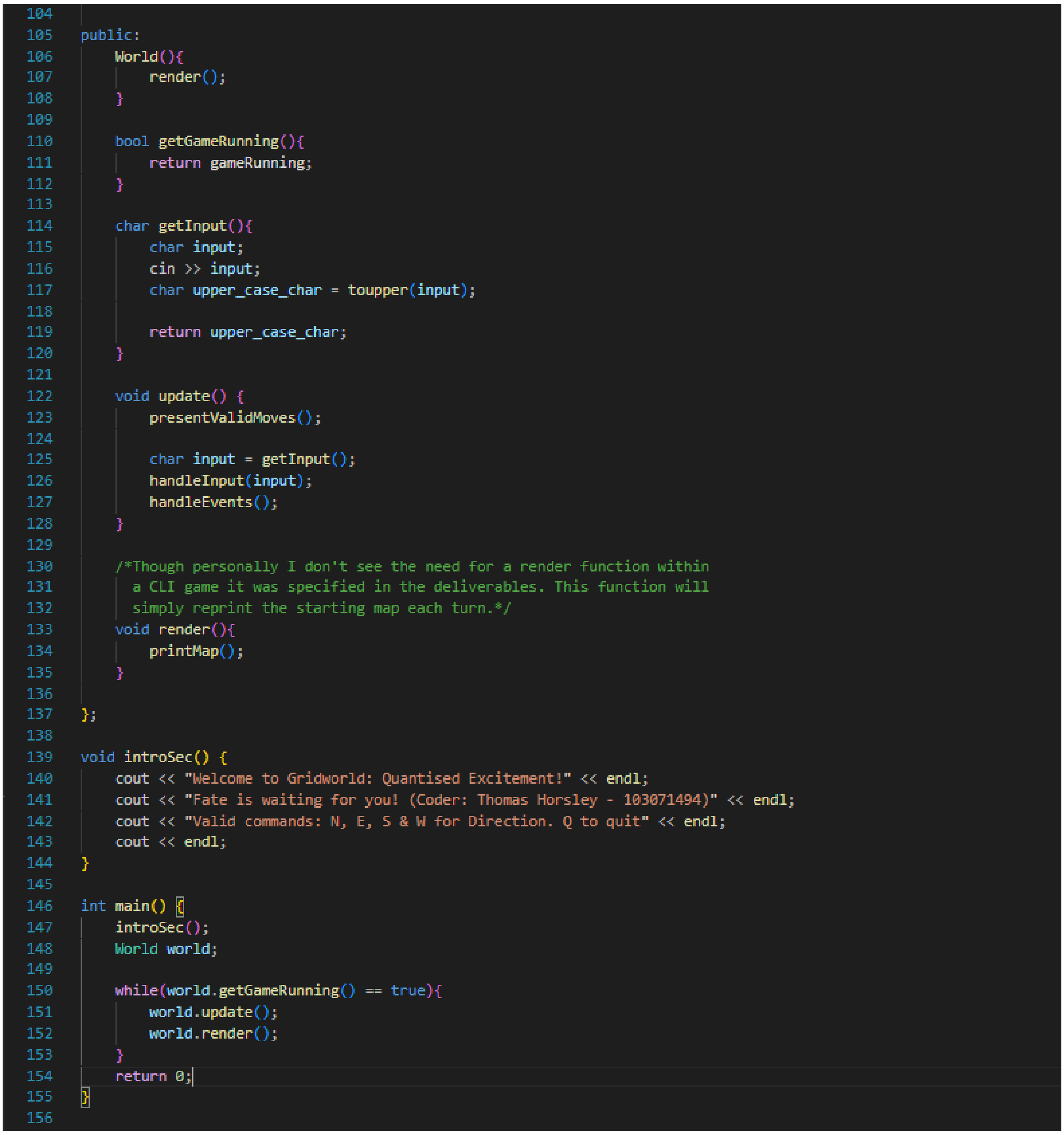
Code



```cpp
#include<iostream>
#include<cctype>

using namespace std;

class World {
private:
    int mapSize = 64;
    char map[64] = { '#', '#', '#', '#', '#', '#', '#', '#',
                     '#', '#', ' ', '^', '#', ' ', ' ', '#',
                     '#', ' ', ' ', ' ', '#', ' ', ' ', '#',
                     '#', '^', ' ', '#', '#', ' ', ' ', '#',
                     '#', ' ', ' ', ' ', ' ', ' ', '^', '#',
                     '#', ' ', ' ', '#', 'G', ' ', '#', '#',
                     '#', 'S', ' ', '#', ' ', '^', '#', '#',
                     '#', '#', '#', '#', '#', '#', '#', '#',};

    int playerPosition = 49; //Value corresponding element S
    bool gameRunning = true;

    char* findValidMoves() {
        //* Dynamic arrays are pretty cewl, I'm assuming it's auto-deleted when
        //* out of scope.
        char* validMoves = new char[4];

        //'T' is a default value loaded for later presentation functionss
        for(int i = 0; i < 4; i++){
            validMoves[i] = 'T';
        }

        if(map[playerPosition - 8] != '#'){
            validMoves[0] = 'N';
        }
        if(map[playerPosition + 1] != '#'){
            validMoves[1] = 'E';
        }
        if(map[playerPosition + 8] != '#'){
            validMoves[2] = 'S';
        }
        if(map[playerPosition - 1] != '#'){
            validMoves[3] = 'W';
        }
        return validMoves;
    }

    void handleEvents(){
        if(map[playerPosition] == '^'){
            cout << "AH! Man's hit a spike!" << endl;
            gameRunning = false;
        }
        if(map[playerPosition] == 'G'){
            cout << "Good job, you got the goop!" << endl;
            gameRunning = false;
        }
    }

```



```cpp

    //Only presents movement directions, ommits default array values
    void presentValidMoves(){
        char* validMoves = findValidMoves();

        cout << "You can move: ";

        for(int i = 0; i < 4; i++){
            if (validMoves[i] != 'T'){
                cout << validMoves[i] << ", ";
            }
        }
        cout << endl;
    }

    void handleInput(char input){
        switch (input){
            case 'N':
                playerPosition -= 8;
                break;
            case 'E':
                playerPosition += 1;
                break;
            case 'S':
                playerPosition += 8;
                break;
            case 'W':
                playerPosition -= 1;
                break;
            case 'Q':
                playerPosition += 0;
                gameRunning = false;
            default:
                playerPosition += 0;
                break;
        }
    }

    void printMap(){
        for(int count = 0; count < 64; count++){
                if(count % 8 == 7){
                    cout << map[count] << endl;
                }
                else{
                    cout << map[count];
                }
            }
    }

```



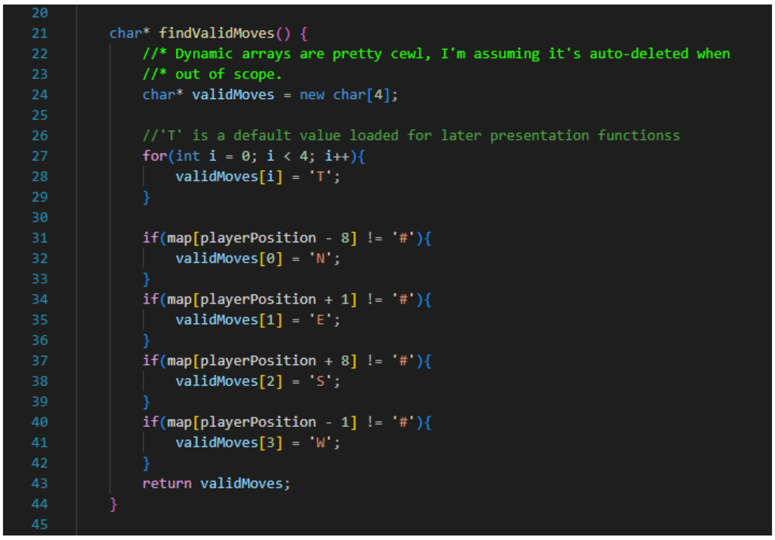
```cpp

public:
    World(){
        render();
    }

    bool getGameRunning(){
        return gameRunning;
    }

    char getInput(){
        char input;
        cin >> input;
        char upper_case_char = toupper(input);

        return upper_case_char;
    }

    void update() {
        presentValidMoves();

        char input = getInput();
        handleInput(input);
        handleEvents();
    }

    /*Though personally I don't see the need for a render function within
      a CLI game it was specified in the deliverables. This function will
      simply reprint the starting map each turn.*/
    void render(){
        printMap();
    }

};

void introSec() {
    cout << "Welcome to Gridworld: Quantised Excitement!" << endl;
    cout << "Fate is waiting for you! (Coder: Thomas Horsley - 103071494)" << endl;
    cout << "Valid commands: N, E, S & W for Direction. Q to quit" << endl;
    cout << endl;
}

int main() {
    introSec();
    World world;

    while(world.getGameRunning() == true){
        world.update();
        world.render();
    }
    return 0;
}

```

🧠

Spike Summary Report

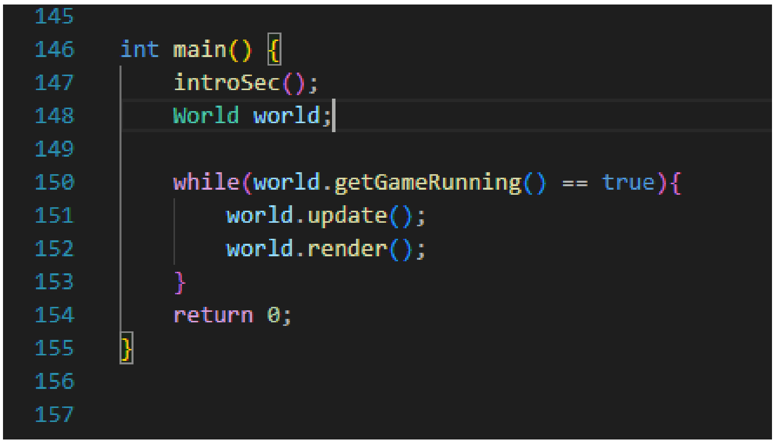
What was Learned?

💡 Programming Gridworld requires a solid understanding of C++ arrays (and by extension pointers and memory allocation) and how to manipulate them appropriately. Additionally, there’s an emphasis on basic games architecture and flow control.



```cpp

    char* findValidMoves() {
        //* Dynamic arrays are pretty cewl, I'm assuming it's auto-deleted when
        //* out of scope.
        char* validMoves = new char[4];

        //'T' is a default value loaded for later presentation functionss
        for(int i = 0; i < 4; i++){
            validMoves[i] = 'T';
        }

        if(map[playerPosition - 8] != '#'){
            validMoves[0] = 'N';
        }
        if(map[playerPosition + 1] != '#'){
            validMoves[1] = 'E';
        }
        if(map[playerPosition + 8] != '#'){
            validMoves[2] = 'S';
        }
        if(map[playerPosition - 1] != '#'){
            validMoves[3] = 'W';
        }
        return validMoves;
    }

```

How I checked for wall collisions



```cpp

int main() {
    introSec();
    World world;

    while(world.getGameRunning() == true){
        world.update();
        world.render();
    }
    return 0;
}


```

The game loop…. so nice and clean!