Spike: Spike No.1
Title: Task 3 - Grid World

Author: Khang Trinh - 102118468

Goals / Deliverables:

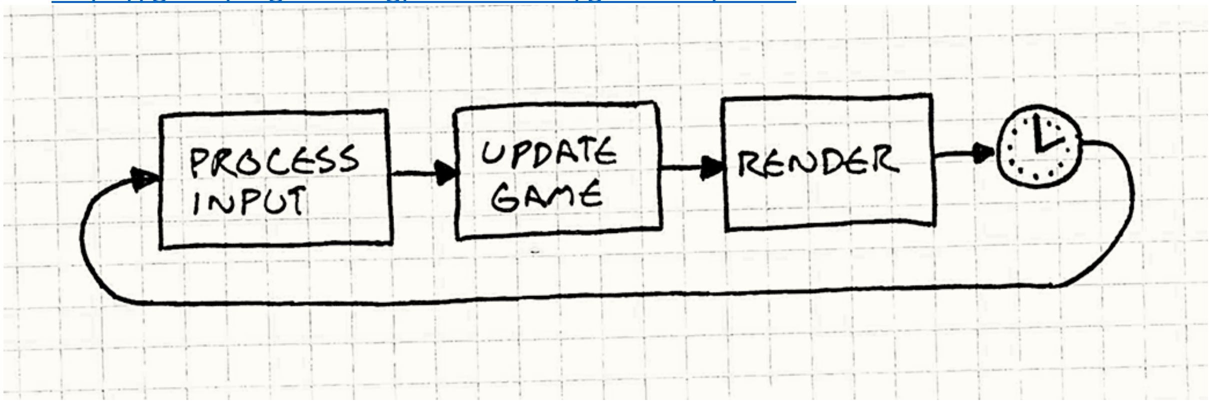
The goal of this spike is to explore the basics of a game loop - the separation of the input/update/render code, using the creation of GridWorld as an example.

Technologies, Tools, and Resources used:

- Visual Studio 2017

Useful Links:

1. **Theory on Game Loop:**
https://gameprogrammingpatterns.com/game-loop.html

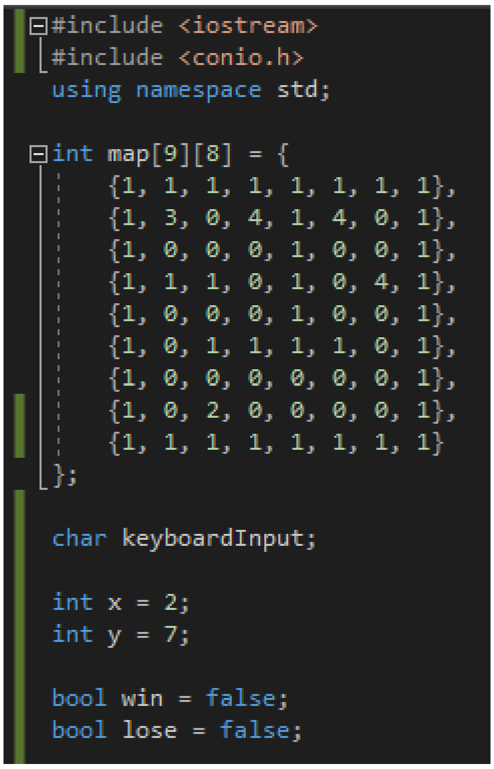


2. **About multi-dimensional array**
https://www.tutorialspoint.com/cplusplus/cpp_multi_dimensional_arrays.htm
3. **For-looping through a 2d array**
https://stackoverflow.com/questions/16509079/foreach-loop-in-2d-arrays-in-c
4. **How to create a text-based adventure game in C++**
https://www.youtube.com/watch?v=K6zAECdwDP8&ab_channel=GDQuest

Tasks undertaken:

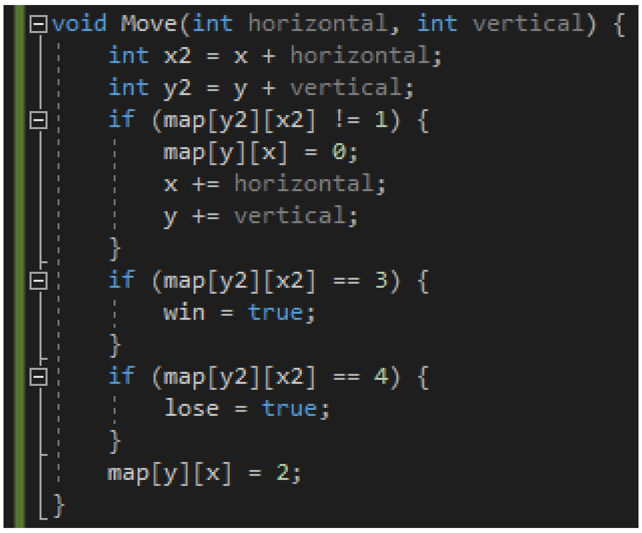
0. Some contexts

Since the focus of this spike is to explore the game loop, ***not*** the creation of GridWorld, some variables will be shown only to help the reader understand code better. Ultimately the main functions (GetIput, Update, Render and main) are the only ones needed to be understood to grasp this concept of "the basic game loop".



```
#include <iostream>
#include <conio.h>
using namespace std;

int map[9][8] = {
        {1, 1, 1, 1, 1, 1, 1, 1},
        {1, 3, 0, 4, 1, 4, 0, 1},
        {1, 0, 0, 0, 1, 0, 0, 1},
        {1, 1, 1, 0, 1, 0, 4, 1},
        {1, 0, 0, 0, 1, 0, 0, 1},
        {1, 0, 1, 1, 1, 1, 0, 1},
        {1, 0, 0, 0, 0, 0, 0, 1},
        {1, 0, 2, 0, 0, 0, 0, 1},
        {1, 1, 1, 1, 1, 1, 1, 1}
};

char keyboardInput;

int x = 2;
int y = 7;

bool win = false;
bool lose = false;
```

Fig 1. Global variables related to the game

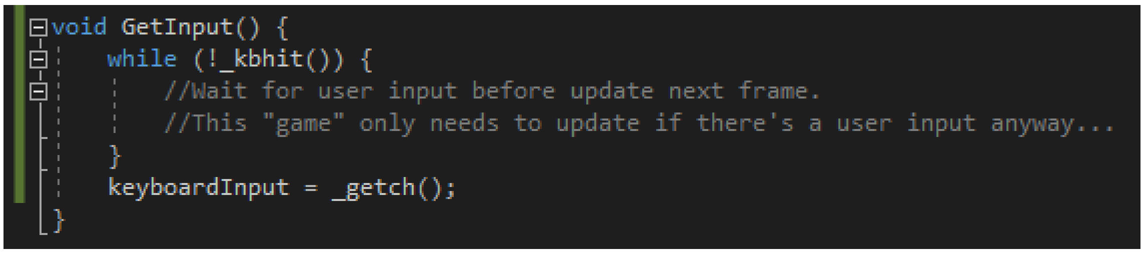


```
void Move(int horizontal, int vertical) {
    int x2 = x + horizontal;
    int y2 = y + vertical;
    if (map[y2][x2] != 1) {
        map[y][x] = 0;
        x += horizontal;
        y += vertical;
    }
    if (map[y2][x2] == 3) {
        win = true;
    }
    if (map[y2][x2] == 4) {
        lose = true;
    }
    map[y][x] = 2;
}
```

Fig 2. A function to be used in Update()

1. Create a GetInput() function

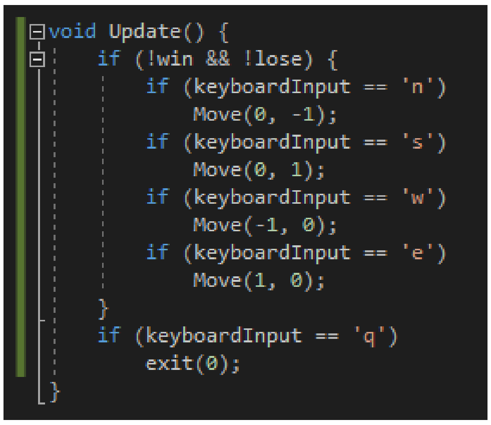
To play a game, the player needs a place for input. So, we'll need to make an Input() function for this. For GridWorld, the game runs on text commands. This means it only needs to update when received a keyboard input from the player. This means we could suspend the game until a key is pressed.



```
void GetInput() {
    while (!_kbhit()) {
        //Wait for user input before update next frame.
        //This "game" only needs to update if there's a user input anyway...
    }
    keyboardInput = _getch();
}
```

Fig 3. The Input() Loop

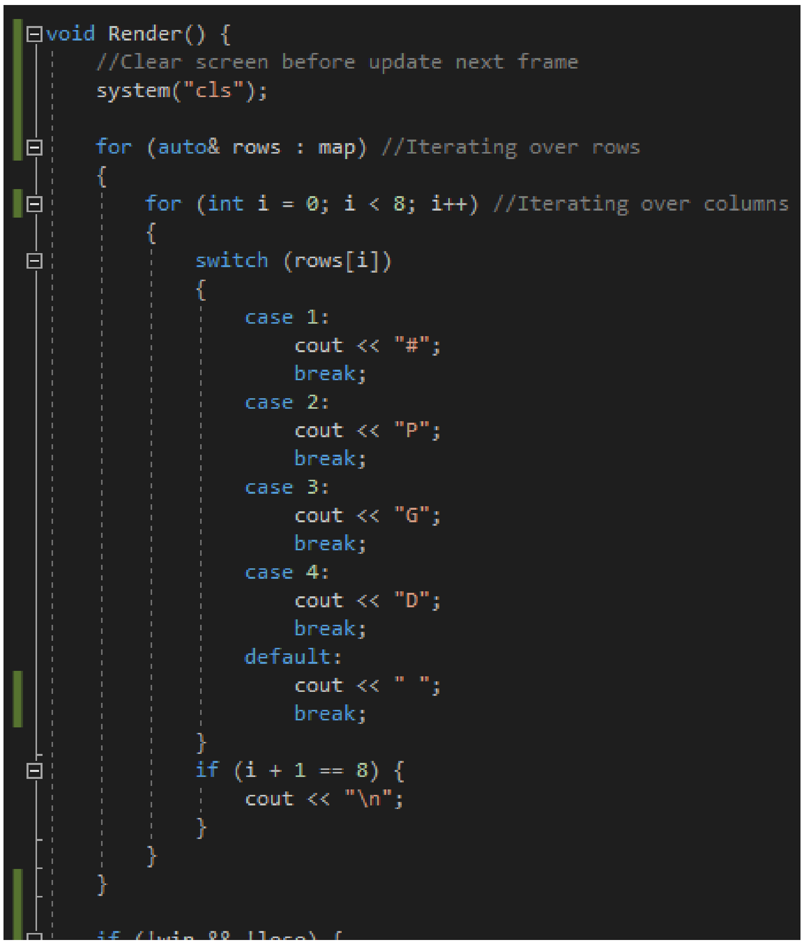
2. **Create an Update() function**
   The Update() loop is where all the logic in the game happens.This is where we'll be processing the player's input, turning them into actions. Eg. "If the player's pressed N, move them 1 up."



```cpp
void Update() {
    if (!win && !lose) {
        if (keyboardInput == 'n')
            Move(0, -1);
        if (keyboardInput == 's')
            Move(0, 1);
        if (keyboardInput == 'w')
            Move(-1, 0);
        if (keyboardInput == 'e')
            Move(1, 0);
    }
    if (keyboardInput == 'q')
        exit(0);
}
```

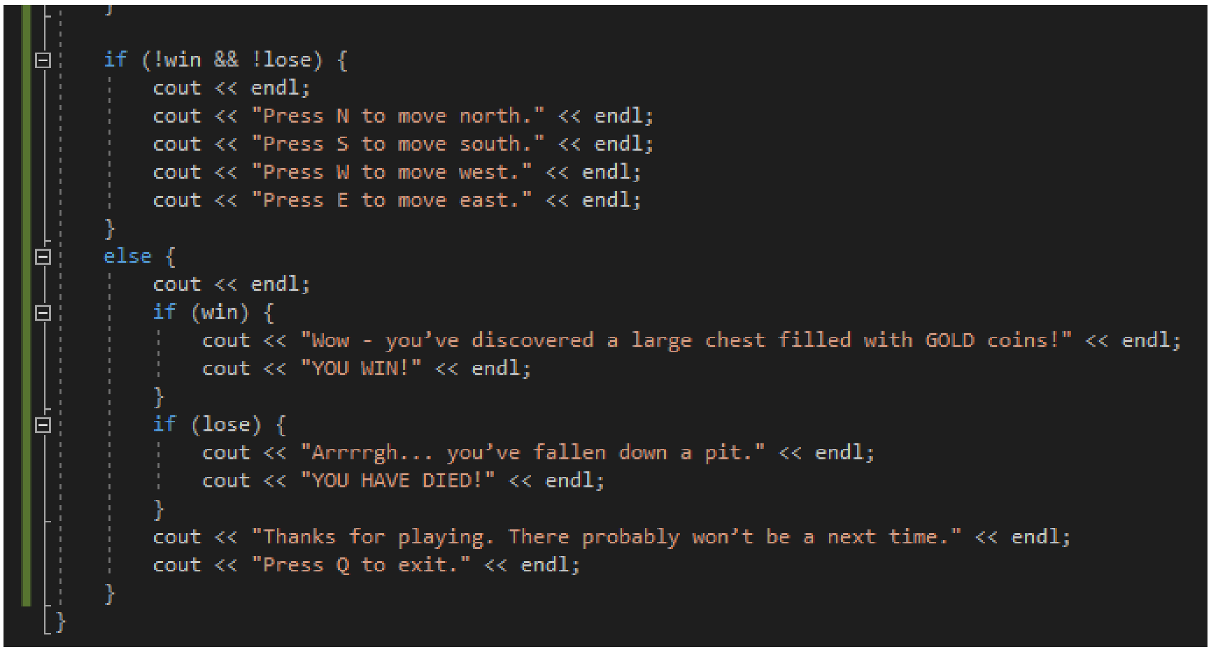
Fig 4. The Update() Loop

3. **Create a Render() function**
   After updating everything in the game, we need to give the player feedback that their input has been taken in and processed. This is where the Render() function comes in. With Render(), we *strictly* only worry about how to best render the game to the player.



```cpp
void Render() {
    //Clear screen before update next frame
    system("cls");

    for (auto& rows : map) //Iterating over rows
    {
        for (int i = 0; i < 8; i++) //Iterating over columns
        {
            switch (rows[i])
            {
                case 1:
                    cout << "#";
                    break;
                case 2:
                    cout << "P";
                    break;
                case 3:
                    cout << "G";
                    break;
                case 4:
                    cout << "D";
                    break;
                default:
                    cout << " ";
                    break;
            }
            if (i + 1 == 8) {
                cout << "\n";
            }
        }
    }

    if (!win && !lose) {
```

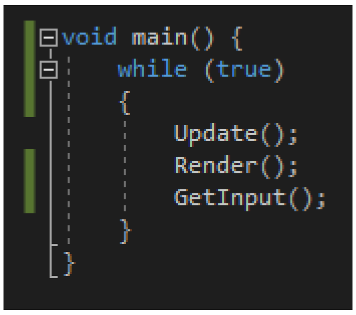
Fig 5. The Render() Loop



```
    if (!win && !lose) {
        cout << endl;
        cout << "Press N to move north." << endl;
        cout << "Press S to move south." << endl;
        cout << "Press W to move west." << endl;
        cout << "Press E to move east." << endl;
    }
    else {
        cout << endl;
        if (win) {
            cout << "Wow - you’ve discovered a large chest filled with GOLD coins!" << endl;
            cout << "YOU WIN!" << endl;
        }
        if (lose) {
            cout << "Arrrrgh... you’ve fallen down a pit." << endl;
            cout << "YOU HAVE DIED!" << endl;
        }
        cout << "Thanks for playing. There probably won’t be a next time." << endl;
        cout << "Press Q to exit." << endl;
    }
}
```

Fig 6. Add some rendering juice here if you want :D

4. **Create the main loop**
   Finally, we need a main loop that will tie everything together.
   ****Note**: Usually the input loop should be at the beginning, but in *this* case, we want to render the game to the player first to show them what they can do before taking their input. Therefore, we put the input loop at the bottom to skip the first input loop.



```
void main() {
    while (true)
    {
        Update();
        Render();
        GetInput();
    }
}
```

Figure 7. The main() Loop

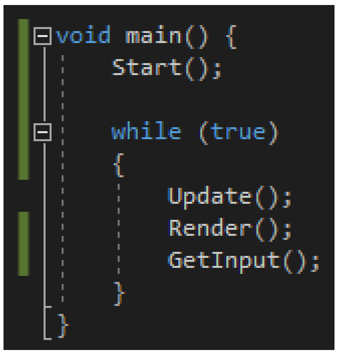
What we found out:

A basic game loop needs to do 3 things: take in the player's input, update the game's state with or without the player's input, and render the game according to the changes to provide feedback to the player.

As mentioned before, for games like GridWorld that runs on text-based commands, and don't have anything that can move independently from the player, these games only need to update when received an input from the player. In these particular cases, both Update() *and* Render() don't need to update every frame, but rather only after Input() has received a keypress from the player.

Recommendations

- If you are attempting to recreate GridWorld. Keep in mind that with the 2D array, it takes in y (the vertical element) first, then x (the horizontal element).
- If initialization is needed, it's worth creating a Start() function to keep things organized.



```
void main() {
    Start();

    while (true)
    {
        Update();
        Render();
        GetInput();
    }
}
```

Fig 8. Where Start() should be