Spike: Task 3
Title: GridWorld
Author: Sam Huffer, 101633177

Goals / deliverables:

- A simple plan of the code design
- Working code where:
    - The player can move their character through a grid-world of block positions, moving North, South, East and West.
    - Each time they move, they can see where they're allowed to go next.
    - If the player finds gold, they win. If they find a pit, they die.
    - If the player inputs Q, rather than N, S, E or W, they quit the game.
    - The game is case insensitive with regards to players' input.

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2017
- Inspiration 8 IE
- Draw.io

Tasks undertaken:

- Completed a quick plan of what variables and procedures would be necessary, as well as the logical flow of the game (see section "Plan")
- Created an empty VS console application project to build this game in.
- Set up variables in main()
- Created the Input(), Update(), and Render() procedures, setting up basic input and output functionality to test that it was registering input properly, and to distinguish between valid single-character input and invalid multi-character input. Input uses "cin >> variable;" to assign the player's input to the variable specified so that it can be processed in Update(). Here, I also set up the procedures ConvertToUppercase() and ConvertToLowercase() to standardise the input's capitalisation.
- Fleshed out the Update() procedure to pass appropriate values to the newly created MoveTo() function according to the input, so that it can determine the validity and outcome of the player's input movement, and to get directions the player can move to next from the also newly created function GetAvailableDirections().
- I updated Render() to output not just where the player went and where they may move to, but also their current position and the ASCII-art map, as the directions the player could move in wasn't lining up with my expectations. The rendered map illustrated that I'd forgotten to reflect and rotate the map to fit the way arrays are declared, rather than simply transpose it to look visually the same in code, so I spent some time figuring out how to reflect and rotate the map to have its output match what was shown in the GridWorld specifications.
- Once the map was reoriented, I tested it again and found that while the map was represented properly when printed, the player was still not moving as expected. On inspecting the MoveTo() and GetAvailableDirections() code, I found that I was modifying the wrong values and doing so incorrectly when updating the player's position, and that I had accidentally reversed each axis when displaying the available directions. Fixing both sections resulted in the game working as intended.
- Lastly, tidied up some of the cout statements and output strings so that everything was output in an organised manner to the terminal.

Plan:

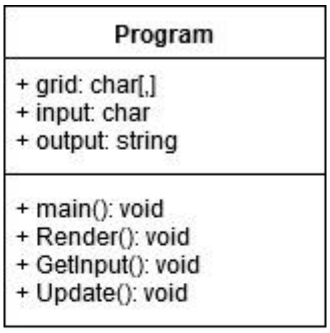


Figure 1: a basic outline of the variables and procedures to be implemented.

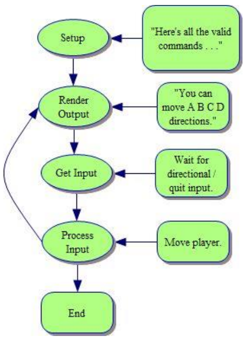
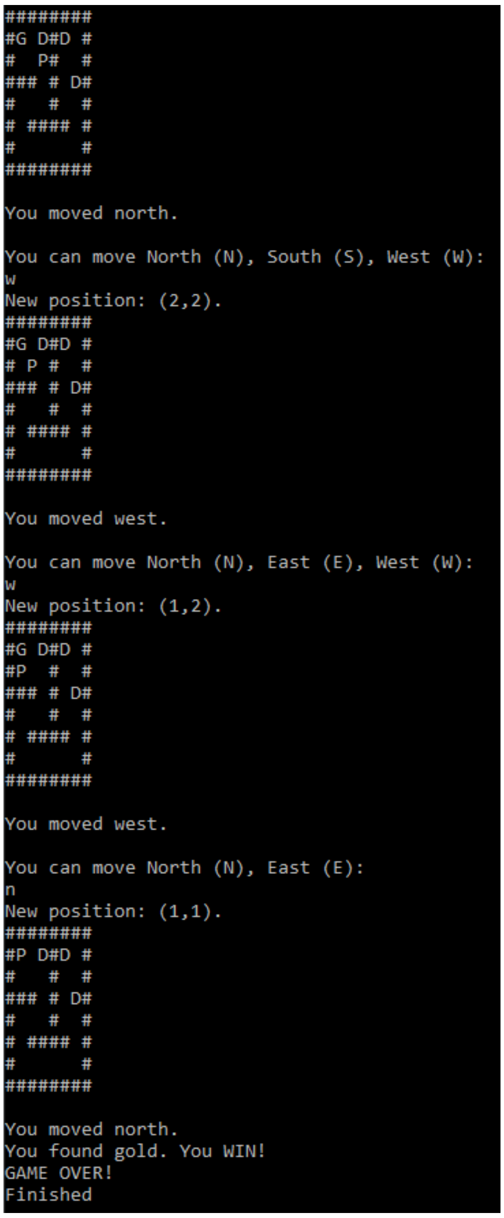

Setup
"Here's all the valid commands . . ."
Render Output
"You can move A B C D directions."
Get Input
Wait for directional / quit input.
Process Input
Move player.
End

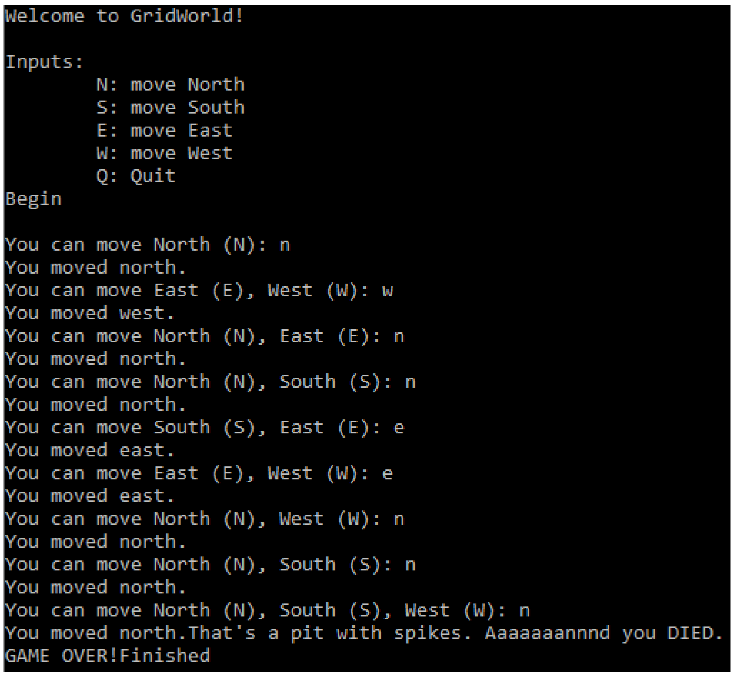
Figure 2: a basic outline of the logical flow of the GridWorld game.

Terminal Screenshots:



```
########
#G D#D #
#  P#  #
### # D#
#   #  #
# #### #
#      #
########

You moved north.

You can move North (N), South (S), West (W):
w
New position: (2,2).
########
#G D#D #
# P #  #
### # D#
#   #  #
# #### #
#      #
########

You moved west.

You can move North (N), East (E), West (W):
w
New position: (1,2).
########
#G D#D #
#P  #  #
### # D#
#   #  #
# #### #
#      #
########

You moved west.

You can move North (N), East (E):
n
New position: (1,1).
########
#P D#D #
#   #  #
### # D#
#   #  #
# #### #
#      #
########

You moved north.
You found gold. You WIN!
GAME OVER!
Finished
```

Figure 3: Screenshot of the output printed to the terminal while testing.



```
Welcome to GridWorld!

Inputs:
        N: move North
        S: move South
        E: move East
        W: move West
        Q: Quit
Begin

You can move North (N): n
You moved north.
You can move East (E), West (W): w
You moved west.
You can move North (N), East (E): n
You moved north.
You can move North (N), South (S): n
You moved north.
You can move South (S), East (E): e
You moved east.
You can move East (E), West (W): e
You moved east.
You can move North (N), West (W): n
You moved north.
You can move North (N), South (S): n
You moved north.
You can move North (N), South (S), West (W): n
You moved north.That's a pit with spikes. Aaaaaaannnd you DIED.
GAME OVER!Finished
```

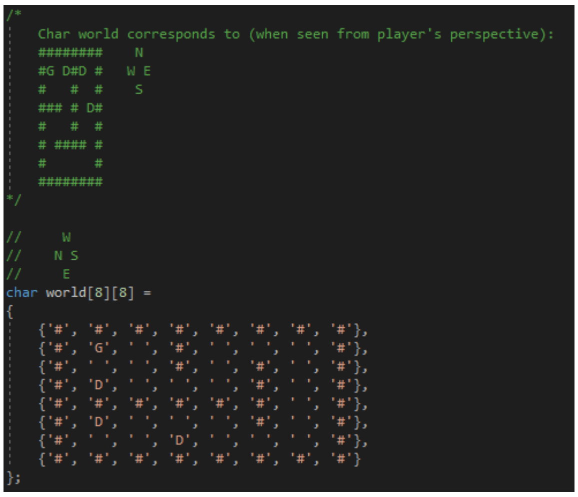
Figure 4: Final, tidied up output without the map rendered; the map would give away too much information about where things are.

What we found out:

This task was pretty good for helping me adjust to the basics of coding in C++ rather than more user-friendly languages like C#, teaching me:

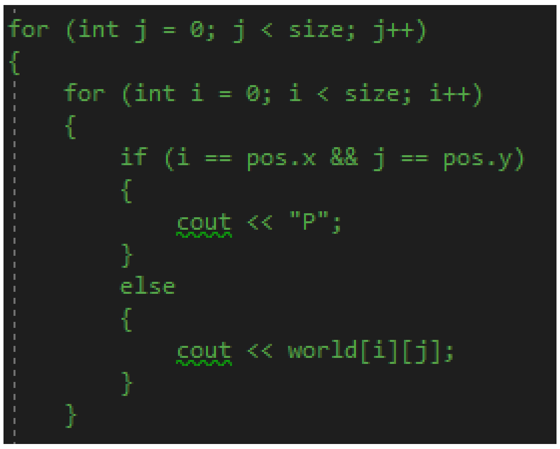
- How to declare 2D arrays (fig. 5).
- That you need to reflect and rotate hard-coded array elements to match the internal structure of a 2D array instead of how it might be oriented on paper (fig. 5). Displaying them needs to be done column by column as well, rather than row by row (fig. 6).
- How to pass 2D arrays to functions (e.g. parameter declaration would be char world[8][8], and passing it would be function(world)); they are automatically passed by reference, though seem to require having their dimensions spelled out in the parameter declaration.
- That arrays don't like to have their dimensions defined by a variable; they need to be hard coded.
- How to pass classes by reference (e.g. parameter declaration would be "Vector2& pos" rather than "Vector2 pos" or "Vector2 * pos", and it'd be passed into the function as function(pos)); passing by pointer got messy, and passing classes the way arrays are passed seemed to pass them just by value rather than by reference.
- How to read input from the terminal (fig. 7).
- How to convert all characters in a string to uppercase or lowercase (fig. 7).
- How to use strings in switches: you can't; you've got to use a basic data type (int, float, char, etc.) instead, as the string data type belongs to a library rather than just being a part of C++. Accessing individual characters within a string does work fine, however; C++ switches recognise them as chars, even if they don't recognise the string data type.

Code Screenshots:



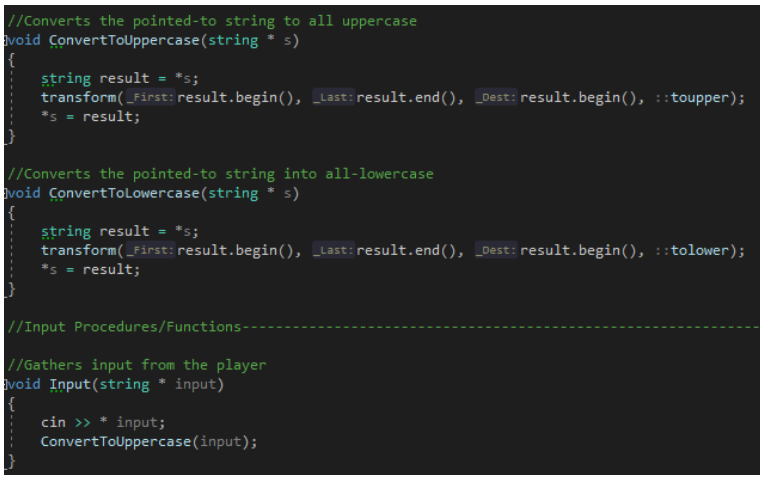
```cpp
/*
    Char world corresponds to (when seen from player's perspective):
    ########    N
    #G D#D #   W E
    #   #  #    S
    ### # D#
    #   #  #
    # #### #
    #      #
    ########
*/

//      W
//     N S
//      E
char world[8][8] =
{
    {'#', '#', '#', '#', '#', '#', '#', '#'},
    {'#', 'G', ' ', '#', ' ', ' ', ' ', '#'},
    {'#', ' ', ' ', '#', ' ', '#', ' ', '#'},
    {'#', 'D', ' ', ' ', ' ', '#', ' ', '#'},
    {'#', '#', '#', '#', '#', '#', ' ', '#'},
    {'#', 'D', ' ', ' ', ' ', '#', ' ', '#'},
    {'#', ' ', ' ', 'D', ' ', ' ', ' ', '#'},
    {'#', '#', '#', '#', '#', '#', '#', '#'}
};
```

Figure 5: declaring a 2D array.



```cpp
for (int j = 0; j < size; j++)
{
    for (int i = 0; i < size; i++)
    {
        if (i == pos.x && j == pos.y)
        {
            cout << "P";
        }
        else
        {
            cout << world[i][j];
        }
    }
```

Figure 6: printing the 2D array column by column, rather than row by row.



```cpp
//Converts the pointed-to string to all uppercase
void ConvertToUppercase(string * s)
{
    string result = *s;
    transform(_First:result.begin(), _Last:result.end(), _Dest:result.begin(), ::toupper);
    *s = result;
}

//Converts the pointed-to string into all-lowercase
void ConvertToLowercase(string * s)
{
    string result = *s;
    transform(_First:result.begin(), _Last:result.end(), _Dest:result.begin(), ::tolower);
    *s = result;
}

//Input Procedures/Functions-----------------------------------------------------------

//Gathers input from the player
void Input(string * input)
{
    cin >> * input;
    ConvertToUppercase(input);
}
```

Figure 7: Standardising the capitalisation of strings, and reading input from the terminal.