

**Spike:** 01 (Task 03)

**Title:** Gridworld

**Author:** Hayden Whiteford, 104001272

**Goals / deliverables:**

- Create the Gridworld game as per the specification document
- Demonstrate separation of input, update and render code
- Use of game data for player and Gridworld map

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

- Xcode for Mac
- W3 schools to brush up on c++

**Tasks undertaken:**

**How to run:**

- Download and install Xcode or visual studio
- Create a new c++ console application project
- Replace main file with my code
- Compile and run

**In terms of Code:**

- Created a vector containing more char vectors to create the Gridworld "map"
- Created a few functions:
- update() - updates player position based on input - decides output with intractable items D and G
- checkMoves() - Checks the valid moves for the player based on surrounding chars in the vector grid
- render() - uses a for loop to output the world, can override the grid world with a marker for the players position - "P"

**What we found out:**

Went fairly smoothly and was able to bang this out mostly in a single night. Did need to brush up on C++ to check out arrays vs vectors. Vectors worked better here as they were dynamic in terms of size, and for the checkmoves() function I could simply use the push\_back function to append to the vectors the valid moves for the turn. Splitting up functions went smoothly as it seemed the most straightforward way of structuring the code - having an update function for portions and interactables, something to check the moves, and something to render the board, which I decided I should implement from a debug perspective.

The screenshot shows a C++ IDE with a project named 'Gridworld'. The main window displays the 'main' file, which contains a function `update` that processes a move. The code is as follows:

```
76 bool update(char aMove, std::vector<char> aMoves)
77 {
78     //affect player position by move variable
79     if (std::find(aMoves.begin(), aMoves.end(), aMove) != aMoves.end() || aMove == 'Q')
80     {
81         if (aMove == 'N')
82         {
83             playerRow--;
84         }
85         else if (aMove == 'S')
86         {
87             playerRow++;
88         }
89         else if (aMove == 'E')
90         {
91             playerCol++;
92         }
93     }
94 }
```

The right sidebar shows a list of files in the project. The bottom panel displays the output of the program, which shows the player's position and the game state. The output is as follows:

```
#####
You Can Move N, W:
N
#####
#G D#D #
# # #
###P# D#
# # #
# #### #
# # #
##S#####
You Can Move N, S:
N
#####
#G D#D #
# P# #
### # D#
# # #
# #### #
# # #
##S#####
You Can Move N, S, W:
N
Arrrrgh... you've fallen down a pit.
YOU HAVE DIED!
Thanks for playing. Maybe next time.
Program ended with exit code: 0
```

### Open issues/risks:

This wasn't written OO style. I felt I didn't need it and would have made it needlessly complicated, but could quite easily implement a game world and player object if that's needed.