



# C++ Design

Documentation of my third implementation of my game, this time in the Object Oriented paradigm with the C++ language.

This was the third and final language and paradigm I chose to tackle.

## Pre-implementation

Prior to starting my C++ implementation, I needed to ensure my Object Oriented knowledge was sufficient to start.

```
53 // Game class
54 class Game {
55     // private variables, private because we don't want them to be accessed outside of the class
56     private:
57         Board * board;
58         char turns[9] = {'0', 'X', '0', 'X', '0', 'X', '0', 'X', '0'};
59         int turn_index = 0;
60
61     void takeTurn(char player) {
62         cout << "It's player " << player << "'s turn\n" << endl; // cout is print and endl ends the line so the next will be
63         cout << "Enter a move index (1-9): " << endl;
64         int move;
65         cin >> move; // cin is input
66         board->add(move - 1, player); // can also use: (*board).add(move - 1, player);
67     }
68
69 public:
70     Game(Board * reqBoard) { // constructor. * reqBoard is a pointer to a Board object - a C++ standard
71         board = reqBoard;
72     }
73
74     bool ended() {
75         return turn_index >= 9; // return true if turn_index is greater than or equal to 9
76     }
77
78     void play() {
79         char player = turns[turn_index];
80         turn_index++;
81         board->displayBoard(); // -> is used to access a member of a pointer to an object
82         takeTurn(player);
83         if (board->winner(player)) { // if the player has won
84             board->displayBoard();
85             cout << "Player " << player << " wins!" << endl;
86             turn_index = 9; // end the game
87         }
88     }
89 }
90
91 int main() {
92     Board * board = new Board(); // new Board() creates a new Board object and returns a pointer to it
93     Game * game = new Game(board); // new Game(board) creates a new Game object and returns a pointer to it
94
95     while(! game->ended())
96         game->play();
97 }
```

I have used OO languages before, however I have not made good use of them and didn't use for long. This time I wanted to ensure I did. I followed the lecture multiple times and made notes to refer to. I also documented the fizz buzz and tic-tac-toe code we created in the labs thoroughly, as seen above.

After doing this, I knew the use of classes would work greatly for my game, specifically for the player buildings. With this knowledge, I started the implement.

## **Implementation**

I created a private GitHub repository for the code. With this I could commit new code, changes and updates and rollback if later features causes game breaking issues. Throughout the code that I shall describe, you will notice my use of camel casing for functions and procedures. This is not typically the format in which I follow. Camel case for variables, snake case for methods. Here my use of camel case was implemented simply by a lack of concentration. I later modify these to snake case.

### **Commit 1 - Print Buildings, Input and Update**

This commit is quite a large one. I did this as I was in a productive streak and wanted to get a large amount done. I started with the building class.

```

1 + #include <iostream>
2 +
3 + using namespace std;
4 +
5 + class Building {
6 + public:
7 +     Building() {
8 +         int startingBuilding[4][5] = {
9 +             {1, 0, 0, 0, 1},
10 +            {0, 1, 0, 1, 0},
11 +            {0, 1, 0, 1, 0},
12 +            {0, 0, 1, 0, 0}
13 +        };
14 +
15 +         for (int i = 0; i < 4; i++) {
16 +             for (int j = 0; j < 5; j++) {
17 +                 building[i][j] = startingBuilding[i][j];
18 +             }
19 +         }
20 +     }
21 +
22 +     void displayBuilding() {
23 +         for (int i = 0; i < 4; i++) {
24 +             for (int j = 0; j < 5; j++) {
25 +                 std::cout << building[i][j] << " ";
26 +             }
27 +             std::cout << std::endl;
28 +         }
29 +     }
30 +
31 +     void updateBuilding(int row, int col, int waterOrFire) {
32 +         if (row >= 0 && row < 4 && col >= 0 && col < 5) {
33 +             building[row][col] = waterOrFire;
34 +         }
35 +     }
36 +
37 + private:
38 +     int building[4][5];
39 + };
40 +

```

**Line 1:** I have limited C++ knowledge but from the lab work specifically, I knew that I needed include “`iostream`” for the input and output system.

**Line 3:** You may notice my use of “`std::`” throughout this initial commit. Due to my inexperience with the language, I originally didn’t realise using the namespace std on line 3, means that you do b’t have to state “`std::`” at the start of the commands that relate to it. This is rectified later.

**Line 5-39:** The building class. The object oriented paradigm starts to come into place. What you are seeing here is called encapsulation -the bundling of data and methods into a single unit, a class.

**Line 37-39:** Jumping to the bottom, this is where I created the attribute for the class, the building array. The array is a 5 item width by a 4 item depth, in C++ you state depth then width. This is data type int as all items in the array are integers. Originally I set this to private to encapsulate the data within the class but I later changed this.

**Line 7-20:** The constructor for the Building class. It initialises the building array with the startingBuilding array. Meaning that every time the game starts, the Building object is created and when the building object is created, this initialisation occurs.

**Line 15-17:** I was unsure how to create an array in C++ so I consulted ChatGPT.

 You  
how to set an array in c++

 ChatGPT  
In C++, you can initialize an array in various ways. Here are a few examples:

### 1. Initializing an Array with Explicit Values:

```
cpp                                         Copy code

#include <iostream>

int main() {
    // Initialize an array with explicit values
    int myArray[5] = {1, 2, 3, 4, 5};

    // Access and print elements of the array
    for (int i = 0; i < 5; ++i) {
        std::cout << myArray[i] << " ";
    }

    return 0;
}
```

C++ doesn't allow you to assign arrays. So we must make one, and copy the elements across.

Set the building array to have the values that are within the startingBuilding. Loop through the rows and columns and put each value into our new array.

**Line 22-29:** Display the building to the players in a easy to read format by the way of a rooms separated by a blank space.

**Line 31-35:** I created this for testing as I built the program. As within the Python and Prolog solutions, the arrays need to update and spread fire/water. Before I implemented that, I simply wanted to update the chosen room, so I passed in the value for water/fire and the row and

column of said room. The row and column is needed as the building arrays are 2D arrays. This requirement is seen on line 33. Change the value to 1 or 0 at the room's location.

## Commit 2 - Game Class

```

41 + class Game {
42 + private:
43 +     Building playerOneBuilding;
44 +     Building playerTwoBuilding;
45 +
46 + public:
47 +     int player;
48 +     int roomNumber, row, col;
49 +     int waterOrFire;
50 +
51 +     void displayBothBuildings() {
52 +         std::cout << "Player One's Building: " << std::endl;
53 +         playerOneBuilding.displayBuilding();
54 +         std::cout << std::endl;
55 +         std::cout << "Player Two's Building: " << std::endl;
56 +         playerTwoBuilding.displayBuilding();
57 +         std::cout << std::endl;
58 +     }
59 +
60 +     void playGame() {
61 +         while (true) {
62 +             displayBothBuildings();
63 +             std::cout << "Player One or Player Two? ";
64 +             std::cin >> player;
65 +
66 +             if (player == 1 || player == 2) {
67 +                 std::cout << "Press 1 to spread water in your building or press 0 to spread fire on your opponent\n";
68 +                 std::cout << "Enter 1 or 0: ";
69 +                 std::cin >> waterOrFire;
70 +                 std::cout << "Which room do you chose (1-20): ";
71 +                 std::cin >> roomNumber;
72 +                 get_row_col(roomNumber, row, col);
73 +
74 +                 if (player == 1 && waterOrFire == 1) {
75 +                     playerOneBuilding.updateBuilding(row, col, waterOrFire);
76 +                 } else if (player == 1 && waterOrFire == 0) {
77 +                     playerTwoBuilding.updateBuilding(row, col, waterOrFire);
78 +                 } else if (player == 2 && waterOrFire == 1) {
79 +                     playerTwoBuilding.updateBuilding(row, col, waterOrFire);
80 +                 } else if (player == 2 && waterOrFire == 0) {
81 +                     playerOneBuilding.updateBuilding(row, col, waterOrFire);
82 +                 }
83 +             }
84 +         }
85 +     }
86 +
87 +     void get_row_col(int roomNumber, int& row, int& col) {
88 +         if (roomNumber < 1 || roomNumber > 20) {
89 +             std::cout << "Invalid room number\n";
90 +         }
91 +         row = 3 - (roomNumber - 1) / 5;
92 +         col = (roomNumber - 1) % 5;
93 +     }
94 + };

```

**Line 41-94:** This is the 'Game' class. Here all the game logic code is stored. It calls methods that are within the Building class, this is encapsulation and I will point it out when we get to that section.

**Line 42-44:** Private variables for both player's buildings. This uses the OO concept of composition. This means that each 'Game' object contains a 'Building' object and here, I have two, being 'playerOneBuilding' and 'playerTwoBuilding'. I made these private as the program should only be able to access and used within the 'Game' class.

**Line 46-49:** Public variables. These public as I knew I would want to access them in the 'Building' class.

**Line 51-58:** A method to display both buildings to the console. You will see the use of encapsulation where I call `displayBuilding()` from the 'Building' class. Encapsulation is a popular Object Oriented concept that allows you to access methods within another class. As evident, I did wrongly camel case this procedure.

**Line 60-85:** Another sloppy naming scheme where I camel cased the procedure that will be adjusted later. While the game is running (this will be replaced with a winner check), Player 1 or 2 can select a room and whether they want to spread fire or water. Again, this is was a test as I start to implement functionality to the game. Some differences in languages can be seen with the use of how to get inputs and show outputs with `cin` and `cout`. In this version, I tested without a taking turn feature, instead I inputted which player is playing. More OO encapsulation is seen in the check of which player is playing and what they chose.

**Line 87-93:** The same functionality as with my Python and Prolog implementation whereby the row and column position in the 2D building array from the player selected room will be calculated. As this was my third iteration of the game, I decided to tested chatGPT's capability of converting my python implementation to C++.



You

```
can you convert this python into C++  
def update_building(roomNumber, row, col):  
    if 1 <= roomNumber <= 20:  
        row = 3 - (roomNumber - 1) // 5 # e.g. roomNumber = 13 so 13 - 1 = 12 // 5 = 2 so row = 3 - 2 = 1  
        which is the second row (0, 1, 2, 3)  
        col = (roomNumber - 1) % 5 # e.g. roomNumber = 13 so 13 - 1 = 12 % 5 = 2 so col = 2 which is  
        the third column (0, 1, 2, 3, 4)  
        return row, col
```



ChatGPT

Certainly! Here's the equivalent C++ code for the given Python function:

```
cpp  
Copy code  
  
#include <iostream>  
  
std::pair<int, int> update_building(int roomNumber, int& row, int& col) {  
    if (1 <= roomNumber && roomNumber <= 20) {  
        row = 3 - (roomNumber - 1) / 5;  
        col = (roomNumber - 1) % 5;  
    }  
    return std::make_pair(row, col);  
}
```

Clearly I changed some things by removing the use of ‘pair’ but I was shocked how easily it worked, however, the similarities between python and C++ are plentiful compared to a language like Prolog. You may notice the use of `int&`. This is a reference to an integer. This is common practice within C++ and while it is not an OO specific practice, I left it in. This is extent to which I used an LLM tool to convert my current code. I mainly did this as a test and consider it ethical to my project.

```
96 + int main() {  
97 +     Game gameInstance;  
98 +     gameInstance.playGame();  
99 +  
100 +     return 0;  
101 + }
```

**Line 96-101:** From my notes during the lecture and lab, I knew that to start a C++ program, you must make an object instance of the class containing the function you want to call. For my game, it needs to start by calling `playGame()` which lies within the ‘Game’ class. You must make a game instance so I named it appropriately. The `main()` is the start of the

program and by creating the ‘gameInstance’ object, the code follows the Object Oriented paradigm from the very beginning.

## Commit 3 - Removal of std::

This is a simple change whereby “I removed” all `std::` from the code as you don’t need to include the instance of it every time when you use the namespace “std” at the start of a C++ file. I quoted “I removed” because I actually used ChatGPT. I did this as it would speed up the process rather than removing every occurrence line by line.



You

can you remove all std:: from this code

```
#include <iostream>

using namespace std;

class Building {
public:
    Building() {
        // ...
    }
}
```

My prompt with the code supplied (not all attached).



ChatGPT

Certainly! Here is the code with all instances of `std::` removed:

```
cpp                                         Copy code

#include <iostream>

using namespace std;

class Building {
public:
    Building() {
        int startingBuilding[4][5] = {
            {1, 0, 0, 0, 1},
            {0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0},
            {0, 0, 1, 0, 0}
        };

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 5; j++) {
                building[i][j] = startingBuilding[i][j];
            }
        }
    }

    void displayBuilding() {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 5; j++) {
                cout << building[i][j] << " ";
            }
            cout << endl;
        }
    }
}
```

Part of the GPT response with lines in the 'displayBuilding' function, showing the removed `std::`, specifically the two `cout` commands.

## Commit 4 - Winner Functionality Mostly Implemented

```

51     public:
52     +     string player;
53             int roomNumber, row, col;
54             int waterOrFire;
55     +     bool gameOver = false;
56     +     bool playerOneWins = false;
57     +     bool playerTwoWins = false;
58     +     bool playerOnesGo = true;
59     +     bool playerTwosGo = false;
60

```

Added global variables for winner and turns.

```

70         void gameLoop() {
71     +             while (!gameOver) {
72     +                 check_winner();
73                     displayBothBuildings();
74     +                     if (playerOnesGo) {
75     +                         take_turn("1");
76     +                         playerOnesGo = false;
77     +                         playerTwosGo = true;
78     +                     }
79     +                     else if (playerTwosGo) {
80     +                         take_turn("2");
81     +                         playerTwosGo = false;
82     +                         playerOnesGo = true;
83     +                     }
84     +                 }
85     +             }
86
87     +             void check_winner() {
88     +                 if(check_player_one_win()) {
89     +                     cout << "Player One wins" << endl;
90     +                     gameOver = true;
91     +                 }
92     +                 if(check_player_two_win()) {
93     +                     cout << "Player Two wins" << endl;
94     +                     gameOver = true;
95     +                 }
96     +             }

```

**Line 70-85:** As long as the game isn't over, players take turns. The game will only be over if a winner is found from `checkWinner()`.

**Line 87-96:** Print the winner if found one from the individual functions. `endl` is a C++ term to, well, end the line.

```
98 +     bool check_player_one_win() {
99 +         bool allOnes = true;
100 +        bool allZeros = true;
101 +        for(int i=0; i<4; i++) {
102 +            for(int j=0; j<5; j++) {
103 +                if(playerOneBuilding.building[i][j] != 1) {
104 +                    allOnes = false;
105 +                }
106 +                if(playerTwoBuilding.building[i][j] != 0) {
107 +                    allZeros = false;
108 +                }
109 +            }
110 +        }
111 +        return allOnes || allZeros;
112 +    }
113 +
114 +    bool check_player_two_win() {
115 +        bool allOnes = true;
116 +        bool allZeros = true;
117 +        for(int i=0; i<4; i++) {
118 +            for(int j=0; j<5; j++) {
119 +                if(playerTwoBuilding.building[i][j] != 1) {
120 +                    allOnes = false;
121 +                }
122 +                if(playerOneBuilding.building[i][j] != 0) {
123 +                    allZeros = false;
124 +                }
125 +            }
126 +        }
127 +        return allOnes || allZeros;
128 +    }
```

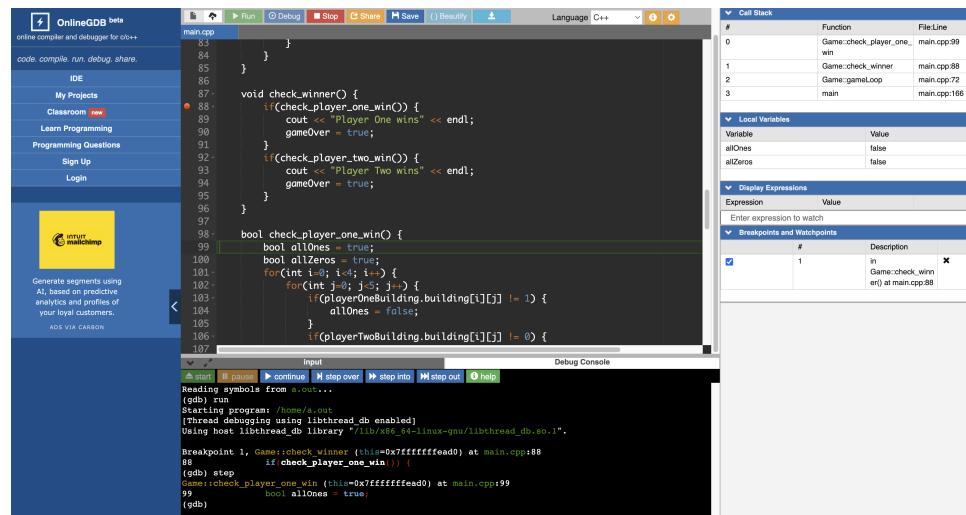
Two near identical functions. I did this originally to get the logic working and refactor later.

Loop through the player's own building to see if it is extinguished by achieving a building array of all ones in every element. Or alternatively, loop through their opponents building to see if it has been set ablaze by achieving a building array of all zeros in every element. These functions utilise good Object Oriented practices whereby I access the building array of each player's building object, seen within the if statements.

```

130 +     void take_turn(string player) {
131 +         cout << "It's Player " << player << "'s turn\n";
132 +         cout << "Press 1 to spread water in your building or press 0 to spread fire on your opponent\n";
133 +         cout << "Enter 1 or 0: ";
134 +         cin >> waterOrFire;
135 +         cout << "Which room do you chose (1-20): ";
136 +         cin >> roomNumber;
137 +         get_row_col(roomNumber, row, col);
138 +
139 +         if (player == "1") {
140 +             if (waterOrFire == 1) {
141 +                 playerOneBuilding.updateBuilding(row, col, waterOrFire);
142 +             } else if (waterOrFire == 0) {
143 +                 playerTwoBuilding.updateBuilding(row, col, waterOrFire);
144             }
145 +         } else if (player == "2") {
146 +             if (waterOrFire == 1) {
147 +                 playerTwoBuilding.updateBuilding(row, col, waterOrFire);
148 +             } else if (waterOrFire == 0) {
149 +                 playerOneBuilding.updateBuilding(row, col, waterOrFire);
150 +             }
151     }
152 }
```

Finally in this commit, I implemented the ability to take a turn, rather than inputting who's go it is. Again, the use of OO encapsulation is seen to update the player buildings.



During this stage, I tested within the “OnlineGDB” compiler as I found it to have good debugging tools to step through my logic.

## Commit 5 - Spread Water & Fire + Winner Refactor

```
43 +     void spread_water(int row, int col) {
44 +         building[row][col] = 1;
45 +         bool flag = false;
46 +         for (int i = 0; i < 4; i++) {
47 +             if (flag && building[i][col] == 1) {
48 +                 for (int waterRow = i; waterRow < 4; waterRow++) {
49 +                     building[waterRow][col] = 1;
50 +                 }
51 +                 return;
52 +             }
53 +             if (building[i][col] == 1) {
54 +                 flag = true;
55 +             }
56 +         }
57 +     }
58 +
59 +     void spread_fire(int row, int col) {
60 +         building[row][col] = 0;
61 +         bool flag = false;
62 +         for (int i = 3; i >= 0; i--) {
63 +             if (flag && building[i][col] == 0) {
64 +                 for (int fireRow = i; fireRow >= 0; fireRow--) {
65 +                     building[fireRow][col] = 0;
66 +                 }
67 +                 return;
68 +             }
69 +             if (building[i][col] == 0) {
70 +                 flag = true;
71 +             }
72 +         }
73 +     }
74 + }
```

Just like my Python and Prolog implementations, loop through the corresponding player building array and set all elements below to 1 if the room above or below is a 1 and likewise for fire and 0, but spread upwards.

```

155         if (player == "1") {
156             if (waterOrFire == 1) {
157                 // playerOneBuilding.updateBuilding(row, col, waterOrFire);
158                 playerOneBuilding.spread_water(row, col);
159             } else if (waterOrFire == 0) {
160                 // playerTwoBuilding.updateBuilding(row, col, waterOrFire);
161                 playerTwoBuilding.spread_fire(row, col);
162             }

```

With these now implemented, I needed to call them rather than `update_building()`.

```

128     // check_winner()
129     void check_winner() {
130         if(check_player_win(playerOneBuilding)) {
131             cout << "Player One wins" << endl;
132             gameOver = true;
133         }
134         if(check_player_win(playerTwoBuilding)) {
135             cout << "Player Two wins" << endl;
136             gameOver = true;
137         }
138     }
139
140     // check_player_win(+building)
141     // Pass in a player's building and see if their building is all 1s or all 0s
142     bool check_player_win(Building playerBuidling) {
143         bool allOnes = true;
144         bool allZeros = true;
145         for(int i=0; i<4; i++) {
146             for(int j=0; j<5; j++) {
147                 if(playerBuidling.building[i][j] != 1) {
148                     allOnes = false;
149                 }
150                 if(playerBuidling.building[i][j] != 0) {
151                     allZeros = false;
152                 }
153             }
154         }
155         return allOnes || allZeros;
156     }
157

```

As promised, I refactored the winner methods. I have attached a screenshot from the IDE that contains my code as the commit formatting on GitHub looks rather messy. I updated the code by merging the two check player functions into one where `check_winner` passes in the corresponding building. A clean refactor that halves the code and retains good OO utilisation.

```

Player One's Building:
1 0 0 0 1
0 1 0 1 0
0 1 0 1 0
0 0 1 0 0

Player Two's Building:
1 0 0 0 1
0 1 0 1 0
0 1 0 1 0
0 0 1 0 0

It's Player 1's turn
Press 1 to spread water in your building or press 0 to spread fire on your opponent
Enter 1 or 0: 1
Which room do you chose (1-20): 11
Player One's Building:
1 0 0 0 1
1 1 0 1 0
1 1 0 1 0
1 0 1 0 0

Player Two's Building:
1 0 0 0 1
0 1 0 1 0
0 1 0 1 0
0 0 1 0 0

```

I wanted to see if the logic works and so I ran my program, again in the “OnlineGDB”.  
 Attempting as Player 1 to spread water down, I chose room 11 as the room above is a one.  
 Selecting water and room 11, it updates all rooms directly below, success!

## Commit 6 - Input Validation

```

172 +     // choose_water_or_fire()
173 +     void choose_water_or_fire() {
174 +         cout << "Press 1 to spread water in your building or press 0 to spread fire on your opponent\n";
175 +         bool validInput = false;
176 +         while (validInput == false) {
177 +             cout << "Enter 1 or 0: ";
178 +             cin >> waterOrFire;
179 +             if (waterOrFire == 1) {
180 +                 break;
181 +                 validInput = true;
182 +             } else if (waterOrFire == 0) {
183 +                 break;
184 +                 validInput = true;
185 +             } else {
186 +                 cout << "Invalid number\n";
187 +                 validInput = false;
188 +             }
189 +         }
190 +     }
191 +
192 +     // get_row_col()
193 +     // Get the row and column from the room number the player chose
194 +     void get_row_col(int roomNumber, int& row, int& col) {
195 +         bool validInput = false;
196 +         while (validInput == false) {
197 +             cout << "Which room do you chose (1-20): ";
198 +             cin >> roomNumber;
199 +             if (roomNumber < 1 || roomNumber > 20) {
200 +                 cout << "Invalid room number\n";
201 +             } else {
202 +                 row = 3 - (roomNumber - 1) / 5;
203 +                 col = (roomNumber - 1) % 5;
204 +                 validInput = true;
205 +             }
206 +         }
207     }

```

When people play the game, they might input incorrectly. Therefore as programmers, we should think of the edge cases such as this. This is a concept we learnt in the first year of university and I have implemented it here, specifically where if the user input's something other than 1 or 0, or a number that is not in the room range of 1-20, they will be prompted to re-enter. You can notice repeated code within the two methods here, however I believe it is suitable given the time constraints and the complexity of such program.

Testing input validation

```

It's Player 1's turn
Press 1 to spread water in your building or p
ress 0 to spread fire on your opponent
Enter 1 or 0: 9
Invalid number
Enter 1 or 0: 0
Which room do you chose (1-20): 99
Invalid room number
Which room do you chose (1-20): ■

```

Entering a value that is not 1 or 0 prints the error message and ask to re-enter and likewise with room number.

## Commit 9 - Intro

```

100 +     // intro()
101 +     void intro() {
102 +         cout << "Welcome to Saviour vs Scorchern\n";
103 +         cout << "\nYOUR BUILDING IS ON FIRE\n";
104 +         cout << "And so is your opponents...\n";
105 +         cout << "You hate your opponent...\n";
106 +         cout << "You can save your building... or destroy your opponents...\n";
107 +         cout << "1 = Water";
108 +         cout << "0 = Fire\n";
109 +         cout << "Press enter to continue\n";
110 +         cin.ignore();
111 +         cout << "If you stack two 1's on top of each other, then the water will flow down to all rooms directly
below\n";
112 +         cout << "If you stack two 0's on top of each other, then the fire will flow up to all rooms directly above\n";
113 +         cout << "Press enter to continue\n";
114 +         cin.ignore();
115 +         display_both_buildings();
116 +         cout << "Press enter to continue\n";
117 +         cin.ignore();
118 +         cout << "These are the building numbers\n";
119 +         display_building_numbers();
120 +         cout << "Press enter to start\n\n";
121 +         cin.ignore();
122 +     }
123 +
124 +     // display_building_numbers()
125 +     void display_building_numbers() {
126 +         cout << (" 16. 17. 18. 19. 20.\n");
127 +         cout << (" 11. 12. 13. 14. 15.\n");
128 +         cout << (" 6. 7. 8. 9. 10.\n");
129 +         cout << (" 1. 2. 3. 4. 5.\n\n");
130 +     }
131 +

```

It may seem ironic to make the introduction of the game, at the end of development, but I focused on the core functionality in the beginning. You'll notice how this is near identical to the two prior implementations as there's not much to change between languages when it comes to simple prints to the console.

```
255     int main() {  
256         Game gameInstance;  
257     +     gameInstance.intro();  
258         gameInstance.game_loop();  
259     }
```

A minor addition to the main function where I call intro before the game loop starts.

## Commit 8 - Cleanup

```
25  
30 -     // displayBuilding()  
31 -     void displayBuilding() {  
26  
30 +     // display_building()  
31 +     void display_building() {  
27
```

A clean up commit mainly for changing names throughout the program to follow the naming conventions I mentioned earlier (camel case etc).

## Commit Summary

Through git commits, I was able to periodically add and modify functionality to my game. I started with core functionality and ended with the IO introduction, with refactoring and testing sprinkled throughout.

## Working Game

Upon completion, I tested the game fully with a friend. This time I tested it in Repl, an online compiler website for almost any language.

main.cpp

```
132 // display both buildings()
133 void display_both_buildings() {
134     cout << "Player One's Building: " << endl;
135     playerOneBuilding.display_building();
136     cout << endl;
137     cout << "Player Two's Building: " << endl;
138     playerTwoBuilding.display_building();
139     cout << endl;
140 }
141
142 // game loop()
143 void game_loop() {
144     while (!gameOver) {
145         check_winner();
146         display_both_buildings();
147         if (playerOneGo) {
148             take_turn("1");
149             playerOneGo = false;
150             playerTwoGo = true;
151         }
152         else if (playerTwoGo) {
153             take_turn("2");
154             playerTwoGo = false;
155             playerOneGo = true;
156         }
157     }
158 }
159
160 // check_winner()
161 void check_winner() {
162     if(check_player_win(playerOneBuilding)) {
163         cout << "Player One wins" << endl;
164         display_both_buildings();
165         gameOver = true;
166     }
167     if(check_player_win(playerTwoBuilding)) {
168         cout << "Player Two wins" << endl;
169         display_both_buildings();
170         gameOver = true;
171     }
172 }
173
174 // check_player_win(+building)
175 // Pass in a player's building and see if their building is all 1s or
```

Console

```
Welcome to Saviour vs Scorcher
```

YOUR BUILDING IS ON FIRE  
And so is your opponents...  
You hate your opponent.  
You can save your building... or destroy your opponents...  
1 = Water 0 = Fire  
Press enter to continue

If you stack two 1's on top of each other, then the water will flow down to all rooms directly below  
If you stack two 0's on top of each other, then the fire will flow up to all rooms directly above  
Press enter to continue

Player One's Building:  
1 0 0 0 1  
0 1 0 1 0  
0 1 0 1 0  
0 0 1 0 0

Player Two's Building:  
1 0 0 0 1  
0 1 0 1 0  
0 1 0 1 0  
0 0 1 0 0

Press enter to continue

These are the building numbers  
16. 17. 18. 19. 20.  
11. 12. 13. 14. 15.  
6. 7. 8. 9. 10.  
1. 2. 3. 4. 5.

Press enter to start

Player One's Building:  
1 0 0 0 1  
0 1 0 1 0  
0 1 0 1 0  
0 0 1 0 0

Player Two's Building:  
1 0 0 0 1  
0 1 0 1 0  
0 1 0 1 0  
0 0 1 0 0

It's Player 1's turn  
Press 1 to spread water in your building or press 0 to spread fire on your opponent  
Enter 1 or 0: 1

In the examples below, you can see how the water spreading down and fire spreading up functionality works.

# Player One Win Water

Here Player 1 has won by putting out their building as seen by all 1's in their matrix.

```
It's Player 1's turn
Press 1 to spread water in your building or p
ress 0 to spread fire on your opponent
Enter 1 or 0: 1
Which room do you chose (1-20): 11
Player One wins
Player One's Building:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

Player Two's Building:
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
```

# Player One Win Fire

Here Player 1 has won by enflaming Player 2's building as seen by all 0's in P2's matrix.

```

Player One's Building:
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 0 0 0 0

Player Two's Building:
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 0

It's Player 1's turn
Press 1 to spread water in your building or p
ress 0 to spread fire on your opponent
Enter 1 or 0: 0
Which room do you chose (1-20): 10
Player One wins
Player One's Building:
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 0 0 0 0

Player Two's Building:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

```

## Player Two Win Water

Here Player 2 has won by putting out their building as seen by all 1's in their matrix.

```

Player One's Building:
1 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 0 1 1

Player Two's Building:
1 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 1 1 1

It's Player 2's turn
Press 1 to spread water in your building or p
ress 0 to spread fire on your opponent
Enter 1 or 0: 1
Which room do you chose (1-20): 11
Player Two wins
Player One's Building:
1 1 1 1 1
0 1 1 1 1
0 1 1 1 1
0 1 0 1 1

Player Two's Building:
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

```

## Player Two Win Fire

Here Player 2 has won by enflaming Player 1's building as seen by all 0's in P2's matrix.

```
Player One's Building:  
0 0 0 0 1  
0 0 0 0 1  
0 0 0 0 1  
0 0 0 0 0  
  
Player Two's Building:  
0 1 1 1 1  
0 1 1 1 1  
0 1 1 1 1  
0 0 0 0 0  
  
It's Player 2's turn  
Press 1 to spread water in your building or p  
ress 0 to spread fire on your opponent  
Enter 1 or 0: 0  
Which room do you chose (1-20): 10  
Player Two wins  
Player One's Building:  
0 0 0 0 0  
0 0 0 0 0  
0 0 0 0 0  
0 0 0 0 0  
  
Player Two's Building:  
0 1 1 1 1  
0 1 1 1 1  
0 1 1 1 1  
0 0 0 0 0
```

## Conclusion

C++ is a language I had no prior experience with and little Object Oriented knowledge. While the heavy lifting of the overall structure and logic had already been created through my Python implementation, the C++ code I have produced and documented shows the use of strong OO practices and knowledge. I made good use of encapsulation often, object creation, and constructors, as well as composition, throughout the program.