Python Design

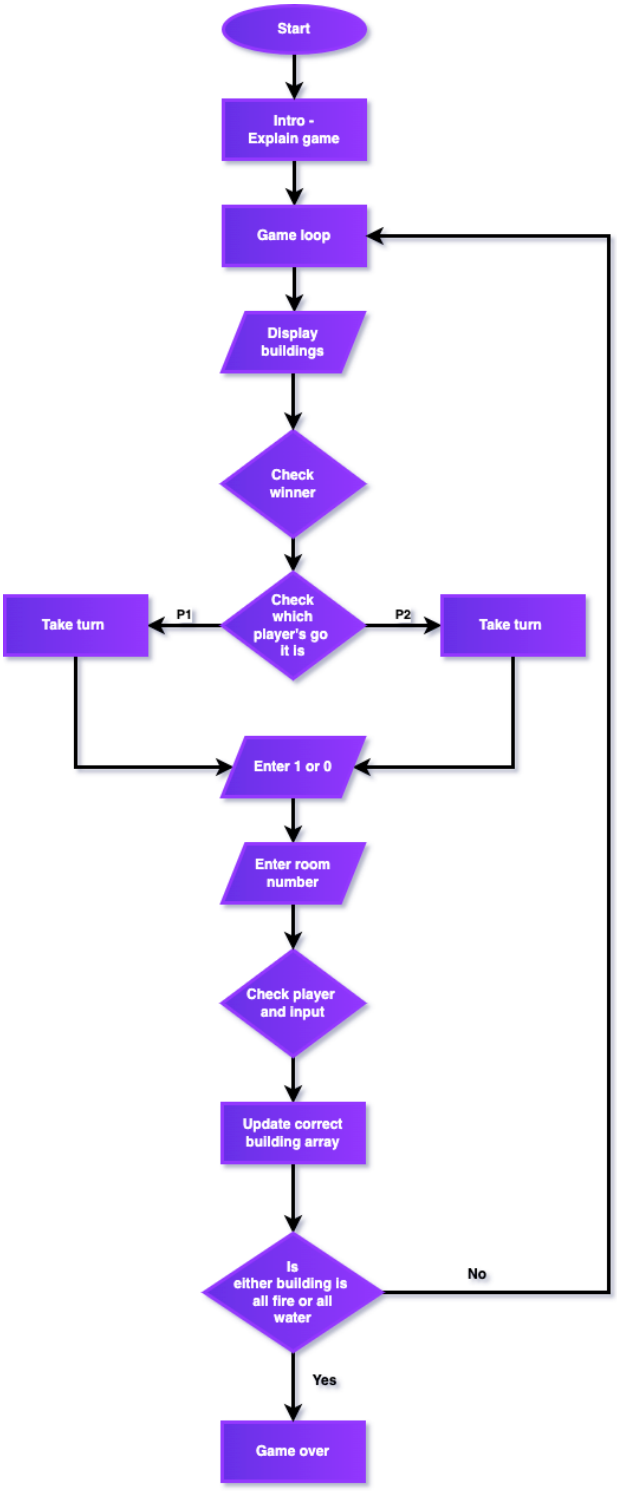
This is the development document for the solution to my game, using the Python programming language and the procedural paradigm.

This was the first language and paradigm I decided to use.

Pre-implementation

Before creating my game in the first language I chose, I decided it would be best to create a diagram to think of the logical steps. Before diving in.

Flow chart of the game process


Start
Intro - Explain game
Game loop
Display buildings
Check winner
Check which player's go it is
P1
P2
Take turn
Take turn
Enter 1 or 0
Enter room number
Check player and input
Update correct building array
Is either building is all fire or all water
No
Yes
Game over

This flow chart shows my thinking. Start the game by telling the players how to play. Loop the play of the game. Show the buildings to the players. Check if a player has won. Check who’s turn it is. Player chose fire or water. Player enters room they want to spread to. Check which player selected what. Update the building corresponding to what the player chose. Check if the either building array contains all zeros or all ones. If not, loop the play of the game again, otherwise, end the game.

Implementation

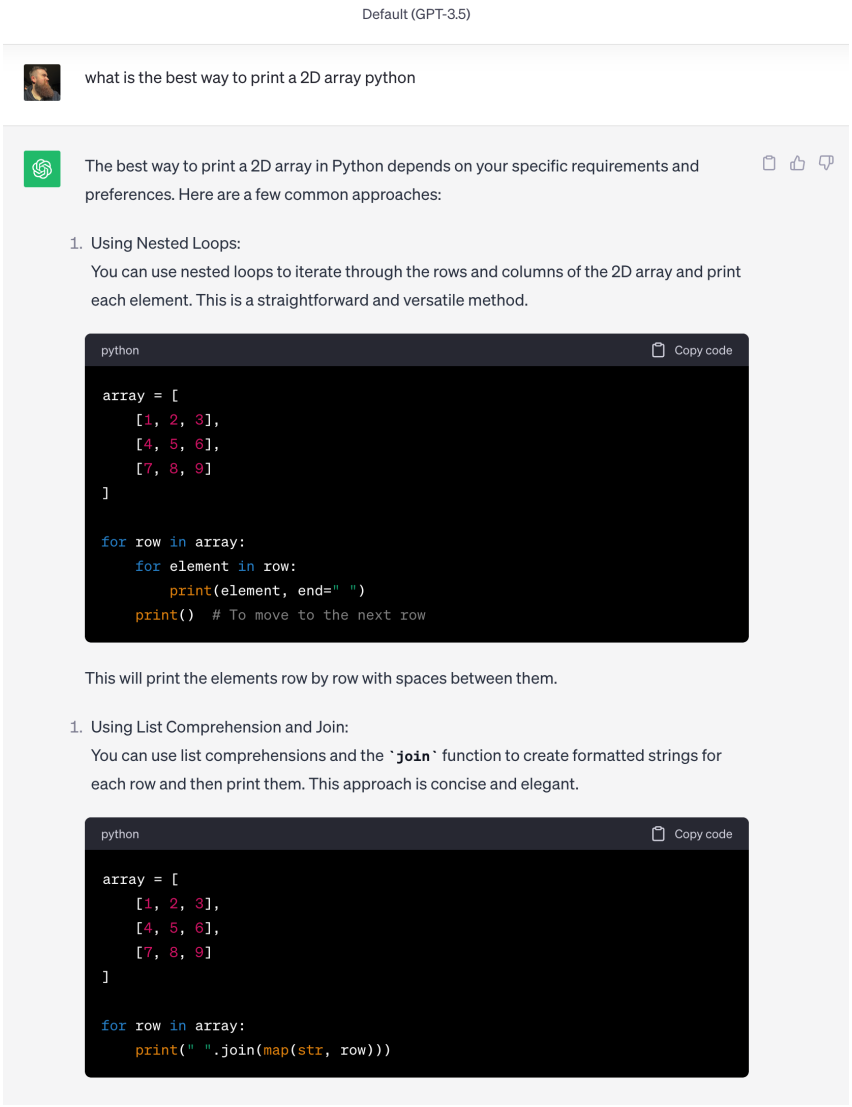
I created a private GitHub repository for the code. With this I could commit new code, changes and updates and rollback if later features causes game breaking issues.

Commit 1 - Print Buildings

```
29  + # Player one building array
30  + playerOneBuilding = [[ 1, 0, 0, 0, 1],    # 16. 17. 18. 19. 20.
31  +                      [ 0, 1, 0, 1, 0],    # 11. 12. 13. 14. 15.
32  +                      [ 0, 1, 0, 1, 0],    # 6.  7.  8.  9.  10.
33  +                      [ 0, 0, 1, 0, 0]]    # 1.  2.  3.  4.  5.
34  +
35  + # Player two building array
36  + playerTwoBuilding = [[ 1, 0, 0, 0, 1],    # 16. 17. 18. 19. 20.
37  +                      [ 0, 1, 0, 1, 0],    # 11. 12. 13. 14. 15.
38  +                      [ 0, 1, 0, 1, 0],    # 6.  7.  8.  9.  10.
39  +                      [ 0, 0, 1, 0, 0]]    # 1.  2.  3.  4.  5.
40  +
41  + # Set the building
42  + building = ["-"] * 20
43  +
44  + # display_playerone_building()
45  + # Use the previously made building array of "-"s to display the building
46  + def display_playerone_building():
47  +     # print(building[0],  building[1],  building[2],  building[3],  building[4])  # - - - - -
48  +     # print(building[5],  building[6],  building[7],  building[8],  building[9])  # - - - - -
49  +     # print(building[10], building[11], building[12], building[13], building[14]) # - - - - -
50  +     # print(building[15], building[16], building[17], building[18], building[19]) # - - - - -
51  +     print("Player 1's building")
52  +     for row in playerOneBuilding:
53  +         print(' '.join(map(str, row)))
54  +     print("\n")
55  +
56  + # display_playertwo_building()
57  + # Use the previously made building array of "-"s to display the building
58  + def display_playertwo_building():
59  +     # print(building[0],  building[1],  building[2],  building[3],  building[4])  # - - - - -
60  +     # print(building[5],  building[6],  building[7],  building[8],  building[9])  # - - - - -
61  +     # print(building[10], building[11], building[12], building[13], building[14]) # - - - - -
62  +     # print(building[15], building[16], building[17], building[18], building[19]) # - - - - -
63  +     print("Player 2's building")
64  +     for row in playerOneBuilding:
65  +         print(' '.join(map(str, row)))
66  +     print("\n")
```

Line 30-42: Beginning with the basics, I set up the player building arrays. One for Player 1 and one for Player 2. Here I also implemented similar code to what we created in the lab, making a building of many dashes ‘-’.

Line 46-66: For printing the building arrays to the console, I consulted ChatGPT as I was unfamiliar to Python.



I expected it to explain steps but as it provided a code example, I opted for the second approach as it was clearly the cleaner solution. Although the solution was provided for me, my understanding of it was clear. The code loops through the building array row by row, converts each room number to a string and then concatenates those strings together with a blank space between them.

I had two procedures that repeated code but I decided to implement first and refactor later.

Commit 2 - Starting Winner

```
68  + # playerOneWin()
69  + def playerOneWin():
70  +     if playerOneBuilding == [[ 1, 1, 1, 1, 1],
71  +                              [ 1, 1, 1, 1, 1],
72  +                              [ 1, 1, 1, 1, 1],
73  +                              [ 1, 1, 1, 1, 1]] or playerTwoBuilding == [[ 0, 0, 0, 0, 0],
74  +                                                                         [ 0, 0, 0, 0, 0],
75  +                                                                         [ 0, 0, 0, 0, 0],
76  +                                                                         [ 0, 0, 0, 0, 0]]:
77  +         playerOneWinner = True
78  +     return playerOneWinner
79  +
80  + # playerTwoWin()
81  + def playerTwoWin():
82  +     if playerTwoBuilding == [[ 1, 1, 1, 1, 1],
83  +                              [ 1, 1, 1, 1, 1],
84  +                              [ 1, 1, 1, 1, 1],
85  +                              [ 1, 1, 1, 1, 1]] or playerOneBuilding == [[ 0, 0, 0, 0, 0],
86  +                                                                         [ 0, 0, 0, 0, 0],
87  +                                                                         [ 0, 0, 0, 0, 0],
88  +                                                                         [ 0, 0, 0, 0, 0]]:
89  +         playerTwoWinner = True
90  +     return playerTwoWinner
91  +
92  + # checkWinner()
93  + def checkWinner():
94  +     if playerOneWin() == True:
95  +         print("Player 1 wins\n")
96  +         gameOver = True
97  +     elif playerTwoWin() == True:
98  +         print("Player 2 wins\n")
99  +         gameOver = True
```

Line 69-90: Player 1 wins if their building is all 1's (water) or if P2's building is all 0's (fire). And the opposite for Player 2. These are both functions however they are used within the procedural paradigm and follow the concept of execution through a series of tasks.

Line 93-99: This currently sets the variable of `gameOver` to be true. This will be used later.

Commit 3 - Turns in Game Loop

```
+ # take_turn(+player)
+ def take_turn(player):
+     print("It's player " + player + "'s turn\n")
+     print("Press 1 to save your building or press 0 to spread fire on your opponent...\n")
+     waterOrFire = int(input("Enter 1 or 0: \n"))
+     move = int(input("Enter move index (1-20): \n")) # get the move index position
+     if player == "1" and waterOrFire == 1:
+         playerOneBuilding[move-1] = 1
+     elif player == "1" and waterOrFire == 0:
+         playerTwoBuilding[move-1] = 0
+     elif player == "2" and waterOrFire == 1:
+         playerTwoBuilding[move-1] = 1
+     elif player == "2" and waterOrFire == 0:
+         playerOneBuilding[move-1] = 0
+
+ def intro():
+     print("YOU'RE BUILDINGS ON FIRE\n")
+     print("And so is your opponents...\n")
+     print("You hate your opponet...\n")
+     print("You can save your building... or destroy your opponents...\n")
+     print("1 = No fire\n")
+     print("0 = Fire!\n")
+
+ # game_loop()
+ def game_loop():
+     global playerOneWinner, playerTwoWinner, playerOnesGo, playerTwosGo, gameOver
+     if gameOver == False:
+         display_playerone_building()
+         display_playertwo_building()
+         if playerOnesGo == True:
+             take_turn("1")
+             checkWinner()
+             playerOnesGo = False
+             playerTwosGo = True
+         elif playerTwosGo == True:
+             take_turn("2")
+             checkWinner()
+             playerTwosGo = False
+             playerOnesGo = True
+     elif gameOver == True:
+         print("Game over\n")
+
+ # main()
+ def main():
+     game_loop()
+
+ # Run the main function
+ # Environment variables
+ if __name__ == "__main__":
+     main()
```

Line 102-114: The take turn procedure that gets two inputs from the current player - water or fire and which room they choose. The if and else logic here is incorrect but it lays out the foundation for the future. My logic is wrong as I was trying to set a value in an array, not a 2D matrix, with the added problem that my building is 1-20 from bottom to top when the matrix will be top to bottom. Logic to fix at a later date.

Line 116-122: The intro to the game, simple print statements.

Line 125-141: The loop of the game. I made global variables here that I knew I would use throughout the program. Starting with ‘gameOver’ to false display the buildings and take turns. If there’s a winner, get ‘gameOver’ to true and end.

Line 144-150: The main procedure `main` calls the game loop procedure which now runs on execution as ‘main’ is called from the Python idiom which checks whether the script is being run as the main program. This is the first procedural paradigm concept in the program.

Commit 4 - Get Row and Col

```
+ def take_turn(player, row, col):
      print("It's player " + player + "'s turn\n")
      print("Press 1 to save your building or press 0 to spread fire on your opponent...\n")
      waterOrFire = int(input("Enter 1 or 0: \n"))
      move = int(input("Which room do you chose (1-20): \n")) # get the move index position
      if player == "1" and waterOrFire == 1:
+         row, col = update_building(move, row, col)

+         playerOneBuilding[row][col] = 1 # User enters 1-20 where 1 is the bottom of the building and 20 is the top so we need to reverse
    the index (19 - flat number) and then - 1 as array is 0-19 not 1-20
+         display_playerone_building()
      elif player == "1" and waterOrFire == 0:
+         row, col = update_building(move, row, col)
+         playerTwoBuilding[row][col] = 0
+         display_playertwo_building()
      elif player == "2" and waterOrFire == 1:
+         row, col = update_building(move, row, col)
+         playerTwoBuilding[row][col] = 1
+         display_playertwo_building()
      elif player == "2" and waterOrFire == 0:
+         row, col = update_building(move, row, col)
+         playerOneBuilding[row][col] = 0
+         display_playerone_building()
+
+ # update_building(+move, -row, -col)
+ def update_building(move, row, col):
+     if 1 <= move <= 20:
+         row = 3 - (move - 1) // 5
+         col = (move - 1) % 5
+     return row, col
```

This code introduced my solution to get the room position to update. The player selects water or fire and a room to spread to. They chose a number 1-20. You can visualise the buildings as a 5x4 graph where room 1 is position (0,0) and room 20 as (5,4). Therefore we need to know the row and column just as the graph is (x,y), the matrix is (row, col).

Line 130-134: This is a function that returns the row and col based off the room selected. While this exhibits a more functional implementation, it it called within the procedural paradigm. I have provided an example room number of 13 below.

Row: roomNumber = 13. 13 - 1 = 12 // 5 = 2 . Therefore row = 3 - 2 = 1 which is the second row.

Column: roomNumber = 13. 13 - 1 = 12 % 5 = 2. Therefore col = 2 which is the third column

As you can see, the maths is quite simple but effective and efficient.

```
159     # game_loop()
160     def game_loop():
161 +       global playerOneWinner, playerTwoWinner, playerOnesGo, playerTwosGo, gameOver, row, col
162         if gameOver == False:
163             display_playerone_building()
164             display_playertwo_building()
165             if playerOnesGo == True:
166 +               take_turn("1", row, col)
167 +               checkWinner(gameOver, playerOneWinner, playerTwoWinner)
168                 playerOnesGo = False
169                 playerTwosGo = True
170             elif playerTwosGo == True:
171 +               take_turn("2", row, col)
172 +               checkWinner(gameOver, playerOneWinner, playerTwoWinner)
173                 playerTwosGo = False
174                 playerOnesGo = True
175         elif gameOver == True:
176             print("Game over\n")
177
```

With this change, I now pass in row and col to the `take_turn` procedure.

```
136 + # intro()
137   def intro():
138 +     print("\nYOU'RE BUILDING IS ON FIRE\n")
139       print("And so is your opponents...\n")
140       print("You hate your opponet...\n")
141       print("You can save your building... or destroy your opponents...\n")
142 +     print("1 = No fire")
143       print("0 = Fire!\n")
144 +     print(input("Press enter to start\n"))
145 +     display_playerone_building()
146 +     display_playertwo_building()
147 +     print(input("Press enter to continue\n"))
148 +     print("These are the building numbers\n")
149 +     display_building_numbers()
150 +     print(input("Press enter to continue\n"))
151 +
152 + # display_building_numbers()
153 + def display_building_numbers():
154 +     print(" 16.  17.  18.  19.  20.")
155 +     print(" 11.  12.  13.  14.  15.")
156 +     print(" 6.   7.   8.   9.   10.")
157 +     print(" 1.   2.   3.   4.   5.\n")
```

```
180     def play():
181         intro()
182         game_loop()
```

Minor addition for making `intro` run on start in `play` and added a procedure to print the building room numbers.

Commit 5 - Above and Below & Cleanup

Starting with cleanup.

```
--
22    # Globle variables
23  + playerOneWins = False
24  + playerTwoWins = False
25    playerOnesGo = True
26    playerTwosGo = False
27    gameOver = False
```

I defined global variables at the top of the file.

```
111  + # display_both_buildings()
112  + def display_both_buildings():
113  +     display_playerone_building()
114  +     display_playertwo_building()
```

A simple procedure to call for displaying both buildings, rather than calling individually multiple times.

```
148 + # above_and_below(+player, +waterOrFire, +row, +col)
149 + # Spread water or fire (1 or 0) below or above the room selected by the user, if the toom below or above is already water or fire (1 or 0)
150 + def above_and_below(player, waterOrFire, row, col):
151 +     # As long as the user doesn't select water for the bottom row or fire for the top row
152 +     # As water cant spread to the bottom if we're already at the bottom
153 +     # and fire cant spread to the top if we're already at the top
154 +     if not(waterOrFire == 1 and row == 3) or (waterOrFire == 0 and row == 0):
155 +         # If p1 selects 1 (water) and the row below is 1 (water) then set all rows below to 1 (water)
156 +         if ((player == "1" and waterOrFire == 1) and (playerOneBuilding[row + 1][col] == 1 or playerOneBuilding[row - 1][col] == 1)):
157 +             # set all rows below row+1 to 1 in a loop
158 +             for i in range(row + 1, len(playerOneBuilding)):
159 +                 playerOneBuilding[i][col] = 1
160 +         # If p1 selects 0 (fire) and the row above is 0 (fire) then set all rows above to 0 (fire)
161 +         elif ((player == "1" and waterOrFire == 0) and (playerTwoBuilding[row + 1][col] == 0 or playerTwoBuilding[row - 1][col] == 0)):
162 +             # set all rows above row-1 to 0 in a loop
163 +             for i in range(row - 1, -1, -1):
164 +                 playerTwoBuilding[i][col] = 0
165 +         # If p2 selects 1 (water) and the row below is 1 (water) then set all rows below to 1 (water)
166 +         elif ((player == "2" and waterOrFire == 1) and (playerTwoBuilding[row + 1][col] == 1 or playerTwoBuilding[row - 1][col] == 1)):
167 +             # set all rows below row+1 to 1 in a loop
168 +             for i in range(row + 1, len(playerTwoBuilding)):
169 +                 playerTwoBuilding[i][col] = 1
170 +         # If p2 selects 0 (fire) and the row above is 0 (fire) then set all rows above to 0 (fire)
171 +         elif ((player == "2" and waterOrFire == 0) and (playerOneBuilding[row + 1][col] == 0 or playerOneBuilding[row - 1][col] == 0)):
172 +             # set all rows above row-1 to 0 in a loop
173 +             for i in range(row - 1, -1, -1):
174 +                 playerOneBuilding[i][col] = 0
```

A main component of the game is the ability for water or fire to spread if stacked. If the player selects a room that has the same value above it or below it, then spread

water down or fire up. This code you can see, got quite messy.

Line 154: The logic being, if water is selected (1) and we're on the bottom (row 3), then we can't spread water below. Likewise for fire if on the top.

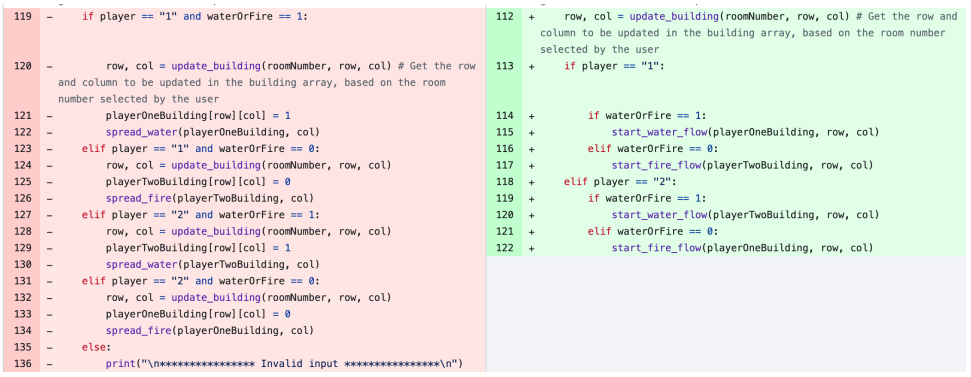
Line 156-174: Complicated, repeated code checking which player selected what and then attempting to spread. The problem being: repetition, duplication, messiness, and that it wouldn't update if the selected room is on top or bottom row - missing functionality.

Commit 5 - Refactor

```
91  - # display_playerone_building()
92  - # Print player one's building
93  - def display_playerone_building():
94  -     print("\nPlayer 1's building")
95  -     for row in playerOneBuilding:
96  -         print(' '.join(map(str, row)))
97  -     print("\n")
98  -
99  - # display_playertwo_building()
100 - # Print player two's building
101 - def display_playertwo_building():
102 -     print("Player 2's building")
103 -     for row in playerTwoBuilding:
104             print(' '.join(map(str, row)))
105     print("\n")
106
107   # display_both_buildings()
108   def display_both_buildings():
109 -     display_playerone_building()
110 -     display_playertwo_building()
```

```
91  + # display_building(+buildingArray)
92  + # Print the building array with a given player's building
93  + def display_building(buildingArray):
94  +     for row in buildingArray:
95              print(' '.join(map(str, row)))
96      print("\n")
97
98    # display_both_buildings()
99    def display_both_buildings():
100 +     print("Player 1's building\n")
101 +     display_building(playerOneBuilding)
102 +     print("Player 2's building\n")
103 +     display_building(playerTwoBuilding)
```

A rector to how I displayed the buildings. The left shows my prior solution where I had two procedures doing the same functionality, but for different arrays. The right shows the much simpler version where it takes in a building array. This reduces the code by half and has no duplication.



```
119 -     if player == "1" and waterOrFire == 1:
120 -         row, col = update_building(roomNumber, row, col) # Get the row and column to be updated in the building array, based on the room number selected by the user
121 -         playerOneBuilding[row][col] = 1
122 -         spread_water(playerOneBuilding, col)
123 -     elif player == "1" and waterOrFire == 0:
124 -         row, col = update_building(roomNumber, row, col)
125 -         playerTwoBuilding[row][col] = 0
126 -         spread_fire(playerTwoBuilding, col)
127 -     elif player == "2" and waterOrFire == 1:
128 -         row, col = update_building(roomNumber, row, col)
129 -         playerTwoBuilding[row][col] = 1
130 -         spread_water(playerTwoBuilding, col)
131 -     elif player == "2" and waterOrFire == 0:
132 -         row, col = update_building(roomNumber, row, col)
133 -         playerOneBuilding[row][col] = 0
134 -         spread_fire(playerOneBuilding, col)
135 -     else:
136 -         print("\n********** Invalid input **********\n")
```

```
112 +     row, col = update_building(roomNumber, row, col) # Get the row and column to be updated in the building array, based on the room number selected by the user
113 +     if player == "1":
114 +         if waterOrFire == 1:
115 +             start_water_flow(playerOneBuilding, row, col)
116 +         elif waterOrFire == 0:
117 +             start_fire_flow(playerTwoBuilding, row, col)
118 +     elif player == "2":
119 +         if waterOrFire == 1:
120 +             start_water_flow(playerTwoBuilding, row, col)
121 +         elif waterOrFire == 0:
122 +             start_fire_flow(playerOneBuilding, row, col)
```

The logic for which player and the ability they select was messy. Now is clear and cleaner. This calls `start_water_flow` and `start_fire_flow` but the functionality currently isn't working.

```
+ # start_water_flow(+buildingArray, +row, +col)
+ def start_water_flow(buildingArray, row, col):
+     buildingArray[row][col] = 1
+     spread_water(buildingArray, col)
+
+ # start_fire_flow(+buildingArray, +row, +col)
+ def start_fire_flow(buildingArray, row, col):
+     buildingArray[row][col] = 0
+     spread_fire(buildingArray, col)
+
```

Commit 6 - Spread Water & Fire Refactor

```
# spread_water(+buildingArray, +row, +column)
# Pass in the building (p1's or p2's) and in the column for the room number index the player selected
# If player selected water (1) and there is a 1 above or below (two 1's on top of each other)
# then spread the water down the column
def spread_water(buildingArray, row, column):
    buildingArray[row][column] = 1
    flag = False
    # loop for each row
    for row in range(0, 4):
        # check the flag
        if(flag == True and buildingArray[row][column] == 1):
            # flow the water down the building column
            for waterRow in range(row, 4):
                buildingArray[waterRow][column] = 1 # set that row and column index to 1
            return
        # if we are water then flag true
        if(buildingArray[row][column] == 1):
            flag = True

# spread_fire(+buildingArray, +row, +column)
# Pass in the building (p1's or p2's) and in the column for the room number index the player selected
# If player selected fire (0) and there is a 0 above or below (two 0's on top of each other)
# then spread the fire up the column
def spread_fire(buildingArray, row, column):
    buildingArray[row][column] = 0
    flag = False
    # loop for each row
    for row in range(3, 0, -1):
        # check the flag
        if(flag == True and buildingArray[row][column] == 0):
            # flow the fire up the building column
            # range(the current row, down to including 0, down 1 each loop)
            for fireRow in range(row, -1, -1): # start at row and go down to -1 (down to 0 as python ra
                buildingArray[fireRow][column] = 0
            return
        # if we are fire then flag true
        if(buildingArray[row][column] == 0):
            flag = True
```

Above you can see my refactor for spreading water and fire. While a further refactor would be ideal to limit repeated code, this solution is fully functional and the best I have achieved in the scope of the project. Pass in the building array with the row and col previously calculated. Where water flows down from row 0 to 3 (top to bottom), fire does flows from 3 to 0. You'll notice 0 to 4 and 3 to -1. This is due to Python's 'range' ability, gos up to but not including the number you state, so we must go to one position further on each.

Commit 7 - Error Handling

As I am unfamiliar with Python and it's syntax, I asked chatGPT about finding out whether a user's input is an integer.

python, how to check user input is an integer and error if not

You can check if user input is an integer and handle errors if it's not by using a `try...except` block in Python. Here's an example of how to do this:

python Copy code

```
try:
    user_input = int(input("Enter an integer: "))
    # If the input is successfully converted to an integer, you can use 'use
except ValueError:
    print("Invalid input. Please enter an integer.")
```

In this code, we try to convert the user's input to an integer using `int(input("Enter an integer: "))`. If the user enters something that cannot be converted to an integer, it will raise a `ValueError` exception, and the code inside the `except` block will be executed, printing an error message.

Again, it did provide a code snippet which I modified and expanded.

```
+ def check_integer(number):
+     while True:
+         player_input = input(number)
+         try:
+             return int(player_input)
+         except ValueError:
+             print("**************** Invalid input ****************\n")
```

I added the functionality to loop for as long as the user input's something other than a number.

Commit 8 - Final Clean Up

```
56        print("You can save your building... or destroy your opponents...\n")
57        print("1 = Water")
58        print("0 = Fire\n")
59  +     print(input("Press enter to continue\n"))
60  +     print("If you stack two 1's on top of each other, then the water will flow
      down to all rooms directly below\n")
61  +     print("If you stack two 0's on top of each other, then the fire will flow up
      to all rooms directly above\n")
62  +     print(input("Press enter to continue\n"))
63        display_both_buildings()
64        print(input("Press enter to continue\n"))
65        print("These are the building numbers\n")
66        display_building_numbers()
67  +     print(input("Press enter to start\n"))
68
```

To finish, I added some minor changes to the code for a better experience.

Commit Summary

As demonstrated in this section, I periodically added functionality to my game. Whilst I have not chosen to include every minute insertion, I have ensured the key additions, changes, and tests are present. Hopefully you have a good understanding of the process I undertook.

Working Game

Upon completion, I tested the game fully with a friend. We played the game multiple times, running in the terminal locally, in Visual Studio Code.

Player One Win Water

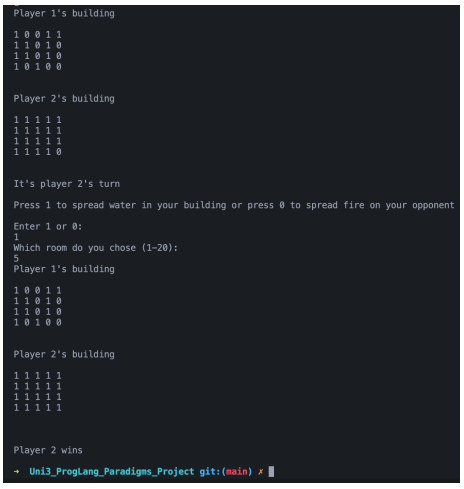
Here Player 1 has won by putting out their building as seen by all 1's in their matrix.

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.

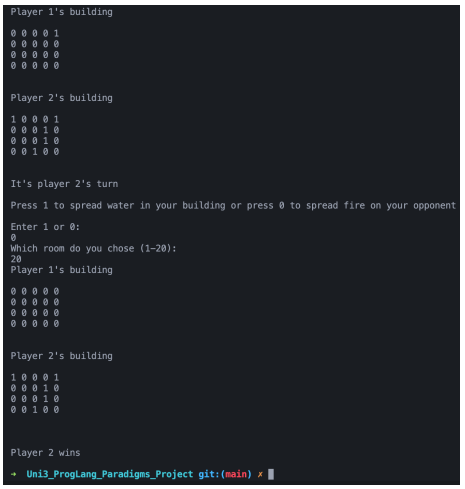
Player Two Win Water

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



Player Two Win Fire

Here Player 2 has won by enflaming Player 1's building as seen by all 0's in P1's matrix.



Conclusion

The Python solution to my game makes good use of the procedural paradigm. The code is clean, easy to read, and well documented. The procedural paradigm is used

from the start with a main procedure calling two other procedures. During the creation, I made use of the LLM tool, ChatGPT for two basic queries, predominately due to my lack of knowledge in this language. I ensured my use of the tool was limited and relied on my own intuition to complete the majority of the task. I am happy with my solution and think the game works great.