WORKSHOP #9 - DUNGEON

## APS145NFF - Group 2 Members:

* Nasita - Logic 1 - Flowchart
* Joshua Civello - Logic 1 - Pseudocode
* Seulgi Lee - Logic 2 - Flowchart
* Md Ridwan Hossain - Logic 2 - Pseudocode
* Luca Novello - Logic 3 - Pseudocode
* Luca Novello - Logic 3 - Flowchart

## Workshop Overview

It is the post-apocalyptic era, and hardship and depression runs rampant. You are currently at the bottom of a staircase in the basement of an abandoned building looking at what is pure pitch-black darkness ahead (this is the starting point of your journey). You've heard there are some uplifting discoveries to be made in the rooms found in the depths of this building - but this building has a very complex maze of hallways where many who have attempted exploration have not returned. The only way to ensure your safety is to use your flashlight and manage your time carefully so you do not venture to the point of no return!

The flashlight is your lifeline (as is your smarts). Without these things, you will never escape, so it is vital you monitor the flashlight’s remaining power to determine how much of the basement you can discover before having to turn around and head back to safety (return to the bottom of the staircase where you started). It is equally vital you keep accurate records of all your movements so you can retrace your steps and return without getting lost!

Your objective is to explore as many rooms as possible and maximize the use of the battery time that remains for your flashlight. The journey is only successful if you can return to the staircase landing before you are forever left in the dark!

## Workshop Details

* Your flashlight battery has only 4.5 hours of life
* All hallways are forty (40) steps long
* Each “[***Move***]” unit is 4 steps in distance and takes 1.5 minutes in time
* You must “[***CheckRoom***]” after each “[***Move***]” command until you decide to “[***TurnAround***]”
* You can only travel in the forward direction
* At the end of a hallway, you must decide to turn left or right (use “[ChangeDirection]”).   
  ***Note****: This is an endless labyrinth with no known end – each hallway offers two directions at* ***each end****!*
* You must track your moves to know when you are at the end of a hallway (and when to make a directional change)
* You can only call the “[***TurnAround***]” command **ONCE** (this marks the beginning of your return back to the beginning).
* You can call the “[***TurnAround***]” command at any time but it should only be done when you have maximized the use of your flashlight battery and can return safely to the beginning
* On the return back to the beginning, you must successfully reverse navigate your recorded journey and before the flashlight battery dies.
* You can’t “[***CheckRoom***]” to explore rooms when you are on your return journey back to the beginning.

# DATA STRUCTURES

| DUNGEON | |
| --- | --- |
| flashlightBattery | *In minutes, initial battery life* |
| totalTime | *In minutes, total time in dungeon including rooms and travel* |
| explorationTime | *In minutes, total travel time in dungeon including rooms and travel* |
| totalDistance | *In minutes, total distance traveled* |

# 

| ROOM | |
| --- | --- |
| totalTime | *In minutes, total time spent in room* |
| maxTimeRoomA | *Worst case scenario for total time in Room A* |
| maxTimeRoomB | *Worst case scenario for total time in Room B* |

# PSEUDOCODE:

## Main Pseudocode:

1. **Start**
2. Create and **Initialize variables**:
   1. **flashlightBattery = 270**
   2. **totalTime = 0**
   3. **explorationTime = 0**
   4. **totalDistance = 0**
3. **Import and store values** for:
   1. **maxTimeRoomA**
   2. **maxTimeRoomB**
4. Is (**flashlightBattery - totalTime**) > (**flashlightBattery / 2**)?
   1. **Yes**:call **TurnAround()** & **return to start** -> **End**
   2. **No**: continue -> **Step 5**
5. **Move**
6. **Add 1.5 to explorationTime** & **add 4 to totalDistance**
7. Call **CheckRoom()**
8. Is the return value "**Room-A**"?
   1. **Yes**: continue -> **Step 9**
   2. **No**: continue -> **Step 5**
9. Is (**flashlightBattery - totalTime**) >= (**totalTime + maxTimeRoomA**)?
   1. **Yes**: go to **Logic 2** -> **Step 12**
   2. **No**: continue -> **Step 4**
10. Is the return value "**Room-B**"?
    1. **Yes**: continue -> **Step 9**
    2. **No**: continue -> **Step 13**
11. Is (**flashlightBattery - totalTime**) >= (**totalTime + maxTimeRoomB**)?
    1. **Yes**: go to **Logic 3** -> **Step 12**
    2. **No**: continue -> **Step 4**
12. Add **Room.totalTime** to **totalTime**
13. Is it the **end of the hallway**?
    1. **Yes**: Call **ChangeDirection(left/right)** -> **Step 5**
    2. **No**: continue -> **Step 5**

## Logic 1 - Pseudocode:

1. Initialize variables

flashlightBattery = 4.5; (hours)

totalTime = 0;

totalDistance = 0;

exploreTime = 0;

averageExploreTime = 0;

longestExploreTime = 0;

explorationRiskFactor = 0;

2) Define ExploreRoom-A

1. Play X's and O's game
2. Initialize game
3. Play 5 games
4. Update exploration time based on game outcomes

3) Define ExploreRoom-B

1. Play Hang-Person game
2. Generate a secret word
3. Allow maximum of 8 letter guesses
4. Provide option to guess the secret word at any time
5. Update exploration time based on game outcomes

4) Define ChangeDirection(direction)

1. Rotate direction 90 degrees (left or right)

5) Define TurnAround

1. Rotate direction 180 degrees
2. Start return journey
3. Stop exploring rooms
4. Calculate total time and distance for return journey
5. Output time analysis summary
6. End program

6) Define main logic for exploring the dungeon

1. Loop while flashlight\_battery > 0
2. Move forward one unit
3. Update total\_distance\_traveled
4. Update total\_time\_explored
5. CheckRoom
6. If Room-A
7. ExploreRoom-A
8. Else if Room-B
9. ExploreRoom-B

2) Else

1. Continue exploration
2. Update exploration time metrics

3) If exploring\_room

1. Update room\_explore\_time

3) Calculate average explore time and longest explore time

1. Set average\_explore\_time as the average of room\_explore\_time
2. Set longest\_explore\_time as the maximum of room\_explore\_time

4) Calculate pessimistic risk factor

1. Set exploration\_risk\_factor as the sum of average\_explore\_time and longest\_explore\_time

5) Check if it's time to turn around

1. If total\_time\_explored + exploration\_risk\_factor + (total\_distance\_traveled \* 1.5) > flashlight\_battery

6.1.6.1.1 TurnAround

6) If flashlight runs out before turning around

a) Output "You didn't return in time and are now forever lost in darkness."

1. End program

## Logic 2 - Pseudocode:

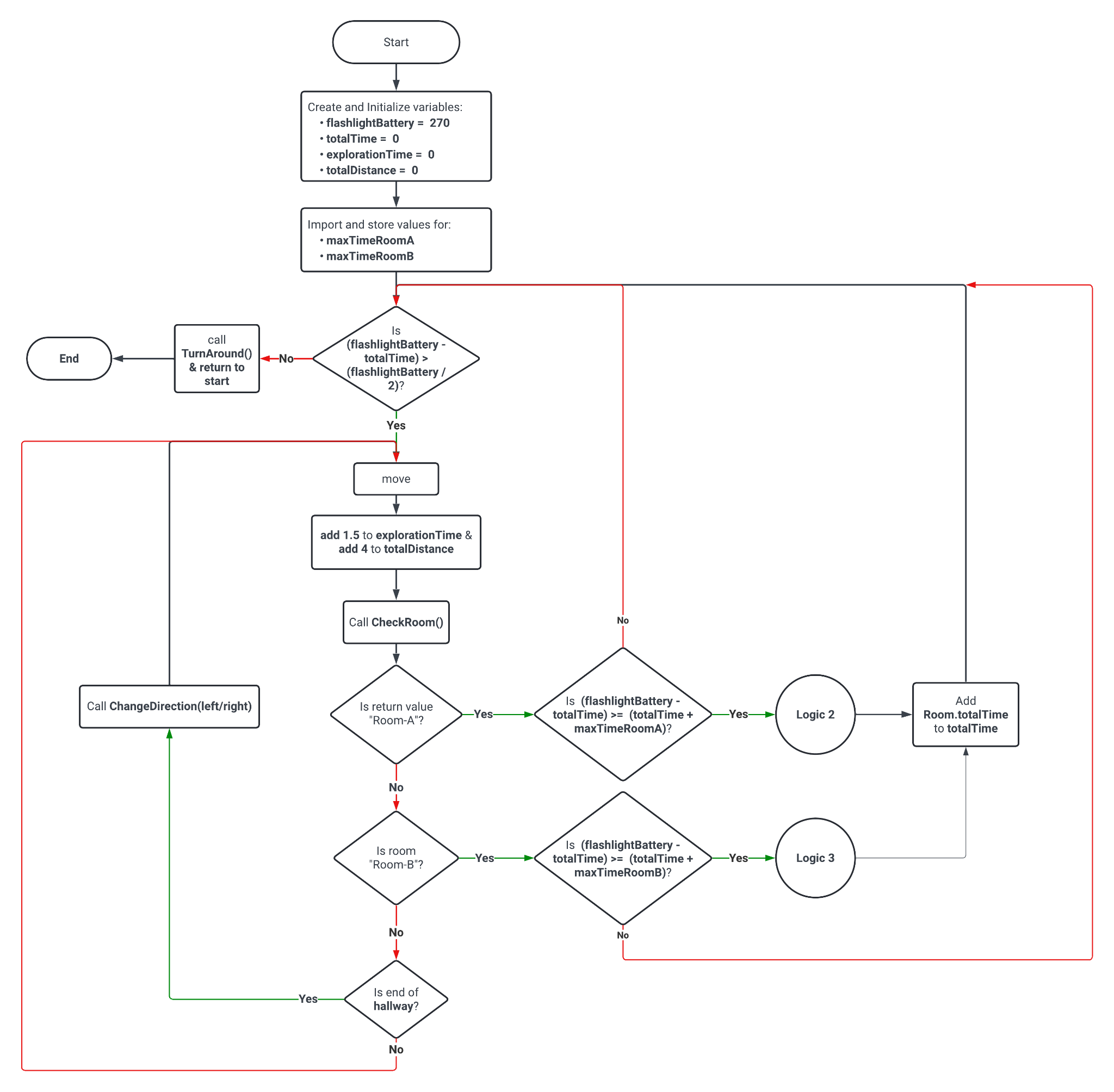
1. Start/Previous Logic
2. Import & store Dungeon (structure)
3. Create variables **“totalGames, winner, currentPlayer, player1Symbol, player2Symbol”;**
4. Initialize totalGames = 0, dungeon.exploredTime = 0, winner = "";
5. Consider humanPlayer as Player-1 and computerPlayer as Player-2.
6. **Did totalGames** == **0**?
   1. **Yes**:First game -> **Step 7**
   2. **No**: Other game -> **Step 8**
7. Call **RandomBinary()** and return 1?
   1. **Yes**:Set currentPlayer = “player1”; **-> Step 9**
   2. **No**: Set currentPlayer = “player2”; **-> Step 9**
8. **Did winner != ""?**  // is there previos winner ? (not tie)
   1. **Yes**:Set currentPlayer = winner;
   2. **No**: **Step 7**
9. Prompt humanPlayer to select their symbol (X or O)
10. Did Player1 choose the symbol 'X'?
    1. **Yes**:Set player1Symbol = ‘X’, player2Symbol = ‘O’;
    2. **No**: player1Symbol = ‘O’, player2Symbol = ‘X’;
11. Play the game
12. Is the board full?
    1. **Yes**:call **CheckWinner();** -> **Step 13**
    2. **No**: call **MakeMove(currentPlayer);** -> **Step 14**
13. CheckWinner() return value is..
    1. **0 (No winner)** : Switch currentPlayer to the other player -> dungeon.exploredTime += 4 -> **Step 12**
    2. **1 (Player1 wins):** Set winner = “player1”;
    3. **2 (Player2 wins):** Set winner = “player2”; -> dungeon.exploredTime += 15;
    4. **3 (Tie):** dungeon.exploredTime += 8;
14. Call **ClearBoard()** and Calculate **totalGames++;**
15. Is **totalGames > 5**?
    1. Yes: **Step 16**
    2. No: continue -> **Step 8**
16. End/Next Logic.

## Logic 3 - Pseudocode:

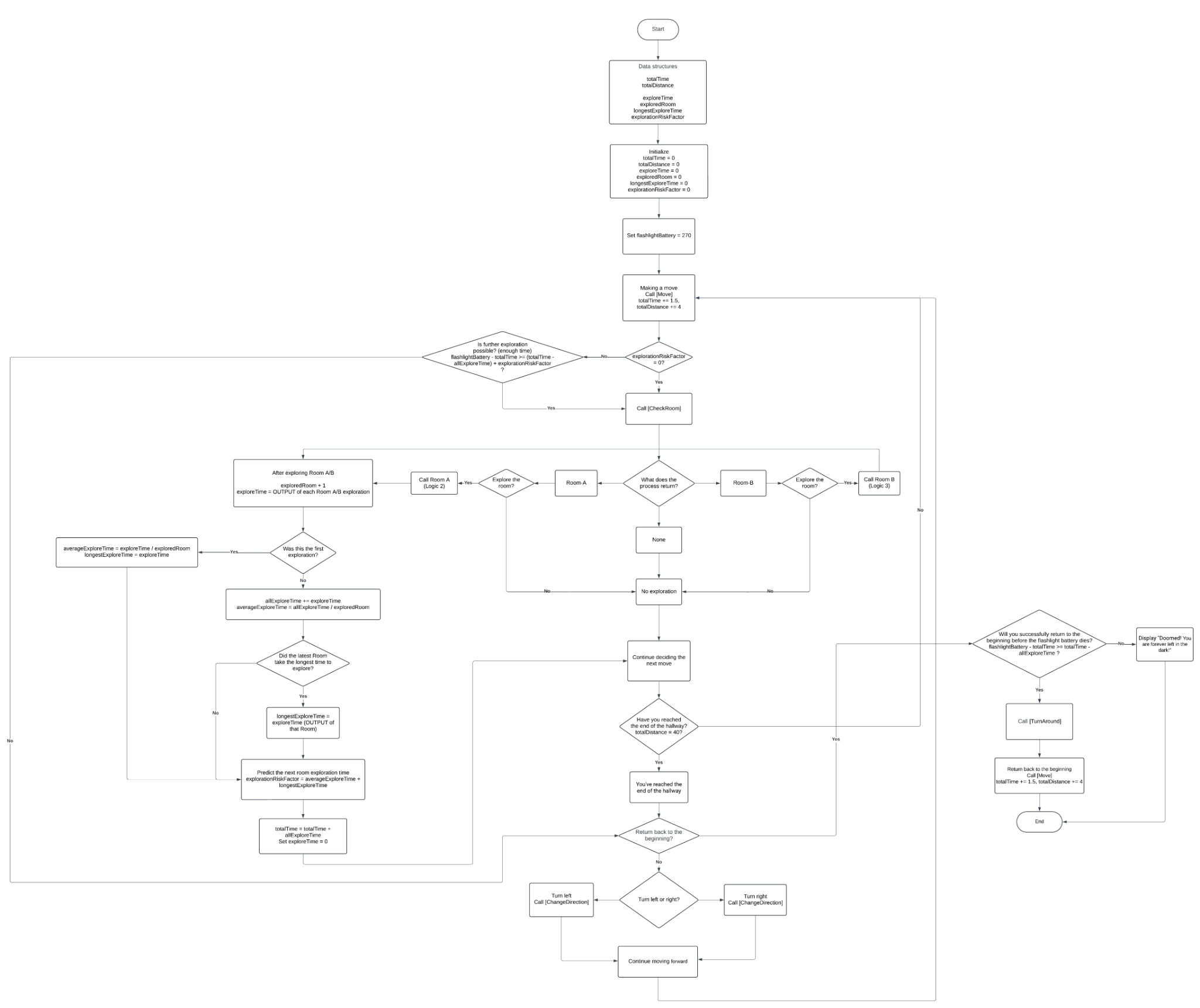
1. Start/Previous Logic
2. Create a variable **“totalTime”** and **initialize it to 0**
3. Create a variable “**totalGuesses**” and **initialize it to 0**
4. Call **GenerateWord**
5. Does the **user** want to **guess a letter**?
   1. **Yes**:Call **GuessLetter** with **user input** and **(totalGuesses + 1)** -> **Step 6**
   2. **No**: continue -> **Step 9**
6. Did **GuessLetter** return **“1”**?
   1. **Yes**: **(totalTime + 3)** -> **Step 8**
   2. **No**: continue -> **Step 7**
7. Did **GuessLetter** return **“-1”**?
   1. **Yes**: **(totalTime + 6)** -> **Step 8**
   2. **No**: **(totalTime + 2)** -> **Step 12**
8. Does **totalGuesses = 8**?
   1. **Yes**: continue -> **Step 10**
   2. **No**: continue -> **Step 9**
9. Does the **user** want to **guess the word**?
   1. **Yes**:continue -> **Step 10**
   2. **No**: continue -> **Step 5**
10. Call **GuessWord** with **user input**
11. Did **GuessWord** return **“1”**?
    1. **Yes**:if **(totalGuesses = 8)** then **(totalTime + 5)** else **(totalTime + 2)** -> **Step 12**
    2. **No**: if **(totalGuesses = 8)** then **(totalTime + 25)** else **(totalTime + 20)** -> **Step 12**
12. Return **totalTime**
13. End/Next Logic.

# FLOWCHARTS:

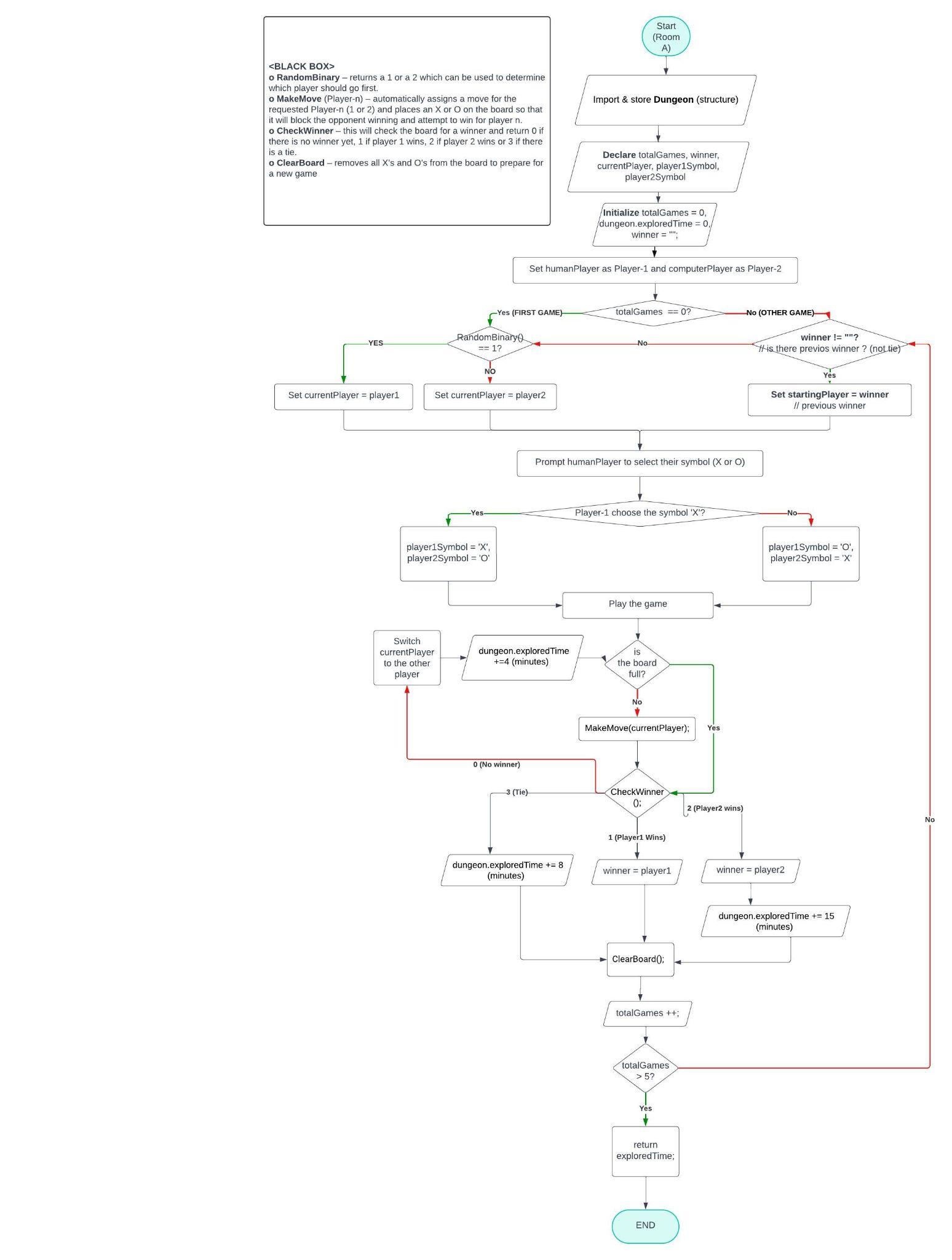
## Main Flowchart:



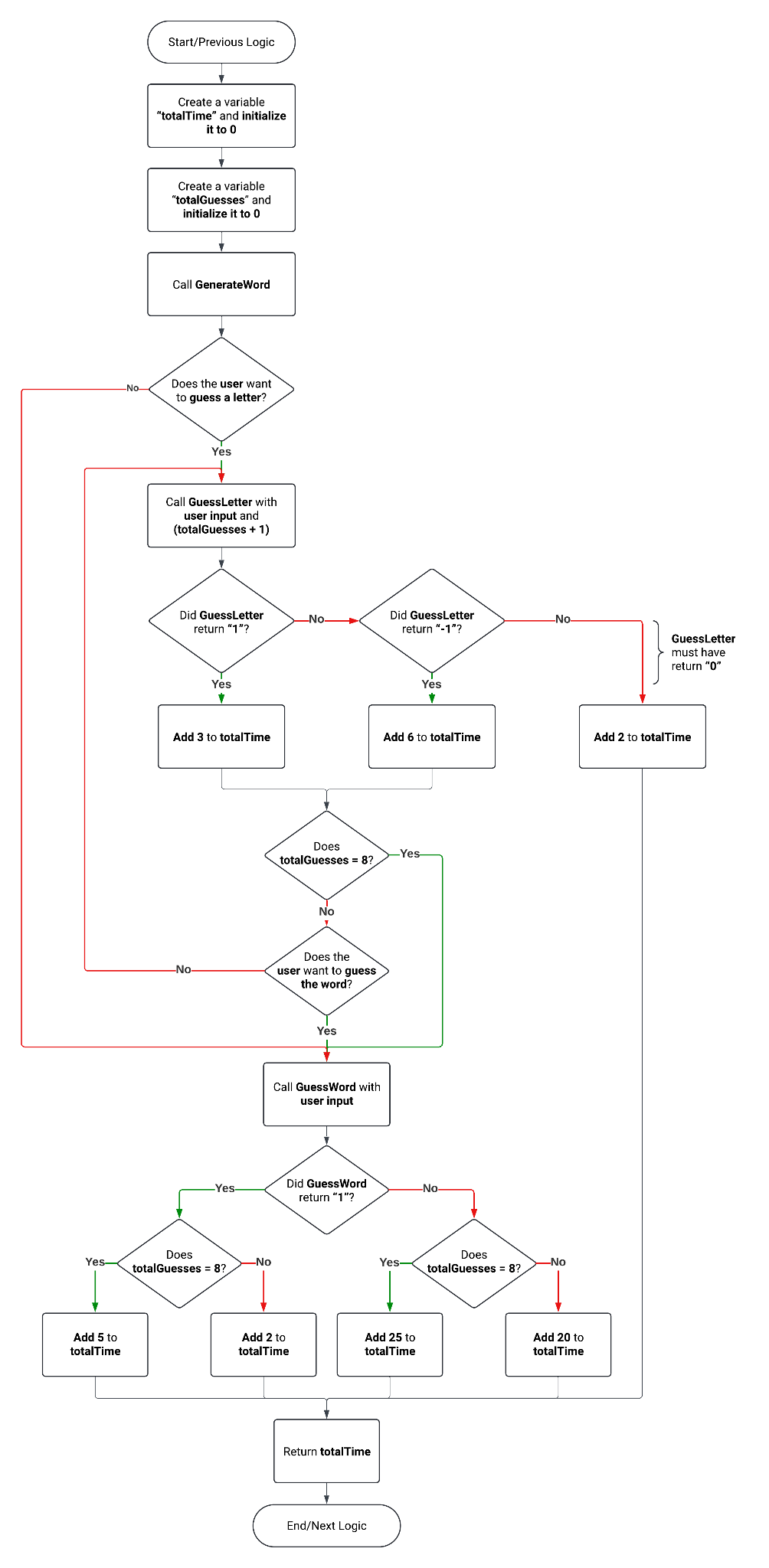
## Logic 1 Flowchart:



## Logic 2 Flowchart:



## Logic 3 Flowchart:



# TEST CASES

1. The player starts at the bottom of the staircase with a full battery.

**Expected Output:** The player should be able to explore a certain number of rooms before returning to the staircase landing.

2. The player explores rooms without keeping track of their movements or battery life.

**Expected Output:** The player gets lost and runs out of battery, leading to failure.

3. The player manages battery life efficiently but gets lost due to not keeping accurate records of their movements.

**Expected Output:** The player runs out of battery before returning to the staircase landing, leading to failure to end the game.

4. The player keeps accurate movement records but fails to manage battery life effectively.

**Expected Output:** The player reaches the maximum distance but runs out of battery before returning to safety, leading to failure.

5. The player encounters obstacles or challenges in certain rooms that require additional time or resources.

**Expected Output:** The player may need to backtrack or find alternative routes to conserve battery and finish successfully.

6. The player reaches a dead-end room or encounters an insurmountable obstacle.

**Expected Output:** The player must backtrack and find an alternative route, potentially consuming more battery life.

7. The player finds shortcuts or efficient pathways through the maze.

**Expected Output:** The player conserves battery life and maximizes exploration by utilizing these shortcuts effectively.

8. The player tries to explore every single room in the building.

**Expected Output:** The player may succeed if they manage battery life and navigation efficiently, returning to safety before running out of power.

9. The player deliberately rushes through the maze without managing the battery or tracking movements.

**Expected Output:** The player quickly runs out of battery and gets lost, leading to failure.

10. The player finds the room with the game of hangman and decides to play.

**Expected Output:** The game of hangman starts.

11. The player finds the word for the hangman game in under 2 minutes.

**Expected Output:** 2 minutes added to the exploration timer.

12. The player fails to find the word or runs out of time in the hangman game.

**Expected Output:** 20 minutes added to the exploration timer.

# FINAL DESCRIPTION

The program offers a systematic approach to navigating a post-apocalyptic dungeon, focusing on resource management and exploration tactics. The method begins by initializing critical variables such as the overall time spent exploring, the distance traveled, and the life of the flashlight battery.

These indicators form the basis for monitoring progress and making wise choices. The exploration process is divided into two distinct rooms: Room A, where the main character plays games like X and O, and Room B, where they participate in Hang-Person games. Exploring each room adds to the total exploration time, making the quest more complex. To navigate the labyrinthine passageways efficiently, the pseudocode includes routines for changing direction and turning around when necessary. These routines maximize exploration efficiency while ensuring flexibility in response to the maze's twists and turns.

The primary loop, which repeats if there is enough battery life, is at the heart of the exploration logic. In this cycle, the main character advances, examines, and investigates rooms, modifies exploration metrics, and chooses when to start the journey back. The return trip is crucial to the protagonist's survival as they must safely retreat to the starting point before the lantern battery runs out. The pseudocode ensures the return trip is only started when necessary by considering variables such as exploration danger and remaining battery life. The program includes instructions on playing games and interpreting letters and words found during the investigation. These elements enhance the adventure's overall immersive quality by adding depth and intrigue.

In conclusion, the program provides a comprehensive strategy for an exciting journey in a post-apocalyptic environment. It creates a compelling story about survival in a dark and unknown area by combining thoughtful decision-making, resourceful utilization, and enjoyable games. The quest becomes a test of fortitude and cunning as you venture inside the abandoned building, encountering various obstacles.