Workshop - 11

Workshop Value: 10 marks (4.4% of your final grade)

Learning Outcomes

Upon successful completion of this workshop, you will have demonstrated the abilities:

- to decipher and identify a problem
- to analyze and decompose a problem
- to identify the required detailed steps to solve a problem
- apply modular design and concurrent workflows in developing a solution
- to communicate the solution to fellow peers and non-technical business persons

Workshop Grading and Promotion Policy

Workshops for this course will be assessed using the following criteria:

- Workshops must be submitted before the class time to be graded
- You must successfully complete 9 workshops (if more than 9 are completed, the best 9 will be used)
- Each student is expected to be a presenter of the workshop solution at least once by the end of the term
- Workshop solutions and presentations will be evaluated using the published workshop rubrics

Workshop Overview

It is the post-apocalyptic era, and hardship and depression run 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 life-line (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

Constraints and Rules

- Your flashlight battery has only 3 hours of life
- All hallways are forty (40) steps long

- Each "[***Move***]" unit is 10 steps in distance and takes 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 make a decision 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 can't make the same direction change twice in a row (example: at the end of hallway 1, you turn left… then at the end of hallway 2, you MUST turn right)
- 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.

Navigation "Action" Sub-processes

[**Move**] Moves forward one unit:
- "Time" is incremented by 5 minutes
- "Distance" is incremented by 10 steps

[**CheckRoom**] This process should be called after each "Move" action (only when exploring). If there is a room at the current location, it is explored and the time taken to explore the room is returned (in minutes). Note: if there was no room, zero is returned

Expected Logic
- Determine which sub-process to call based on the output from of the **black box** sub-process [***RandomSearch***] which will return **A**, **B**, or **X** (do not define the black box process, just know you will get a random character output):

Evaluate the [**RandomSearch**] black box output

A: Return the value generated by the call to sub-process [***ExploreRoom-A***]
- **See details for this process later in this document**

B: Return the value generated by the call to sub-process [***ExploreRoom-B***]
- **See details for this process later in this document**

X: Return zero (no room to explore)

[**ChangeDirection**] Rotates your direction 90 degrees (left or right) in-place without affecting distance or time. The output will always be opposite the input value (example: if [**L**]eft is the input, the output should be [**R**]ight.

[**TurnAround**] Rotates your direction 180 degrees in-place (without affecting distance or time)

Expected Logic

- Call [**ChangeDirection**] twice in sequence providing the same input value (either "L" or "R")
- There is no need to store the output value generated by the [**ChangeDirection**] sub-process call

Other Important Details…

Each room you discover will take an unknown/variable number of minutes (based on what lies within). Track room exploration time separately from travel/distance time. You will need to develop a tracking method to store each room duration and use this information to calculate predictive estimates in the determination of whether you can risk further exploration time or should turn around and return to safety before your flashlight power runs out.

To help you accomplish these estimates, you should apply the following logic:

1. Maintain an ongoing average room exploration time (recalculate after each room exploration)
2. Keep track of the longest time spent in a room (this will represent the worst-case scenario based on actual data)
3. Apply a pessimistic risk factor in **predicting** the next room exploration time by *adding both the average room exploration time with the room that took the longest time*. This estimated time should be used in the assessment of whether further exploration is possible.
4. You must also factor the time required to return to safety (distance traveled).

Note: When returning, no room explorations can be performed so the time is based purely on the traveled distance (result of a "[***Move***]" command).

ExploreRoom-A

Congratulations you found a room that will provide you with some much-needed fun time playing "Tic-Tac-Toe"! Use the small dedicated handheld "Tic-Tac-Toe" game machine and enjoy!

Tic-Tac-Toe is a game comprised of a board of 3 squares by 3 squares. This is a 2-player game of which you will be "Player-1" (Player-2 will be the game machine). The rules are simple, each player uses their own unique identifier ("X" or "O") and places their identifier in one of the empty squares. Turns are alternated until all squares are filled OR when a winner can be declared. The objective is to get 3 matching squares in sequence either horizontally, vertically, or diagonally. The best of 3 games must be played to determine the winner.

Black Box Processes you can use

- There are 4 black box processes you can use to simulate the playing of the game.
    - **RandomBinary** – returns a 1 or a 2 which can be used to determine which player should go first.
    - **MakeMove**(Player n) – performs strategy and makes a move for player n (1 or 2) and places an X or O on the board such that it will block the opponent winning and attempt to win for player n.

- **CheckWinner** – this will check the board for a winner and return 0 if no winner yet, 1 if player 1 wins, 2 if player 2 wins or 3 if there is a tie.
- **ClearBoard** – removes all X's and O's from the board to prepare for a new game

Constraints

- Determine who starts the first game by calling a black box [***RandomBinary***] process that returns a number (1 or 2). Do not define the black box process, just know you will get a random number output:
  - **1**: means Player-1 starts
  - **2**: means Player-2 starts
- Each subsequent game's starting player will be whoever won the previous game or, if a tie, then the player who did not start last time
- Each player move adds 1 minute to the explored time
- Each tied game adds another 2 minutes to the explored time
- Each game you lose (Player-1), 5 minutes is added to the explored time
- The OUTPUT returned by this process is the total time accumulated to play all three games.

ExploreRoom-B

Congratulations you found another room that will provide you with some much-needed fun time playing "Hang-Person" (formerly known as "Hangman")! Use the small dedicated handheld "Hang-Person" game machine and enjoy!

This game requires 1 player. The game begins when a random generated hidden secret word (between 5 and 15 characters) is chosen by the game logic. You are shown a series of underscore characters (underlines) that represent the hidden characters of the secret word. You must guess letters one at a time in an attempt to reveal/show as many matching letters in the secret word as you can. If you can correctly guess the word you win otherwise, you lose!

Black Box Processes you can use

- **RandomWord** – generates a random 5-15 letter word to be guessed.
- **Guess** – guesses a letter in the word using artificial intelligence to find a match. If it guesses a letter correctly, it returns 1 or -1 if the letter is wrong. If it guesses a letter which completes the word, it returns 0.
- **ShouldSolve** – uses AI to compute the probability that you can successfully solve for the correct word. Returns 1 is there is a high probability of solving the word and 0 if there is a low probability.
- **Solve** – attempts to guess the solution to the word using AI. It returns 1 on success and 0 on failure.

Constraints

- You can call ***RandomWord*** to create a word to guess and then call ***Guess*** and/or ***Solve*** until you guess the word or run out of guesses.

- You have a maximum of 10 letter guesses
- Each letter guess that matches a letter in the secret word will add 1 minute of explore time
- Each letter guess that does not match a letter in the secret word will add another 2 minutes to the explore time. If a letter guess solves the word, then 5 minutes is added to the explore time.
- You can attempt to solve (identify) the secret word at any time
- If you incorrectly guess at the secret word, 10 minutes is added to your explore time
- If you successfully identify the secret word, you win! Only 5 minutes is added to the explore time.
- If you run out of letter guesses and can't identify the secret word, you lose! Another 10 minutes is added to the explore time
- The OUTPUT returned by this process is the total time accumulated to play the game.

Your Task

[Logic 1] the logic for Room A which will return the time it took to explore Room A.
[Logic 2] the logic for Room B which will return the explore time for the room.
[Logic 3] the main logic for exploring the dungeon (including returning to the starting point) which will invoke logic 1 or logic 2 when it needs to explore room A or room B.

Task	Subtask	Member(s)	Marks	Comments
Pseudocode	Logic 1	1	40%	
	Logic 2	2	40%	
	Logic 3	3	40%	
	Combined	1-3	60%	
FlowChart	Logic 1	4	40%	
	Logic 2	5	40%	
	Logic 3	6	40%	
	Combined	4-6	60%	
Video	Presentation	3 or 6	100%	Members rotate weekly