COMP20050 – Software Engineering Project II

SPRINT 1 - DOCUMENTATION

Sneha Chhipa - 22320616
Pallavi Thapliyal - 22206412
Neha Abey - 22342831

GROUP 20

Documentation

Original Sprint 1

1.1 A feature that implements the board layout with coordinates marked out.
1.2 A feature implementing the "Welcome" screen to the user.
1.3 A feature to place atoms and successfully commence the game.
1.4 A feature that displays the in-game options. (Show/Check/Quit/New Ray/Place Atom)

Sprint 1 Modification

1.1 A feature that implements the **board layout on UI & stores the straight path of from every entry point.**
1.2 A feature implementing the "Welcome" screen to the user on **command line interface and UI**.
1.3 A feature to store **atom locations and successfully commence the game**
1.4 A feature implementing a ray that **doesn't encounter any atoms**.
1.5 A feature implementing a ray that **makes a direct hit to an atom**

Reason for sprint modification:

We have decided to work on the logic of the game first over the UI, which we plan on implementing gradually in the later sprints. This is because it seemed to be a better decision to work on implementing the logic first than the UI because we still need to learn more about JavaFX. Therefore, we worked on some of our sprint 2 features as well – features 2.1 & 2.2 of the previous project plan. We will be implementing the UI for these in Sprint 2.

Modified Sprint 1 Implementation Details

1.1 A feature that implements the board layout on UI & stores the straight path of from every entry point.

-- Board

The hexagonal board is implemented using JavaFX and SceneBuilder. Currently this board is static only. Its purpose is just to give user an idea of how the game board looks like. The fully interactive board will be implemented in the future sprints along with the UI components to engage the interaction between the user and the game.

Implemented by Neha Abey.

(Two separate boards will not be running in this stage. We are planning on implementing this in Sprint 4, after we can fully implement all the features needed to run one round of the game.)

-- Data structure for possible entry points

This data structure stores the co-ordinates of every hexagon that a ray would meet until it reaches its exit point. 3D Array List is used to store this information, it is a structure consisting of 54 different paths containing the coordinates of each hexagon the path would cross, assuming that the path is clear, and no atom is present. Each path is unique to each unique entry point. The last element in each path is the exit point of that path.

Implemented by Sneha Chhipa.

1.2 A feature implementing the "Welcome" screen to the user on command line interface.

The welcome screen on command line interface indicates to the user that the game is ready to be played. It asks the user for their name and game mode. The interaction between textual interface and the user was implemented using the inbuilt Java Scanner class and its methods.

Implemented by Sneha Chhipa.

There is also a simple UI welcome screen which allows the user to press 'START GAME' to view the game board. - *Implemented by Neha Abey.*

1.3 A feature that stores the atom coordinates and its circle of influence.

A 3D Array List of doubles is also used to store the coordinates of atoms placed and its corresponding circle of influence. It is a structure consisting of 6 different lists, where each list represents the data for each atom. Each such list consists of utmost 7 elements, where the first element within each atom list is the coordinate of the atom itself and the remaining elements is the coordinates of the surrounding hexagons representing the circle of influence. The data in this list is populated either from user input or is randomly generated by the computer. Two separate functions are made to achieve this individually.

Implemented by Pallavi Thapliyal

1.4 A feature implementing a ray that doesn't encounter any atoms.

A function called *isClearPath* is created to implement this feature. It returns a Boolean value, which is used as a variable in the *detectPath* function. The *detectPath* function is a recursive function to analyse the path of a given ray. The return value of the function *isClearPath* is used as one of base cases of the recursive function.

Implemented by Neha Abey.

1.5 A feature implementing a ray that makes a direct hit to an atom

A function called *directHit* is created to implement this feature. It returns a Boolean value, which is used as a variable in the *detectPath* function. The *detectPath* function is a recursive function to analyse the path of a given ray. The return value of the function *detectPath* is also used as one of base cases of the recursive function.

Implemented by Neha Abey & Pallavi Thapliyal.

Features that weren't implemented (according to the old project plan):

1.4 A feature that displays the in-game options. (Show/Check/Quit/New Ray/Place Atom)

This also has a lot to do with UI. So, we decided to leave this for later sprints as it doesn't affect the logic. It is related with the view part according to the MVC model.

Revised Project Plan

Sprint	Features
1	1.1 A feature that implements the board layout on UI & stores the straight path of from every entry point. 1.2 A feature implementing the “Welcome” screen to the user on command line interface and UI. 1.3 A feature to store atom locations and successfully commence the game 1.4 A feature implementing a ray that doesn’t encounter any atoms. 1.5 A feature implementing a ray that makes a direct hit to an atom
2	2.1 A feature implementing a ray that approaches one atom and is deflected 60deg. 2.2 A feature to generate the atoms on the board UI. 2.3 A feature for users to place the atoms on the board UI. 2.4 A feature to show the entry/exit points of rays on board.
3	3.1 A feature implementing a ray that approaches two atoms and is deflected 120deg (two 60deg deflections). 3.2 A feature implementing a ray that approaches two/three atoms and is deflected 180deg. 3.3 A feature implementing a ray that immediately hits a circle of influence and gets reflected straight back. 3.4 A feature that considers another complex path.
4	4.1 A feature that displays the full board with all rays, atoms and circles of influences 4.2 A feature that displays the detailed breakdown of score 4.3 A feature that displays the Welcome Screen with options

Testing

Our test class right now is not usable but will be fully useable in Sprint 2.
We used print statements to test our code ourselves, however we have commented these out so that the textual interface doesn’t look overcrowded and so that the player can play the game normally.

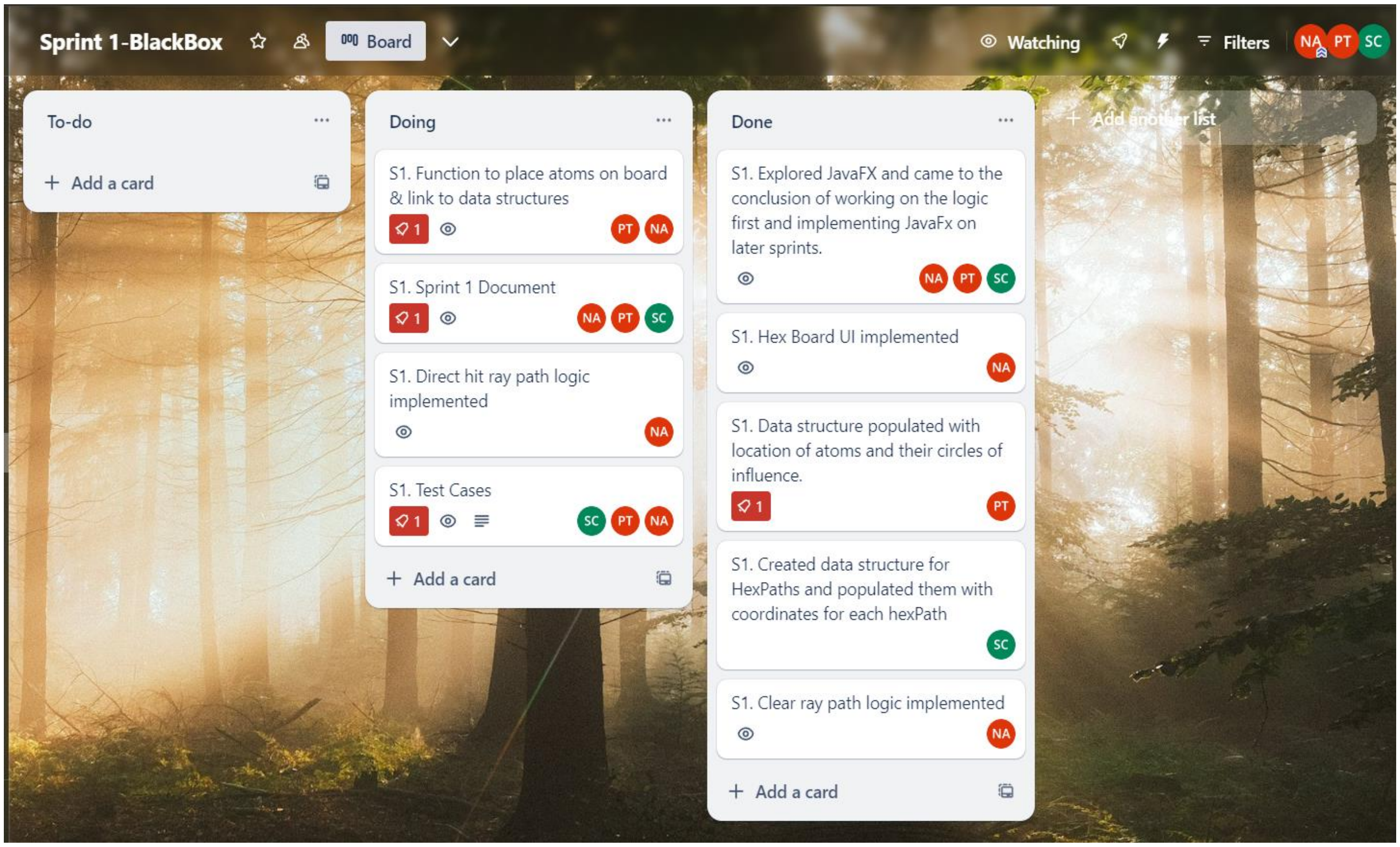
How we tested:

- First, all the coordinates for each entry are being printed; this follows the table on the next page.
- Then user input is taken for game mode – whether atoms are randomly generated, or user generated.
- It then prints the generated atoms and their circles of influence.
- Then user input for ray is taken.
- It prints out the comparison for each ray path - the atoms and circles of influences that may be in their path.
- If it is a clear path, it prints out ‘No Hit. Path is Clear!’.
- If it hits an atom, it prints out ‘Absorbed’.
- If neither condition is satisfied, it prints out ‘Game under development. Remaining analysis of the rays will be available soon.’

Trello

https://trello.com/invite/b/4EFrBUoK/ATTI4691f0792f2fc12eff840d82649aa8119D5D2EB2/sep-blackbox

Below is a screenshot of our Trello Board. We had to move some tasks from 'Done' to 'Doing' to be able to capture all the tasks in one screenshot. However, we have included the link to our Trello board above.



To-do

+ Add a card

Doing

- S1. Function to place atoms on board & link to data structures — 1 — PT NA
- S1. Sprint 1 Document — 1 — NA PT SC
- S1. Direct hit ray path logic implemented — NA
- S1. Test Cases — 1 — SC PT NA

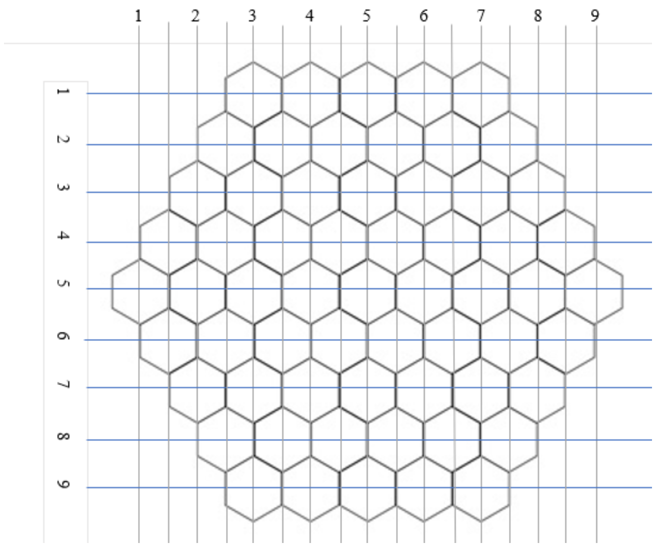
+ Add a card

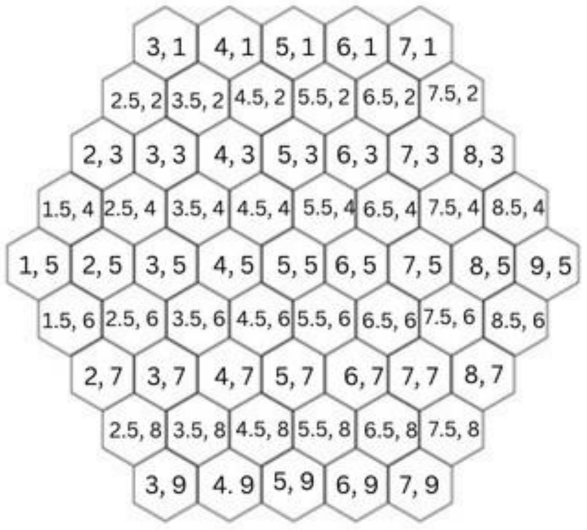
Done

- S1. Explored JavaFX and came to the conclusion of working on the logic first and implementing JavaFx on later sprints. — NA PT SC
- S1. Hex Board UI implemented — NA
- S1. Data structure populated with location of atoms and their circles of influence. — 1 — PT
- S1. Created data structure for HexPaths and populated them with coordinates for each hexPath — SC
- S1. Clear ray path logic implemented — NA

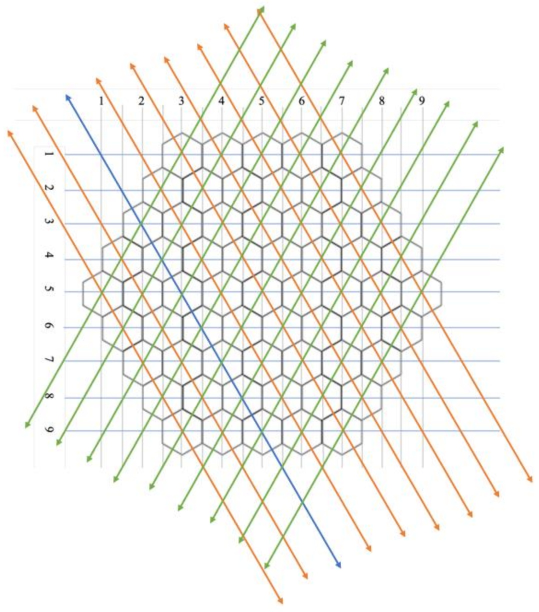
+ Add a card

Coordinate System





Ray Paths (diagonal rays shown only)

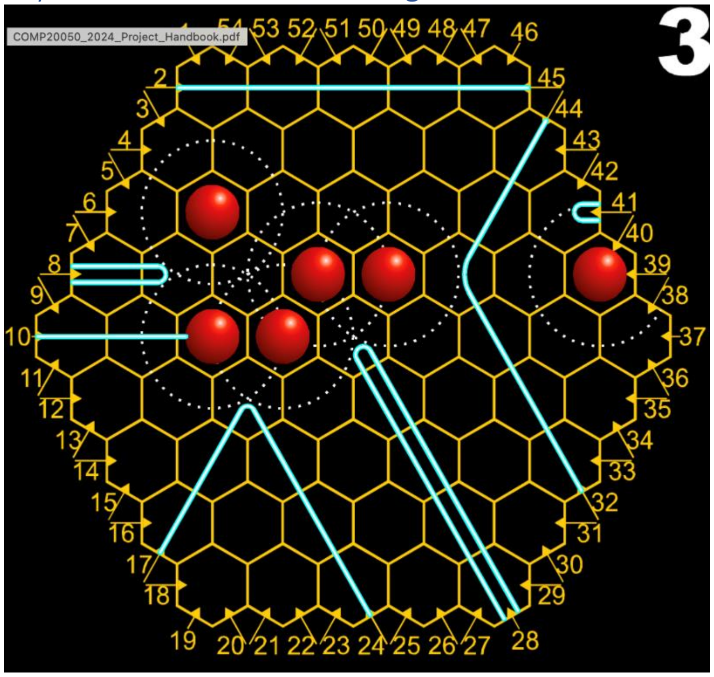


Coordinate grid for each of the entry points/possible ray paths on the board:

1		[3, 1]	[3.5, 2]	[4, 3]	[4.5, 4]	[5, 5]	[5.5, 6]	[6, 7]	[6.5, 8]	[7, 9]
2		[3, 1]	[4, 1]	[5, 1]	[6, 1]	[7, 1]				
3		[2.5, 2]	[3, 3]	[3.5, 4]	[4, 5]	[4.5, 6]	[5, 7]	[5.5, 8]	[6, 9]	
4		[2.5, 2]	[3.5, 2]	[4.5, 2]	[5.5, 2]	[6.5, 2]	[7.5, 2]			
5		[2, 3]	[2.5, 4]	[3, 5]	[3.5, 6]	[4, 7]	[4.5, 8]	[5, 9]		
6		[2, 3]	[3, 3]	[4, 3]	[5, 3]	[6, 3]	[7, 3]	[8, 3]		
7		[1.5, 4]	[2, 5]	[2.5, 6]	[3, 7]	[3.5, 8]	[4, 9]			
8		[1.5, 4]	[2.5, 4]	[3.5, 4]	[4.5, 4]	[5.5, 4]	[6.5, 4]	[7.5, 4]	[8.5, 4]	
9		[1, 5]	[1.5, 6]	[2, 7]	[2.5, 8]	[3, 9]				
10		[1, 5]	[2, 5]	[3, 5]	[4, 5]	[5, 5]	[6, 5]	[7, 5]	[8, 5]	[9, 5]
11		[1, 5]	[1.5, 4]	[2, 3]	[2.5, 2]	[3, 1]				
12		[1.5, 6]	[2.5, 6]	[3.5, 6]	[4.5, 6]	[5.5, 6]	[6.5, 6]	[7.5, 6]	[8.5, 6]	
13		[1.5, 6]	[2, 5]	[2.5, 4]	[3, 3]	[3.5, 2]	[4, 1]			
14		[2, 7]	[3, 7]	[4, 7]	[5, 7]	[6, 7]	[7, 7]	[8, 7]		
15		[2, 7]	[2.5, 6]	[3, 5]	[3.5, 4]	[4, 3]	[4.5, 2]	[5, 1]		
16		[2.5, 8]	[3.5, 8]	[4.5, 8]	[5.5, 8]	[6.5, 8]	[7.5, 8]			
17		[2.5, 8]	[3, 7]	[3.5, 6]	[4, 5]	[4.5, 4]	[5, 3]	[5.5, 2]	[6, 1]	
18		[3, 9]	[4, 9]	[5, 9]	[6, 9]	[7, 9]				
19		[3, 9]	[3.5, 8]	[4, 7]	[4.5, 6]	[5, 5]	[5.5, 4]	[6, 3]	6.5, 2]	[7, 1]
20		[3, 9]	[2.5, 8]	[2, 7]	[1.5, 6]	[1, 5]				
21		[4, 9]	[4.5, 8]	[5, 7]	[5.5, 6]	[6, 5]	[6.5, 4]	[7, 3]	[7.5, 2]	
22		[4, 9]	[3.5, 8]	[3, 7]	[2.5, 6]	[2, 5]	[1.5, 4]			
23		[5, 9]	[5.5, 8]	[6, 7]	[6.5, 6]	[7, 5]	[7.5, 4]	[8, 3]		
24		[5, 9]	[4.5, 8]	[4, 7]	[3.5, 6]	[3, 5]	[2.5, 4]	[2, 3]		
25		[6, 9]	[6.5, 8]	[7, 7]	[7.5, 6]	[8, 5]	[8.5, 4]			
26		[6, 9]	[5.5, 8]	[5, 7]	[4.5, 6]	[4, 5]	[3.5, 4]	[3, 3]	[2.5, 2]	
27		[7, 9]	[7.5, 8]	[8, 7]	[8.5, 6]	[9, 5]				
28		[7, 9]	[6.5, 8]	[6, 7]	[5.5, 6]	[5, 5]	[4.5, 4]	[4, 3]	[3.5, 2]	[3, 1]
29		[7, 9]	[6, 9]	[5, 9]	[4, 9]	[3, 9]				
30		[7.5, 8]	[7, 7]	[6.5, 6]	[6, 5]	[5.5, 4]	[5, 3]	[4.5, 2]	[4, 1]	

31		[7.5, 8]	[6.5, 8]	[5.5, 8]	[4.5, 8]	[3.5, 8]	[2.5, 8]			
32		[8, 7]	[7.5, 6]	[7, 5]	[6.5, 4]	[6, 3]	[5.5, 2]	[5, 1]		
33		[8, 7]	[7, 7]	[6, 7]	[5, 7]	[4, 7]	[3, 7]	[2, 7]		
34		[8.5, 6]	[8, 5]	[7.5, 4]	[7, 3]	[6.5, 2]	[6, 1]			
35		[8.5, 6]	[7.5, 6]	[6.5, 6]	[5.5, 6]	[4.5, 6]	[3.5, 6]	[2.5, 6]	[1.5, 6]	
36		[9, 5]	[8.5, 4]	[8, 3]	[7.5, 2]	[7, 1]				
37		[9, 5]	[8, 5]	[7, 5]	[6, 5]	[5, 5]	[4, 5]	[3, 5]	[2, 5]	[1, 5]
38		[9, 5]	[8.5, 6]	[8, 7]	[7.5, 8]	[7, 9]				
39		[8.5, 4]	[7.5, 4]	[6.5, 4]	[5.5, 4]	[4.5, 4]	[3.5, 4]	[2.5, 4]	[1.5, 4]	
40		[8.5, 4]	[8, 5]	[7.5, 6]	[7, 7]	[6.5, 8]	[6, 9]			
41		[8, 3]	[7, 3]	[6, 3]	[5, 3]	[4, 3]	[3, 3]	[2, 3]		
42		[8, 3]	[7.5, 4]	[7, 5]	[6.5, 6]	[6, 7]	[5.5, 8]	[5, 9]		
43		[7.5, 2]	[6.5, 2]	[5.5, 2]	[4.5, 2]	[3.5, 2]	[2.5, 2]			
44		[7.5, 2]	[7, 3]	[6.5, 4]	[6, 5]	[5.5, 6]	[5, 7]	[4.5, 8]	[4, 9]	
45		[7, 1]	[6, 1]	[5, 1]	[4, 1]	[3, 1]				
46		[7, 1]	[6.5, 2]	[6, 3]	[5.5, 4]	[5, 5]	[4.5, 6]	[4, 7]	[3.5, 8]	[3, 9]
47		[7, 1]	[7.5, 2]	[8, 3]	[8.5, 4]	[9, 5]				
48		[6, 1]	[5.5, 2]	[5, 3]	[4.5, 4]	[4, 5]	[3.5, 6]	[3, 7]	[2.5, 8]	
49		[6, 1]	[6.5, 2]	[7, 3]	[7.5, 4]	[8, 5]	[8.5, 6]			
50		[5, 1]	[4.5, 2]	[4, 3]	[3.5, 4]	[3, 5]	[2.5, 6]	[2, 7]		
51		[5, 1]	[5.5, 2]	[6, 3]	[6.5, 4]	[7, 5]	[7.5, 6]	[8, 7]		
52		[4, 1]	[3.5, 2]	[3, 3]	[2.5, 4]	[2, 5]	[1.5, 6]			
53		[4, 1]	[4.5, 2]	[5, 3]	[5.5, 4]	[6, 5]	[6.5, 6]	[7, 7]	[7.5, 8]	
54		[3, 1]	[2.5, 2]	[2, 3]	[1.5, 4]	[1, 5]				

Layout of our atoms in Testing class.


COMP20050_2024_Project_Handbook.pdf
3
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54