COMP20050 – Software Engineering Project II

SPRINT 2 - DOCUMENTATION

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

GROUP 20

Documentation

Sprint 2 Implementation Details

2.1 A feature implementing a ray that approaches one atom and is deflected 60deg.

-- Ray approaches atom and is deflected 60deg.

Two functions were created, one boolean function to detect if the ray path is a 60 degree deflection (isSixtyDeflection) and one to trace the new path of the ray (sixtyDeflection) and return the path. Another helper function was created to help detect the new ray path (getNeighbours). The 'sixtyDeflection' function tracks the coordinate where the deflection occurs. Using the helper function we get the hexagon (validNeighbor) the ray will traverse after deflection. Finding the path that contains the deflectionCoordinate and the validNeighbor in that order, the path after deflection is found. After combining the deflection path with the original path, the combined path is returned.
Right now, the way the whole Ray class functions is that it has the base cases (directHit and isClearPath) but the recursive cases (60, 120 and 180) of which only 60 is implemented, are not yet recursive. It calculates only one deflection. The recursive functionality and other recursive cases will be implemented next sprint.

Implemented by Pallavi Thapliyal

2.2 A feature to generate the atoms on the board UI.

-- Updated co-ordinate system

In Sprint 1, a basic draft of coordinate system was used as a guide to implement initial game features on command line interface. In Sprint 2, many UI components are required to be implemented. Therefore, the co-ordinate system is updated which corresponds to the x and y location of any component on the graphical user interface. The transition from the old coordinates of hexagons to the new coordinates is the following: the difference in x values of two consecutive hexagon in a row is +70 and the difference in y values of two consecutive hexagon vertically is +63. All primary data structures were updated with this change!

Implemented by Sneha Chhipa

-- Generating atoms

To generate and display atoms on the UI hexagonal game board, it was essential to give each hex cell of the board a unique id. This was done using SceneBuilder. Each hex cell is given a unique fx:id for example 'hex1'. This unique allowed to detect and handle 'on mouse clicked' events. The functions from the JavaFx Node.class were used to implement this feature. Functions like getLayoutX() and getLayoutY() was used to get the centre coordinates of the hex cells and functions like setLayoutX() and setLayoutY() was used to map the centre coordinates of the atoms to the selected hex cell. Features like the opacity of a shape was also used to implement the show/hide functionality of the atoms. The circle of influence for each atom is implemented similarly to the atoms.

Implemented by Sneha Chhipa

-- Randomly generated atoms

The randomly generated atoms function was implemented in Sprint 1 and modified in Sprint 2 according to the new coordinate system specified above. The display of computer generated atoms would be displayed in the game UI as mentioned above.

Implemented by Pallavi Thapliyal

2.3 A feature for users to place the atoms on the board UI.

-- Place atoms on the game board - Setter

During Sprint 1, our initial implementation of generating atoms for the board game was through command line interface. In Sprint 2, this feature is further developed by incorporating user interaction with the game UI. The player can now just click the hex cell on the game board to indicate where they want to place an atom. When the user has placed all 6 atoms, they click ‘finish’ to submit. If they change their mind and want to place atoms at a different location, they click ‘reset’ which allows the user to re-enter their choice of atom locations.

-- Place atoms on the game board - Experimenter

Similarly to the setter, the experimenter can place atoms on the hex board to submit their guesses and indicate end of round. When the experimenter clicks ‘Check’ after finishing, the coordinates of their guessed atoms are stored in a separate array list and a check function is run which is implemented to analyse how many of the experimenter’s guesses were successful and how many were incorrect. The result of the analysis is printed on the command line interface.

Implemented by Sneha Chhipa

2.4 A feature to show the entry/exit points of rays.

-- UI arrow buttons for user to shoot ray.

Buttons were placed around the board for each spot a ray can enter the board. CSS was used to create the arrow shape of the button so that it would be clear for the experimenter to see where they would be shooting the rays. Each button has a unique fx:id, for example ‘b1’, ‘b2’ etc. This allows it to detect ‘onAction’ events. Once the button is clicked, it calls the ‘shootRay’ for each unique entry point. The ‘shootRay’ function (also implemented in the Controller class) creates a new Ray object with the given entry point and adds it to the ‘userRays’ array list.

Implemented by Neha Abey

-- Displaying exit points of ray.

Once the button is clicked, the entry point of the ray path and the path it has detected is printed on command line. For example, it prints ‘27 - CLEAR’, ‘1 - DIRECT HIT’ and ’45 - SIXTY DEGREE DEFLECTION’. It also prints the exit point for each ray – if there is one. For any paths we have not implemented yet, it prints the message, “Game under development. Remaining analysis of the rays will be available soon.”

Implemented by Pallavi Thapliyal

Additional game utilities

-- Modification in ‘Welcome Screen’

Allows the user to choose between playing against computer (where atoms are randomly generated) or against a player (atoms are placed by a player). If computer is chosen, the screen switches to ‘experimenter-view’. If player is chosen, the screen switches to ‘setter-view’.

-- Setter View

Text in the corner which specifies the view – ‘Setter View’.
The setter view displays the hexagonal board as well as the atoms that can be placed.

-- Experimenter View

Text in the corner which specifies the view – ‘Experimenter View’.
The experimenter view displays the hexagonal board as well as the atoms that can be placed and the buttons for each ray that can be shot.

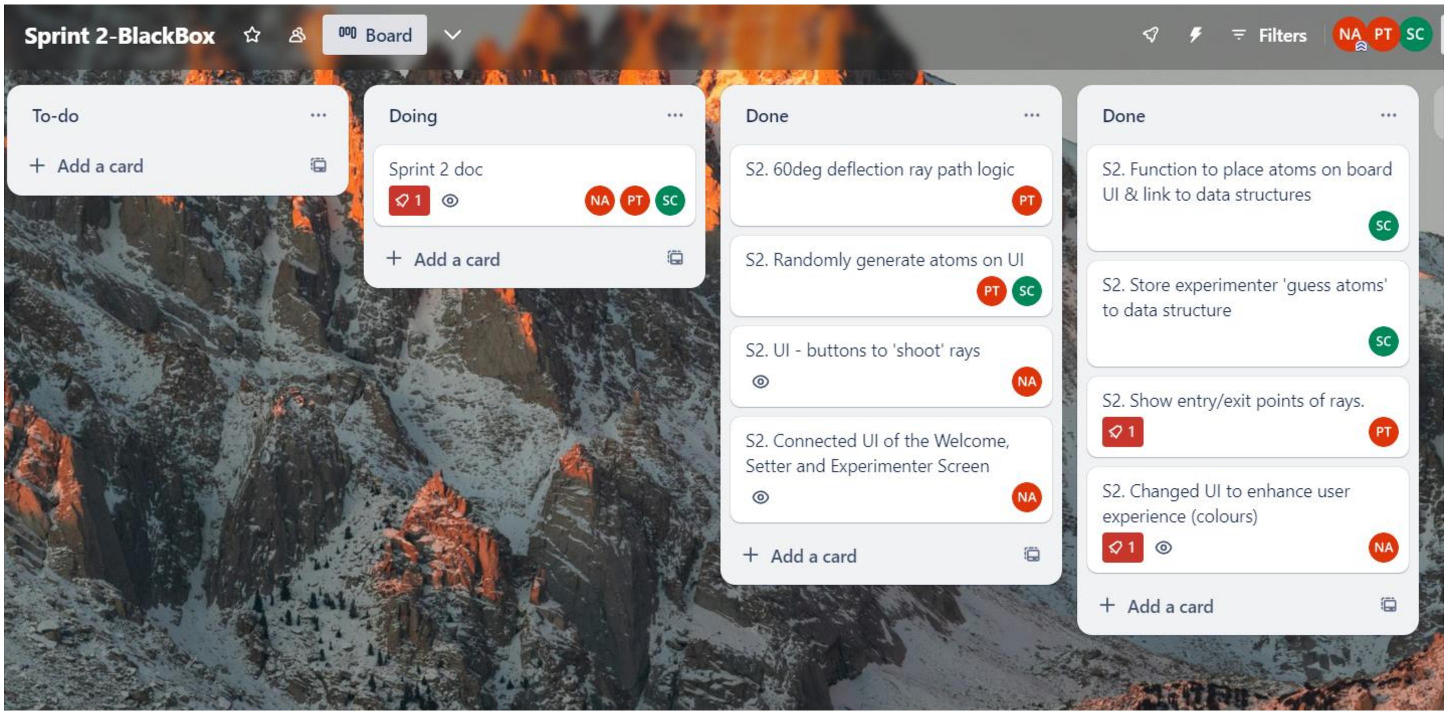
Implemented by Neha Abey

Testing

According to the project plan, the completed features include features relating to the ray logic, features relating to storing game data, and features that are related to implementing the graphical user interface components. The ray logic and correct storage of game data is tested in the file TestGame.java. The testing of the functions is implemented using J-Unit 4. Some of the data structures and logic were also tested using print statements on the command line.

Trello

Link to Trello Board:
https://trello.com/invite/b/Fay1maN9/ATTIbb29d8f75f8b193a1ea60057a1faf85cB3867940/sprint-2-blackboxBelow
Below is a screenshot of our Trello Board.



Sprint 2-BlackBox

Board

Filters NA PT SC

To-do
+ Add a card

Doing
Sprint 2 doc — 1 — NA PT SC
+ Add a card

Done
S2. 60deg deflection ray path logic — PT
S2. Randomly generate atoms on UI — PT SC
S2. UI - buttons to 'shoot' rays — NA
S2. Connected UI of the Welcome, Setter and Experimenter Screen — NA
+ Add a card

Done
S2. Function to place atoms on board UI & link to data structures — SC
S2. Store experimenter 'guess atoms' to data structure — SC
S2. Show entry/exit points of rays. — 1 — PT
S2. Changed UI to enhance user experience (colours) — 1 — NA
+ Add a card

Additional Resources

Coordinate System

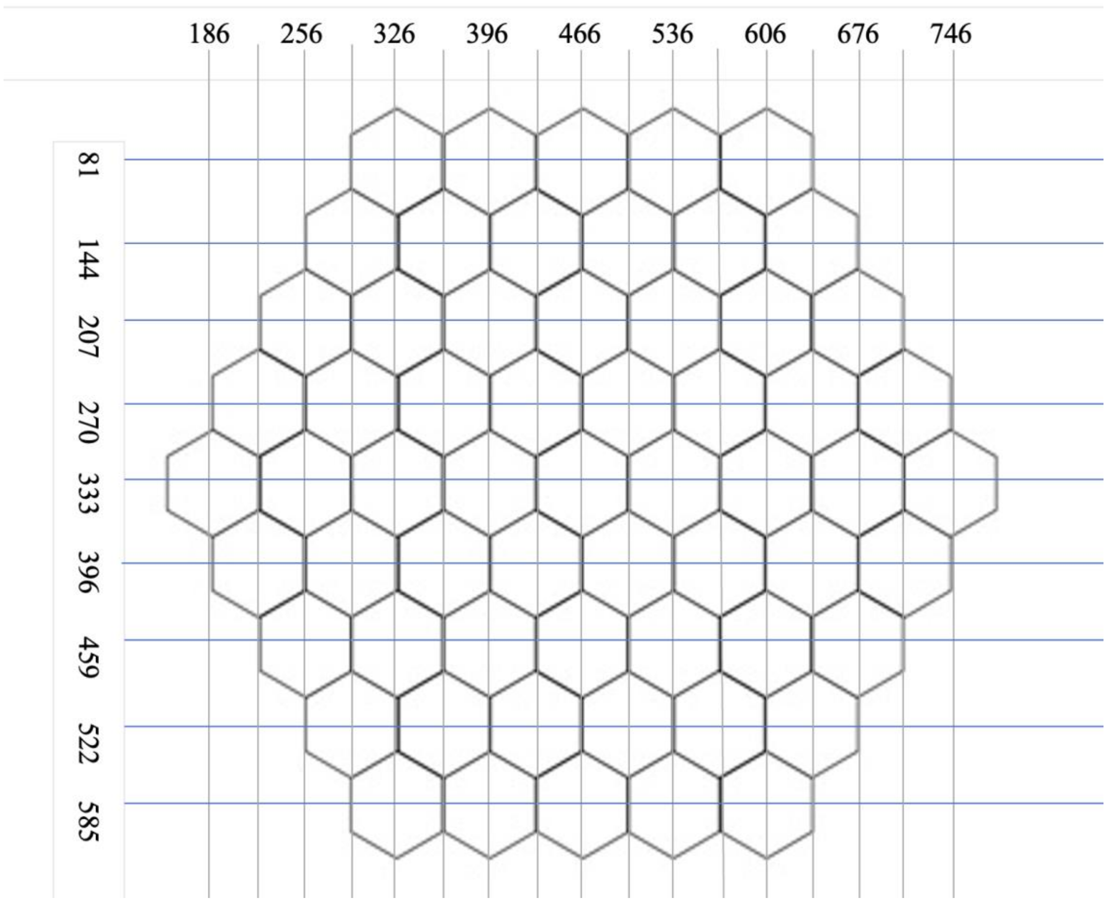


Figure 1 Updated Co-ordinate System