**Project Explanation**

**Students:**

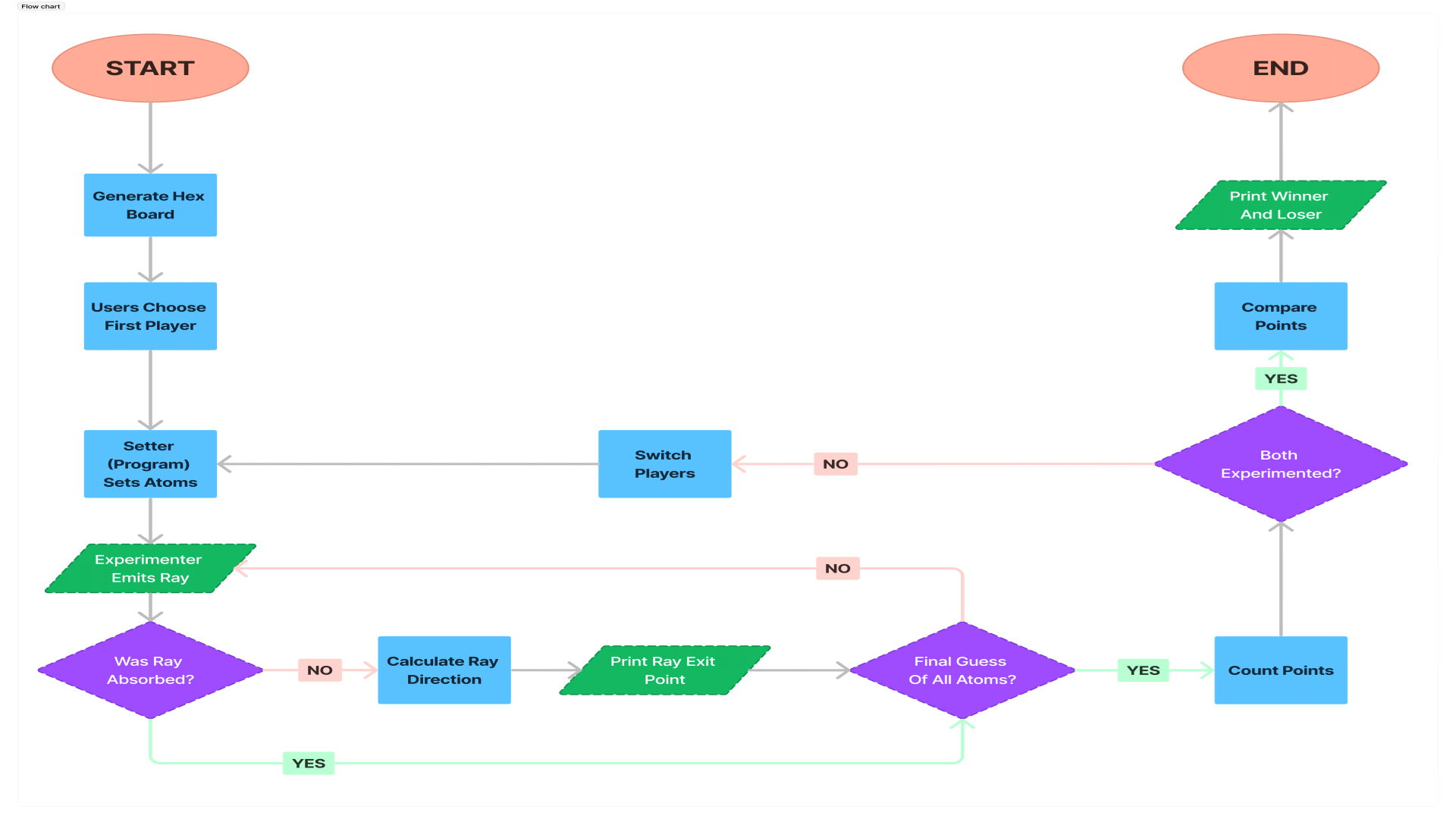
Liam Kerrin - 22371791

Narcis Petrica-Balint - NUMBER

Abdullah Shinwari - 22496944

**Software Architecture and Logic**

In this brief section we will cover what we did in planning the logic or the game. Here was obviously a need for a clear gameplay loop, and a series of steps involved in making this occur, so we constructed a flowchart in a program called FigJam (<https://www.figma.com/figjam/>). Here it is below, then an explanation will follow:



A high-definition image should also be available on GitHub repository, in:

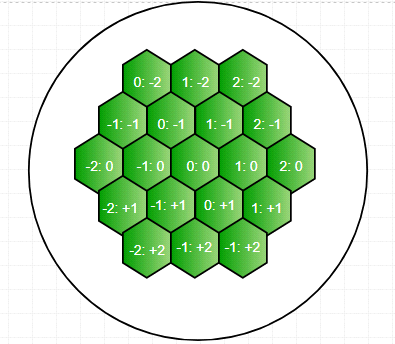
Documentation/img/flowchart\_1.png

The circular shapes are start/end, the rectangles are logical calculations for the program to do, the parallelograms are input/output, and the diamonds are branches.

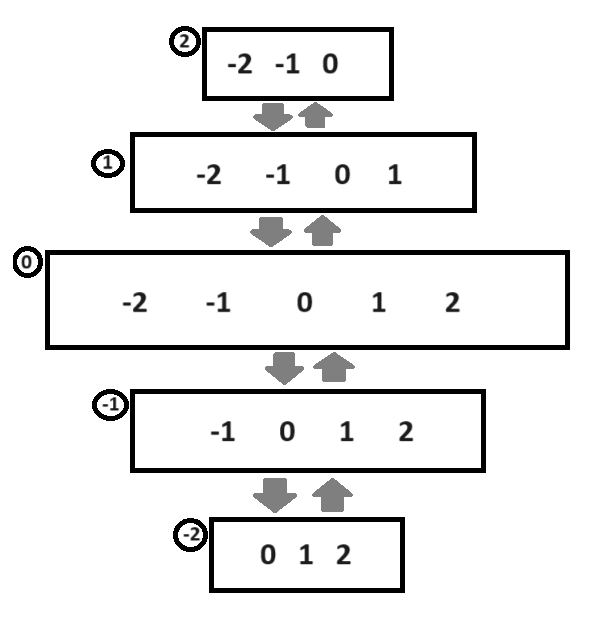
Essentially, the game would begin and generate a visual for the hexagonal board, then the player would choose who experiments first. Once this has been chosen, there is an opportunity for the setter (computer) to place the atoms. Once the atoms have been placed, it is the first experimenter’s turn to begin playing. This then begins the main gameplay loop, where the player can continue placing rays until they are ready to make an educated guess for the placement of all the atoms. Once the guess has been made, points are tallied for the player, then the second player experiments, and the gameplay loop begins again.

It is important to note that the points are tallied before the switch, because the state of the game will be reset when the consecutive players starts playing, so we use the current data to add points to the first player before the board is reset. Finally, once both players have experimented, the program compares points, and prints the winner (the one with the least points overall) and the loser, then the program ends.

Now I will essentially explain the co-ordinate system we plan to implement using a diagram from GeeksForGeeks that helped us come up with this system. (<https://github.com/narcisbalint2002/software_engineering_2024/commit/cfa9b7d4331a4678b7fa5cbf9b3a26ddd9b63139>)



Before we begin, imagine turning the hexagon anti-clockwise by around 60 degrees such that the 0: diagonal in the middle is now fully horizontal. Basically, that is one of the rows. Then each row has a different number of columns. The way we plan to represent this is using nodes for each respective row, -2,-1,0,1,2, then each node will have an array. Since the shape of the board doesn't change, each row will have that set number of columns. For example, row 0 has 5 columns, -2,-1,0,1,2, but row 2 has 3 columns, -2,-1,0. Because of the shape of the hexagon, there's no way these would have a different number of columns, so the number of columns for that row will be fixed. This is, of course, obvious to us visually as we can follow the diagram and see which hexagons connect and which don't. But the computer does not have this visual and needs a programmatic way of representing it, like so:



Imagine this as the hexagon but turned anti-clockwise so the 0 diagonal is now horizontal. This is similar to how it would be represented in our project. The nodes each link to one another as a doubly-linked list (note there is no pointer to null in the diagram, this is just for the sake of simplicity, there would be a pointer to null on either end in a fully implemented doubly-linked list).

This then helps us represent circles of influence. We will explain the case for one atom, and that should clarify the general concept.

If we place an atom in say, 0:-2, we can see that the circle of influence occupies tiles 1:-2, 0:-1 and -1:-1. The computer cannot see this, but there is a pattern. For any co-ordinate, tiles that are part of the circle of influence will be +/-1: +/-1, this means they could be in an adjacent row, or in an adjacent column, or both. We just need to go through the current node and the two adjacent nodes in our linked list (in the example I set with atom at 0:-2, we check current node 0 for both 0:-3 and 0:-1) and check for the respective columns in the array for each node. Now, you may be thinking, 0:-3 doesn't exist. That is true. But we have already defined all possible values in the array for 0, which are -2,-1,0,1,2, and -3 is not in that array, so nothing happens to that, it's not even in the board, so no circle of influence.

To further expand on that, we can take an example where it is NOT in the circle of influencebut it DOES exist in the board. For example, 0:0. Visually we can see this is not in the circle of influence for the atom at 0:-2. The computer can also come to this conclusion because, while it is in the same node (i.e. same "row"), it is +2 far away from 0:-2, so therefore it's not in the circle of influence.