COMP20050 - Group 47 - Features
Sprint 1

Student Numbers: 22441636, 22450456, 22715709

In this sprint our main goal was to firstly, have the ability to hold the board's state and secondly, allow the idea of a “setter” to be created and to then interact with the board.

Implemented Features:

Atom:

Our atoms are represented through an Atom class which holds all necessary information about each atom. Once an atom is created, its circle of influence is generated around it in the correct position according to our board. Each atom object holds an array list of its circle of influences which are divided into individual parts.

```
//adding circle of influence to atom array list of circle of influences
//creating new circle of influence object
circleOfInfluence.add(new CircleOfInfluence(xCo_ord,  yCo_ord: yCo_ord - 1,  orientation: 45));
circleOfInfluence.add(new CircleOfInfluence(xCo_ord,  yCo_ord: yCo_ord + 1,  orientation: 45));
circleOfInfluence.add(new CircleOfInfluence( xCo_ord: xCo_ord + 1, yCo_ord,  orientation: 90));
circleOfInfluence.add(new CircleOfInfluence( xCo_ord: xCo_ord - 1, yCo_ord,  orientation: 90));
circleOfInfluence.add(new CircleOfInfluence( xCo_ord: xCo_ord - 1,  yCo_ord: yCo_ord + 1,  orientation: 135));
circleOfInfluence.add(new CircleOfInfluence( xCo_ord: xCo_ord + 1,  yCo_ord: yCo_ord - 1,  orientation: 135));
```

Circle of Influence:

Each circle of influence has an x coordinate, y coordinate and also an orientation on the board. The orientation will become useful as we advance the project and incorporate rays into it and also useful in giving a graphic representation of an atom

Intersecting Circle of Influence:

In the event that two or more circles of influences are in the same hexagon, we decided to create another class called “IntersectingCircleOfInfluence” which is vital in showing the state of the board but also when rays will interact with it. For now it’s implementation includes an array list of circle of influences which will now allow us to access each part of a circle of influence contained in a single hexagon

Board:

We implemented the idea of the board using a 2D object array data structure. We came together as a team and agreed that this would allow us to accurately represent the precise state of the board.

```
//object array which is the board of the game
private final Object[][] board = new Object[HEIGHT][WIDTH];
```

This data structure is included in our “Board” class which will be the centre point for all game logic and the “model” in our MVC architecture. When a new Board object is created in our “Game” class, in the constructor, the board is initialised, placing “NullHex” objects in out of bound locations and also “EmptyRayMarker” objects in the perimeter around the main game board. See below the representation of the board in a 2D visualisation.

Black denotes valid atom input
Red denotes invalid position on board
Blue denotes position which will include a ray marker

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0	9,0	10,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2	10,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4	10,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6	9,6	10,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7	9,7	10,7
0,8	1,8	2,8	3,8	4,8	5,8	6,8	7,8	8,8	9,8	10,8
0,9	1,9	2,9	3,9	4,9	5,9	6,9	7,9	8,9	9,9	10,9
0,10	1,10	2,10	3,10	4,10	5,10	6,10	7,10	8,10	9,10	10,10

Our board class includes methods to interact directly with the board placing an atom in a given position. This method then executes another important method called `”placeCircleOfInfluence”` which takes in the newly created atom object as argument and then cycles through that atoms circle of influence array list and places each onto the board in the correct position.

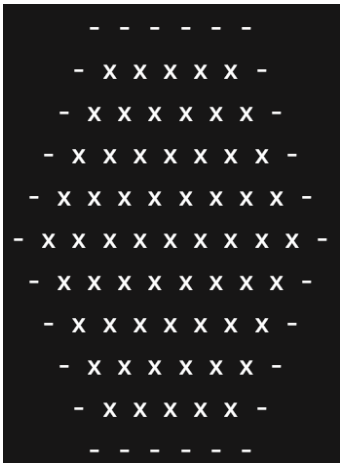
As we have accounted for positions where ray markers will be present on the board, it simplifies the logic in placing a circle of influence as we don’t have to check for edge cases around the board as when the `”placeCircleOfInfluence”` method is placing a circle of influence, if the position it will place in contains an empty ray marker, it will not place it. This similarly works for atoms and allows us to not override atoms.

GameView:

In sticking with our MVC architecture, we have a separate class which just handles showing the user the current state of the board. This class takes in a “Board” object which will be the one used in our main “Game” class and has methods to just print to the terminal, a visual representation of the board. This class will however become more developed as we progress through the project and will eventually become a GUI opposed to a text based interface. For now, it cycles through the board[][] and depending on what object is present at a given position will print a different graphic. We also incorporated some ANSI colour to make differentiating items in the board easier. The method to print the board also correctly spaces each item allowing for a hexagonal graphic.

Below are examples of what the board looks like

1) Blank Board



2) Atom placement in standard and edge case

3) Circle of Influences intersecting

In the above graphics “/”, “\” and “|” all represent parts of a circle of influence, “x” represents an intersection point and “o” represents an atom.
The perimeter of the board hold “-” which depicts a place in which a ray marker can be placed

PlayerInput:

This class is in line with our idea of MVC architecture and plays a part in the “controller” side of things along with the “Game” class. For now this class serves the purpose of taking in user input through the command line and then validating it. It then returns the correct inputs to the “Game” class which then uses it to execute other methods. As of now, we have included a `”inUserName”` method which returns

a valid player name and also a `"getAtomInput"` method to return an 2 integer array holding at [0], an x coordinate and at [1] a y coordinate.

Game:

The main “Game” class is where all of these individual parts come together to begin a game. This class includes the `"playGame"` method to start a new game. It will execute commands to receive user input and then will feed this input to the board (model) and then once the model has taken care of logic will execute the print board function (view) to update it for the user.

With use of the “Game” class, we have fulfilled all objectives of our sprint as we have the ability to create a board and allow a user to interact with it, that user being the “setter”.

Testing:

In testing our software, we mainly focused on the board[][] as it holds all information such as atoms, circles of influence and intersecting circles of influence. The atom was the first object we wanted to ensure was placed onto the board correctly. This resulted in creating various atoms and just placing them onto the board ensuring that the coordinates contained an atom.
(Below is an example of one of the tests)

```
Board b = new Board();
b.placeAtom( x: 5,  y: 1);
assertTrue(b.getBoardPosition( x: 5,  y: 1) instanceof Atom);
```

All of which passed leaving us satisfied that the atom placement is working and orderly.
We also ensured that an error is thrown when an atom is placed into an incorrect place (Below is an example of one of the tests)

```
assertThrows(IllegalArgumentException.class, () -> {
    b.placeAtom( x: 1,  y: 1);
});
```

Next we ensured that placing an atom ensured that a circle of influence was placed around it. We created tests to place an atom and then assert that a circle of influence surrounded it.
(Below is an example of one of the tests)

```
b.placeAtom( x: 4, y: 5);
assertTrue(b.getBoardPosition( x: 4, y: 6) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 3, y: 6) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 5, y: 5) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 3, y: 5) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 4, y: 4) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 5, y: 4) instanceof CircleOfInfluence);
```

We also ensured that edge cases were captured through asserting that in certain areas, i.e. edge of the board where ray markers are placed or on top of another atom, a circle of influence was not placed, asserting it false.
(Below is an example of one of the tests)

```
b.placeAtom( x: 5, y: 1);
assertTrue(b.getBoardPosition( x: 6, y: 1) instanceof CircleOfInfluence);
assertFalse(b.getBoardPosition( x: 4, y: 1) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 4, y: 2) instanceof CircleOfInfluence);
assertFalse(b.getBoardPosition( x: 5, y: 0) instanceof CircleOfInfluence);
assertTrue(b.getBoardPosition( x: 5, y: 2) instanceof CircleOfInfluence);
assertFalse(b.getBoardPosition( x: 6, y: 0) instanceof CircleOfInfluence);
```

All tests for circle of influence placement passed, asserting that our program is working correctly.

Another aspect of the circle of influence we tested was its orientation variable which not only affects the display but also will affect ray paths when we implement them.
(Below is an example of one of the tests)

```
b.placeAtom( x: 5,  y: 5);

CircleOfInfluence c = (CircleOfInfluence) b.getBoardPosition( x: 4,  y: 6);
assertEquals( expected: 300, c.getOrientation());

CircleOfInfluence c2 = (CircleOfInfluence) b.getBoardPosition( x: 5,  y: 6);
assertEquals( expected: 240, c2.getOrientation());

CircleOfInfluence c3 = (CircleOfInfluence) b.getBoardPosition( x: 5,  y: 4);
assertEquals( expected: 60, c3.getOrientation());

CircleOfInfluence c4 = (CircleOfInfluence) b.getBoardPosition( x: 6,  y: 4);
assertEquals( expected: 120, c4.getOrientation());

CircleOfInfluence c5 = (CircleOfInfluence) b.getBoardPosition( x: 4,  y: 5);
assertEquals( expected: 90, c5.getOrientation());

CircleOfInfluence c6 = (CircleOfInfluence) b.getBoardPosition( x: 6,  y: 5);
assertEquals( expected: 270, c6.getOrientation());
```

These tests all passed so we are confident that the circle of influence is firstly being placed in the correct place but also with the correct orientation around the atom.

We also tested for an intersecting circle of influences as it is a key component of the game and we wanted to ensure that it was working correctly. For testing, atoms were placed beside each other and we wanted to make sure that the intersecting points were the correct object
(Below is an example of one of the tests)

```
Board b = new Board();
b.placeAtom( x: 5,  y: 5);
b.placeAtom( x: 6,  y: 4);

assertTrue(b.getBoardPosition( x: 6, y: 5) instanceof IntersectingCircleOfInfluence);
assertTrue(b.getBoardPosition( x: 5, y: 4) instanceof IntersectingCircleOfInfluence);
```

We also ensured that if a new atom was placed in a position in which one of its circles of influence would intersect an already existing intersection that the game handled it correctly. We did this by verifying that the size of the objects array list would increase, in this case from 2 to 3.
(Below is an example of the test)

```
b.placeAtom( x: 5,  y: 3);
assertEquals( expected: 3, c2.getCircleOfInfluences().size());
```

This also worked as expected, concluding that all of our features for this sprint involving board manipulation worked correctly and as expected.

Finally we tested the user input side of the game which just ensures that accurate name and coordinates are inputted. As this is more part of the UX, we decided not to throw an error if invalid coordinates were input but instead ask a user to try again. We tested this to ensure an invalid set of coordinates were never input and we would be continually prompted to input correct details.

This concluded our tests for this sprint which allowed us to see that our program, so far, is performing as expected allowing us to conclude sprint 1 as fully completed.