Implementation and Testing

Introduction

Overall the sprint was successful, all tasks in the project plan were complete and a fully functional game has finally been created. We are very happy with how the game turned out and are excited for our peer review feedback.
All bugs that we know of have been removed with the help of unit tests. UI has been upgraded vastly – adding a working finish screen to the game,
Updated 2 player game mode, handling a switch mechanism allowing the 2 players to swap roles half way through.
Updated sandbox game mode to work flawlessly also having a main menu button. The user interface has become very neat and satisfying to use.
Winner is now successfully being generated for the 2 player game mode, single player displays score as technically the player cant “win”.

Objectives	Tasks	Results
From original sprint 4: -Work on efficiency **New Objectives:** -more testing/ bug fixes -work on UI -Work on winner gen	**From original sprint 4:** -Find a way to make methods more efficient **New Tasks:** -come to a conclusion on final design -implement this design -generate winner and display	**New Tasks:** -complete game

MainMenu:

Two Player:

After changing the game play of two player there was a need to re run the class so that each player was given a go to be both set and guess the atoms.

```
twoPlayerButton.addActionListener(e -> {
    frame.getContentPane().removeAll(); //when its pressed, removes everything on screen
    TwoPlayer twoPlayerGame1 = new TwoPlayer( game_number: 1);
    frame.add(twoPlayerGame1, BorderLayout.CENTER); //adds the hex panel.
    frame.validate(); //validates
    frame.repaint(); //painting
});
```

Here we see the first call to two player with the game number being 1 this tells us that the two player gamemode has only been played once and the switch has not accured
Once the first game is finished the second game is called internally inside the finishing action

```
void finishAction() {
    try {//try catch block for error handling
        //if current plyer is 1 and button pressed move to next player
        if (currentPlayer == 1) {
            currentPlayer = 2;
            scoreBoard.setVisible(true);
            button.setVisible(false);
        } else if (currentPlayer == 2 && comparing) {//if curr player 2 and comparing and button pressed
            endGame = true;
            if (game_number == 1) {//if first game
                MainMenu.setPlayer_1_score(score);//store score in mainMenu
                MainMenu.restartTwoPlayerGame();//create and play another game with roles "switched"
```

Here we see a set of if statements. FinishAction is called if the a switch,compare or end game is necessary here we see that while game_number is 1 to set thescore and retartTwoPlyer.

```
    public static void restartTwoPlayerGame() {
        frame.getContentPane().removeAll(); // Remove all content
        TwoPlayer twoPlayerGame2 = new TwoPlayer( game_number: 2);
        frame.add(twoPlayerGame2, BorderLayout.CENTER); // Adds the new game panel
        frame.validate(); // Validates the frame after changes
        frame.repaint(); // Repaints the frame to display the new content
    }
```

The same inislisation is used as in the addactionListner() with the game_Number being equal to 2.Which will filter down the if statements in finsihAction to call the finishing screen from the mainmenu which will calculate the scores and display the winner accordingly.

```
} else {
    MainMenu.setPlayer_2_score(score);//if second game
    //single player plasy as if its 2nd game avoids restarting
    //if single player display finish screen
    if (this instanceof SinglePlayer) {
        MainMenu.callFinishScreen( isSinglePlayer: true);//call finish screen single player true
    } else {
        MainMenu.callFinishScreen( isSinglePlayer: false);//call finsih screen single player false
    }
}
```

The else condition for the game_number==2

```
2 usages  artemkuc44
public static void callFinishScreen(boolean isSinglePlayer){

    frame.getContentPane().removeAll(); // Remove all content

    FinishScreen FS = new FinishScreen(player_1_score, player_2_score, isSinglePlayer);
    frame.add(FS, BorderLayout.CENTER); // Adds the new game panel

    frame.validate(); // Validates the frame after changes
    frame.repaint(); // Repaints the frame to display the new content

}
```

Heres the callFinishScreen which reuses the existing frame to display the revised finishing screen.

Main():

To improve efficentcy and elimate the need for dubllicate code we removed the contents of main() and left it with the basic call to the displayMainMenu() which houses all the code to create buttons,panels and any labels to help display the main menu and its contents

```
3 usages   Paddy Buckley +1
private static JButton createRulesBtn() {
    JButton button = new JButton( text: "Rules");
    button.setFont(new Font( name: "Arial", Font.BOLD, size: 14 ));
    button.setPreferredSize(new Dimension( width: BUTTON_WIDTH/2, BUTTON_HEIGHT));
    // Set a rounded border without painting the background
    button.setFocusPainted(false);
    //button.setContentAreaFilled(false);
    return button;
}
artemkuc44 +1
public static void main(String[] args) {
    displayMainMenu();
}
```

createRulesBtn():

Again in a need to remove dublicate code a method that returns the rule button as the only diffence beteen the buttons is its location.

SetScores():

Setters were also added to the Two player mode to allows us to call the Finishing screen from the main menu class allowing us to reuse the same frame.

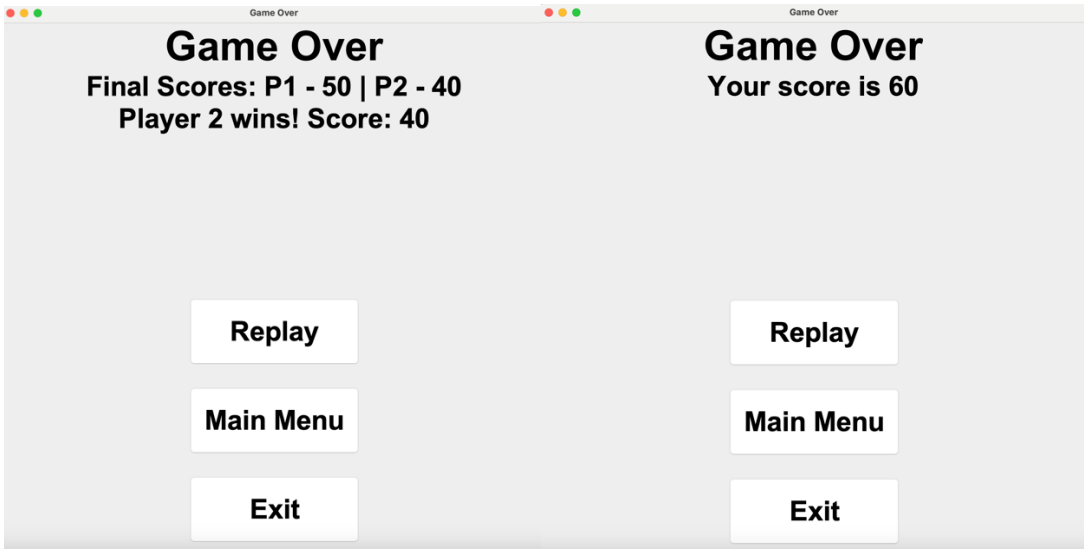
```
1 usage   artemkuc44
public static void setPlayer_1_score(int score) { player_1_score = score; }
1 usage   artemkuc44
public static void setPlayer_2_score(int score) { player_2_score = score; }
3 usages   Paddy Buckley +1 *
```

FinishScreen:

Due to the revised two player mode there was a need to update the Finishing screen to accomade this. We also revised the scoring system to mimic the brief. Now the winner is the user with the LOWEST score. The screen still contains the 3 buttons to replay,main menu or exit but the content of the text has changed and the size of the screen still remains 800*800 to make it seemless for the user.

Two Player Finishing screen:

Single Player Finishing Screen:



We maintained the minimalist display using the GridBagConstraints with the text changing based on what game mode is being player.

Finish Screen code:

Constructor: assigns player 1s score, player 2s score, and a check for if the game mode is single player.

```
public FinishScreen(int player1_score, int player2_score, boolean isSinglePlayer) {
    this.player1_score = player1_score;
    this.player2_score = player2_score;
    this.isSinglePlayer = isSinglePlayer;
    finishScreen();
}
```

The finishScreen method is used called in constructor, it is used to create buttons and assign them to action listeners aswell as add text on the screen.

```
1 usage   Paddy Buckley +1
private void finishScreen() {
    JFrame frame = MainMenu.frame;
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(MainMenu.DISPLAY_WIDTH, MainMenu.DISPLAY_HEIGHT);

    JPanel topPanel = new JPanel();
    topPanel.setLayout(new BoxLayout(topPanel, BoxLayout.PAGE_AXIS));

    JLabel gameOverLabel = createLabel( text: "Game Over", BIG_FONT);
    topPanel.add(gameOverLabel);

    if (!isSinglePlayer) {
        JLabel finalScoreLabel = createLabel( text: "Final Scores: P1 - " + player1_score + " | P2 - " + player2_score, SMALL_FONT);
        topPanel.add(finalScoreLabel);
        JLabel winnerLabel = createLabel(determineWinnerText(), SMALL_FONT);
        topPanel.add(winnerLabel);
    }
    else{
        JLabel finalScoreLabel = createLabel(  determineWinnerText(), SMALL_FONT);
        topPanel.add(finalScoreLabel);
    }

    JButton replayButton = createButton( text: "Replay");
    JButton mmButton = createButton( text: "Main Menu");
    JButton exitButton = createButton( text: "Exit");
```

```
JPanel buttonPanel = new JPanel(new GridBagLayout());

GridBagConstraints gbcReplay = new GridBagConstraints(); //css basically for starting button
gbcReplay.gridwidth = GridBagConstraints.REMAINDER; //skips line
gbcReplay.fill = GridBagConstraints.HORIZONTAL;
gbcReplay.insets = new Insets( top: 0,  left: 0,  bottom: 20,  right: 0); //padding for where the button is.
gbcReplay.anchor = GridBagConstraints.PAGE_END; //ending the css

GridBagConstraints gbcMM = new GridBagConstraints();
gbcMM.gridwidth = GridBagConstraints.REMAINDER;
gbcMM.fill = GridBagConstraints.HORIZONTAL;
gbcMM.insets = new Insets( top: 10,  left: 0,  bottom: 10,  right: 0);

GridBagConstraints gbcExit = new GridBagConstraints();
gbcExit.gridwidth = GridBagConstraints.REMAINDER;
gbcExit.fill = GridBagConstraints.HORIZONTAL;
gbcExit.insets = new Insets( top: 20,  left: 0,  bottom: 10,  right: 0);


buttonPanel.add(replayButton,gbcReplay);
buttonPanel.add(mmButton,gbcMM);
buttonPanel.add(exitButton,gbcExit);

frame.setLayout(new BorderLayout());
frame.add(topPanel, BorderLayout.CENTER);
frame.add(buttonPanel, BorderLayout.SOUTH);

frame.setLocationRelativeTo(null);
frame.setVisible(true);
 private JLabel createLabel(String text, Font font) {
     JLabel label = new JLabel(text, SwingConstants.CENTER);
     label.setFont(font);
     label.setAlignmentX(Component.CENTER_ALIGNMENT);
     return label;
 }
 3 usages  artemkuc44
 private JButton createButton(String text) {
     JButton button = new JButton(text);
     button.setFont(SMALL_FONT);
     button.setPreferredSize(new Dimension(BUTTON_WIDTH, BUTTON_HEIGHT));
     return button;
 }
```

```
// Add ActionListeners
replayButton.addActionListener(e -> replayGame(frame));
mmButton.addActionListener(e -> goToMainMenu(frame));
exitButton.addActionListener(e -> System.exit( status: 0));
```

The replayGame action listener method, starts up a new game depending on the current game mode. It Adds it to the frame removing the old game from it.

```
private void replayGame(JFrame frame) {
    if(isSinglePlayer) {
        SinglePlayer singlePlayerPanel = new SinglePlayer();
        frame.setTitle("Single Player");
        frame.getContentPane().removeAll(); //when its pressed, removes everything on screen
        frame.setSize( width: 800, height: 800);
        frame.setLocationRelativeTo(null); //makes it so when launching, it launches in the middle of the screen.
        frame.add(singlePlayerPanel, BorderLayout.CENTER); //adds the hex panel.
    }else{
        TwoPlayer twoPlayerPanel = new TwoPlayer( game_number: 1);
        frame.setTitle("Two Player");
        frame.getContentPane().removeAll(); //when its pressed, removes everything on screen
        frame.setSize( width: 800, height: 800);
        frame.setLocationRelativeTo(null); //makes it so when launching, it launches in the middle of the screen.
        frame.add(twoPlayerPanel, BorderLayout.CENTER); //adds the hex panel.
    }
    frame.validate(); //validates
    frame.repaint(); //painting
}
```

The go to main menu button is used to start up the main menu again, by calling the starter function located in main menu

```
private void goToMainMenu(JFrame frame) {
    MainMenu.frame.dispose();
    MainMenu.displayMainMenu();
    frame.dispose();
}
```

These methods determine the winner text based on who won and the relative player scores;

```java
private String determineWinnerText() {
    if (isSinglePlayer) {
        return "Your score is " + player2_score;
    } else {
        if (findWinner() == 1) {
            return "Player 1 wins! Score: " + player1_score;
        } else if (findWinner() == 2) {
            return "Player 2 wins! Score: " + player2_score;
        } else {
            return "It's a draw!";
        }
    }
}

2 usages  artemkuc44 +1
private int findWinner() {
    if(player1_score < player2_score){
        return 1;
    }
    if(player2_score<player1_score){
        return 2;
    }
    else {
        return 0;//draw
    }
```

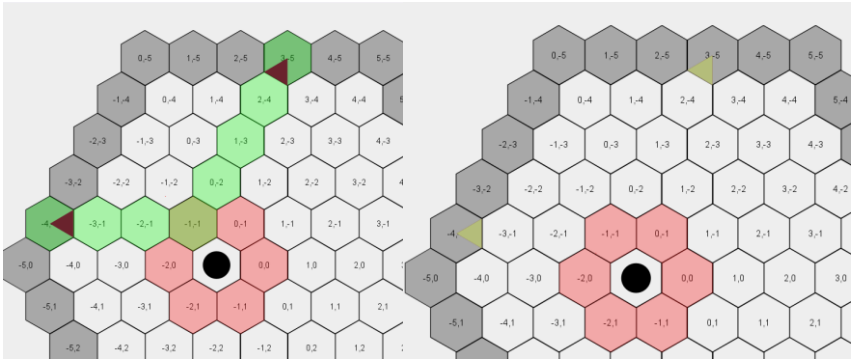
Sandbox game mode changes:

A few changed have been made regarding the sandbox game mode. In last sprints version the sandbox game mode simply created a hexBoard class when pressed (the class containing all of the core game mechanics), this was buggy as ray removal didn't

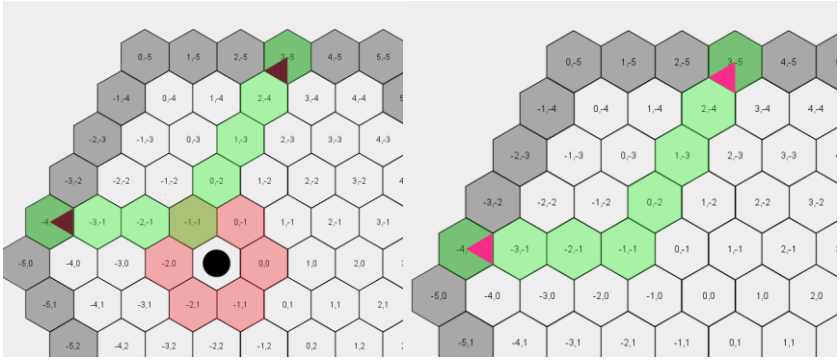
properly work, and ray paths were dodgy at times. Also ray movement wasn't dynamic. These faults are illustrated below:

Before:

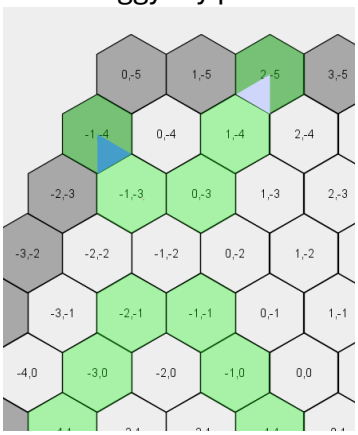
Attempting to remove ray



Attempting to remove atom after ray has been sent

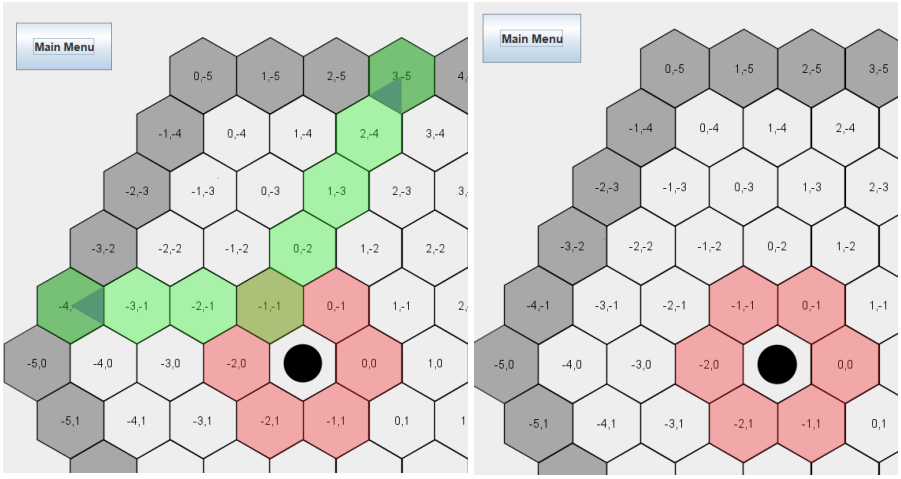


Buggy ray path

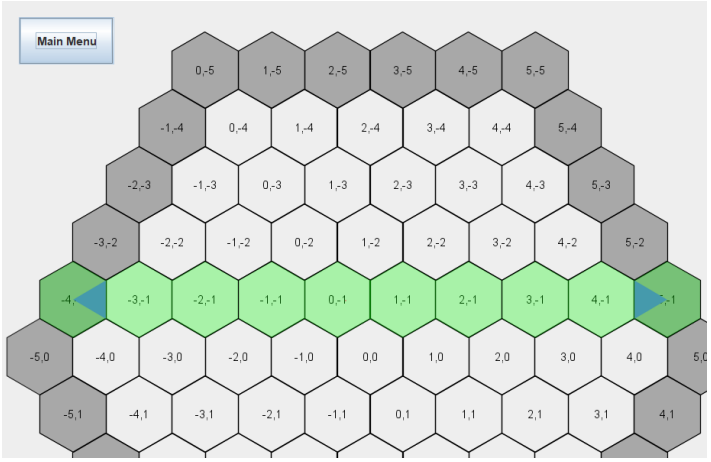


Now:

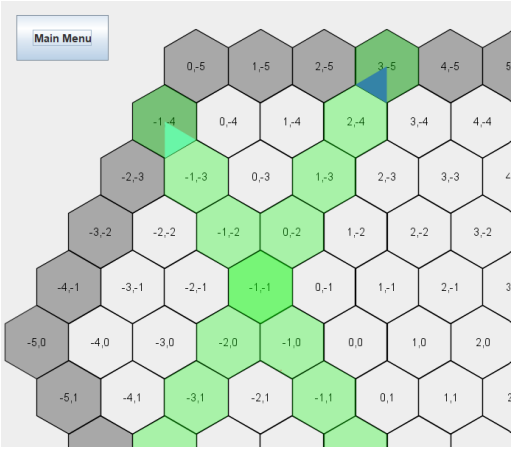
Removing ray



Dynamic rays
(After removing atom rays are redrawn)



Overlapping rays fix



These bug fixes allow for a much smoother user experience in the sandbox mode, allowing new users to easily and effectively get a grasp of game mechanics.

A main menu button was also added to return to main menu.

Sandbox code run through:

Simply extends main HexBoard class (base class)

```
2 usages   artemkuc44 +1 *
public class Sandbox extends HexBoard {
```

Main menu btn created and added to constructor:

```
1 usage   artemkuc44 *
public Sandbox(){
    addMainMenuBtn();
}
1 usage   Paddy Buckley +1 *
private void addMainMenuBtn(){
    JButton MainMenuButton = new JButton( text: "Main Menu");
    MainMenuButton.setBounds( x: 25,  y: 25,  width: 100,  height: 50);
    MainMenuButton.addActionListener(e -> ReturnToMainMenu());
    this.add(MainMenuButton);
    this.setLayout(null);
    MainMenuButton.setVisible(true);
}


1 usage   Paddy Buckley
private void ReturnToMainMenu() {
    MainMenu.frame.dispose();
    MainMenu.displayMainMenu();
}
3 usages   artemkuc44
```

Rays recalculated after every mouseClick for dynamic ray paths.

```java
@Override
protected void handleMouseClick(Point hexCoord, Point clickedPoint) {
    super.handleMouseClick(hexCoord, clickedPoint);
    recalculateRays();
    repaint();
}


1 usage   artemkuc44 *
protected void recalculateRays() {
    ArrayList<Ray> recalculatedRays = new ArrayList<>();
    rayMovement.clear();

    for (Ray ray : rays) {
        Ray newRay = new Ray(ray.getEntryPoint(), ray.getEntryDirection()); // Recreate the ray to reset its path and effects
        newRay.setR(ray.getR());//keep ray colour
        newRay.setG(ray.getG());//^^
        newRay.setB(ray.getB());//^^

        moveRay(newRay, atoms);//move each ray again
        recalculatedRays.add(newRay);//
    }

    rays.clear();//clear all old rays
    rays = recalculatedRays;//reassign to new
}
```

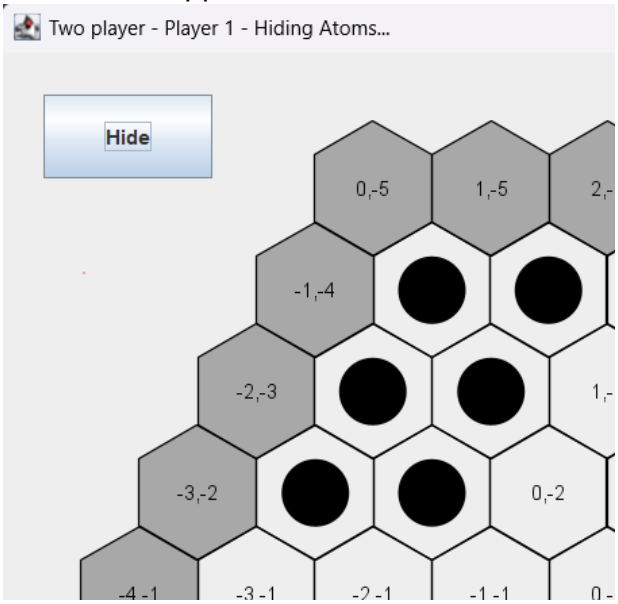
Two Player Game mode changes:

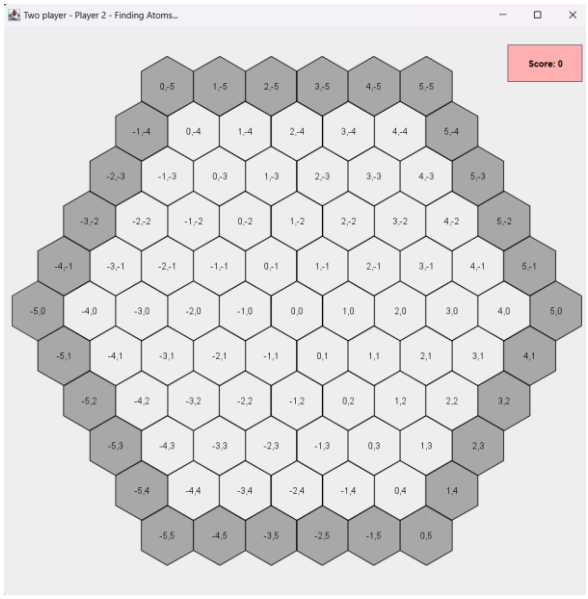
The two player game mode experienced some modifications.
Score has been modified in relation to the brief. +5 for incorrectly guessed atoms, +1 per ray marker.

The game mode now requires the players to switch half way through ie. Player 2 gets a chance to place atoms and player 1 guesses.
This is made clear thanks to the general button text and panel title illustrated below:

Game walk through

Player 1s turn – 6 atoms placed, “Hide” button appears for player 1 to press to hide the atoms.
Once it is pressed the button disappears along with the atoms, the text changes, scoreboard appears.





Player 2s turn – 6 atoms need to be guessed for the compare button to appear.

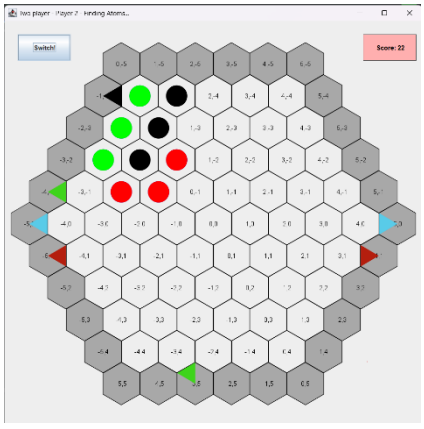
Atoms are then compare (green correctly guess, red incorrectly guessed, black not found).
Score is updated and the switch button appears. When switch is pressed the game restarts but the players are switched, this is shown in the panel title.
The game plays through again, displaying the results at the end

Two player new code run through:

Button text logic is fairly simple, “Hide” and “Compare” text is displayed uniformly throughout game modes, while if it’s the first game at the end the “Switch!” button appears while in the second game the “End Game” text appears.

```
2 usages   artemkuc44
private void updatebuttonState() {
    if (comparing) {//if in comparison mode, button used to end game or switch dependi
        if(game_number == 1){
            button.setText("Switch!");
        }else{
            button.setText("End Game");
        }
        button.setVisible(true);
    } else if (playerOneAtoms.size() == MAX_ATOMS && currentPlayer == 1) {//if player
        button.setText("Hide");
        button.setVisible(true);
    } else if (playerTwoGuesses.size() == MAX_ATOMS && currentPlayer == 2) {//if playe
        button.setText("compare");
        button.setVisible(true);
    } else {
        button.setVisible(false);
    }
}
```

The flags are updated based on the updateButtonAction method shown below, a try catch block is used for error handling. Feel free to read through the well commented code.



```
void updateButtonAction() {
    try {//try catch block for error handling
        if (currentPlayer == 1) {//if current player is 1 and button pressed move to next player

            currentPlayer = 2;
            scoreBoard.setVisible(true);
            button.setVisible(false);
        } else if (currentPlayer == 2 && comparing) {//if curr player 2 and comparing and button pressed move to end game stge
            endGame = true;
            if (game_number == 1) {//if first game
                MainMenu.setPlayer_1_score(score);//store score in mainMenu
                MainMenu.restartTwoPlayerGame();//create and play another game with roles "switched"
            } else {
                MainMenu.setPlayer_2_score(score);//if second game
                //single player plasy as if its 2nd game avoids restarting
                //if single player display finish screen
                if (this instanceof SinglePlayer) {
                    MainMenu.callFinishScreen( isSinglePlayer: true);//call finish screen single player true
                } else {
                    MainMenu.callFinishScreen( isSinglePlayer: false);//call finsih screen single player false
                }
            }
        } else if (currentPlayer == 2) {//if curr player 2 and btn pressed move to comparing stage
            comparing = true;
        }

        repaint();//call repaint after every change
```

Fixed bug , was able to place and remove atoms while comparing. This is avoided by adding the !comparing condition making sure the registered mouseClicks couldn't add or remove atoms.

```
@Override
protected void handleMouseClick(Point hexCoord, Point clickedPoint) {
    if (currentPlayer == 1 && !comparing) {
        if (hexCoordinates.contains(hexCoord)) {//if click within board
            Atom existingAtom = findAtomByAxial(playerOneAtoms, hexCoord);//try
            if (existingAtom != null) {//if clicked and it exists, remove it
                playerOneAtoms.remove(existingAtom);
            } else if (playerOneAtoms.size() != MAX_ATOMS) {
                //atom doesn't exist, create and add to arraylist;
                Atom newAtom = new Atom(hexCoord);
                playerOneAtoms.add(newAtom);
            }
        }
    } else if (currentPlayer == 2 && !comparing) {
        if (hexCoordinates.contains(hexCoord)) {//if click within board
            Atom existingAtom = findAtomByAxial(playerTwoGuesses, hexCoord);//tr
            if (existingAtom != null) {
                // Atom exists, so remove it
                //existingAtom.updateNeighbours();
                playerTwoGuesses.remove(existingAtom);
            } else if (playerTwoGuesses.size() != MAX_ATOMS) {
                // Atom doesn't exist, create and add to arraylist;
                Atom newAtom = new Atom(hexCoord);
                playerTwoGuesses.add(newAtom);
                //newAtom.updateNeighbours();
            }
```

Dynamic ray score calculation logic in handleMouseClick().

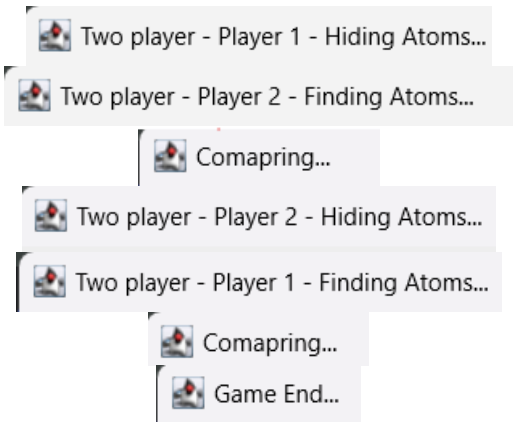
```
        if(ray.getEntryPoint().equals(ray.getExitPoint()) || ray.getType() == 1){
            score++;//inc score
        }else{
            score += 2;
        }
        scoreBoard.setText("Score: " + score);//update scoreboard text
```

Rest of the score calc logic, flag used to avoid recalculation (initially false).

```
    if(!scoreCalcluatedFlag){//if score hasnt yet been calculated
        score += (HexBoard.MAX_ATOMS-guessedCorrectly.size())*5;//calc score
        scoreCalcluatedFlag = true;//mark as calculated
    }
    scoreBoard.setText("Score: " + score); // Update the score display
```

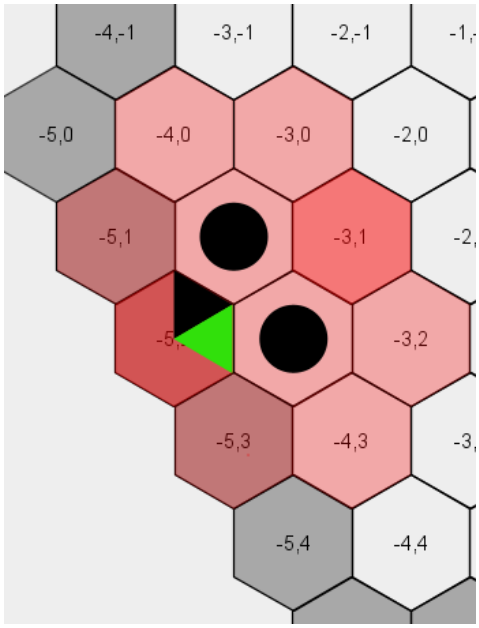
Panel Text visible below:

```
private void setTitleText() {
    //if its the second game being player, the players titles are swapped(player 2 is now playing as player 1).
    if(!comparing) {
        String player = (currentPlayer == 1) ? "Player 1" : "Player 2";
        String action = (currentPlayer == 1) ? "Hiding Atoms..." : "Finding Atoms...";

        if (game_number == 2) {
            player = (currentPlayer == 1) ? "Player 2" : "Player 1";
        }

        String title = "\t\tTwo player - " + player + " - " + action;
        MainMenu.frame.setTitle(title);//set game info at top
    }else{
        MainMenu.frame.setTitle("Comapring...");//set game info at top

    }
}
```

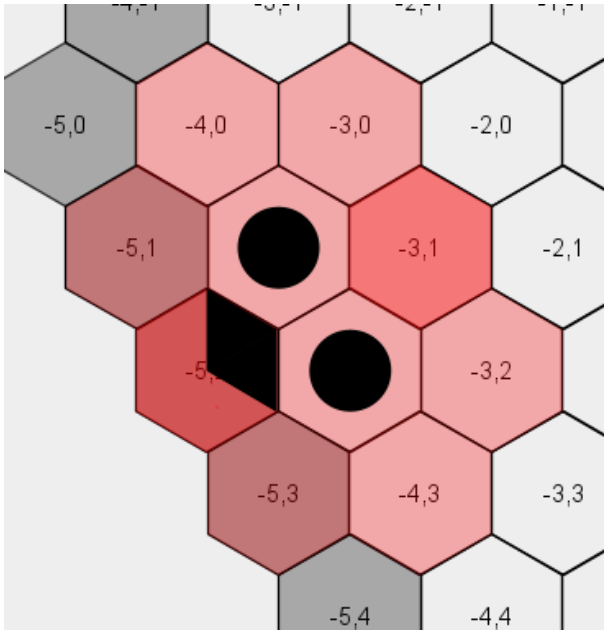


Bug fix:

Before:



After:



Added an extra check to moveRay method to check if any other atom is located along border:

```
for(Atom atom:atomsList){//traverse atom array
    if(atom.getNeighbours().containsKey(ray.getEntryPoint())){//checks for deflection with circle of influence on border
        ray.setExitPoint(ray.getEntryPoint());//deflects back to itself
        if(atomsContainsCoord(new Point( x: ray.getDirection().x + ray.getEntryPoint().x, y: ray.getDirection().y + ray.getEntryPoint().y)
                ,atomsList)){//checks if next hex in ray path contains atom
            ray.setType(1);//absorption
        }
        ray.setdirection(new Point( x: ray.getDirection().x*-1, y: ray.getDirection().y *-1));//stops from markers being drawn back to back
        return;
    }
    if(atom.getNeighbours().containsKey(currPoint)){
        // Retrieve the direction from the atom to the Neighbour (which is the key's value)
        Point neighbourDirection = atom.getNeighbours().get(currPoint);
        // Add directions (deflection)
        ray.setdirection(new Point( x: ray.getDirection().x + neighbourDirection.x,  y: ray.getDirection().y + neighbourDirection.y));
        numNeighboursEncountered++;
    }
}
```

Helper method created to traverse an array of atoms checking if a coordinate is equal to and atoms position

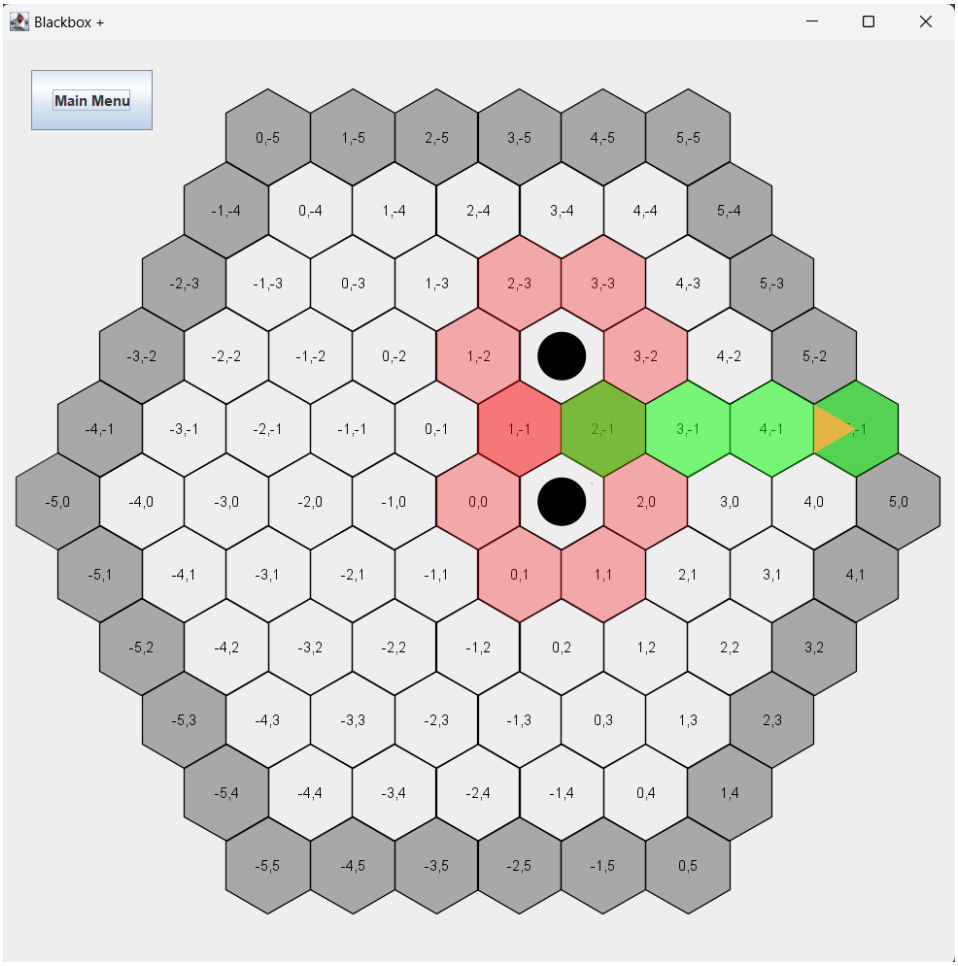
```
1 usage   new *
public boolean atomsContainsCoord(Point coord,ArrayList<Atom> atomArray){
    for(Atom atom:atomArray){
        if(atom.getPosition().equals(coord)){
            return true;
        }
    }
    return false;
}
```

Testing

```
@Test
void testRayAbsorption() {
    Atom atom1 = new Atom(new Point( x: 2, y: -2));
    Atom atom2 = new Atom(new Point( x: 1, y: 0));

    hexGridPanel.atoms.add(atom1);
    hexGridPanel.atoms.add(atom2);

    Ray ray1 = new Ray(new Point( x: 5, y: -1),new Point( x: -1, y: 0));

    hexGridPanel.moveRay(ray1, hexGridPanel.atoms);

    assertEquals(ray1.getEntryPoint(), ray1.getExitPoint(),  message: "Ray should be absorbed by the " +
            "aligned atoms.");
}
```

Aim: Ray gets absorbed when against 2 atoms

Process: created 2 atoms and also a ray with direction (-1,0) which means it is going left. I then call the function moveRay(); which moves the ray. Essert Equals compares its entry and exit point which they should be the same.




```java
@Test
void FindAtomByPoint() {
    Point testPoint = new Point( x: 1,  y: 1);
    Atom atom = new Atom(testPoint);
    hexGridPanel.atoms.add(atom);

    Atom foundAtom = hexGridPanel.findAtomByAxial(hexGridPanel.atoms,testPoint);
    assertNotNull(foundAtom);
    assertEquals(testPoint, foundAtom.getPosition());

    //try find non-existing atom
    Point nonExistingPoint = new Point( x: 4,  y: 4);
    assertNull(hexGridPanel.findAtomByAxial(hexGridPanel.atoms,nonExistingPoint));
}
```

Aim: Makes sure the atom was correctly added and can be found by its point

Process: Creates point which is location of atom and creates atom. Uses the function “findAtomByAxial” to find atom by its co-ordinates. AssertNotNull makes sure it is not null (the atom is correctly found). Also did the test again below but this time I didn’t actually create the atom hence why I used assertNull as it should be null.

```
@Test
void handleMouseTesterP2() {
    TwoPlayer.currentPlayer = 2; // Set to player 2
    Point hexCoord = new Point( x: 2,  y: 2);
    Point clickedPoint = new Point( x: 200,  y: 200);

      game.handleMouseClick(hexCoord, clickedPoint);

    assertFalse(TwoPlayer.playerTwoGuesses.isEmpty(),  message: "Atom should be added to " +
            "playerTwoGuesses list");
}
```

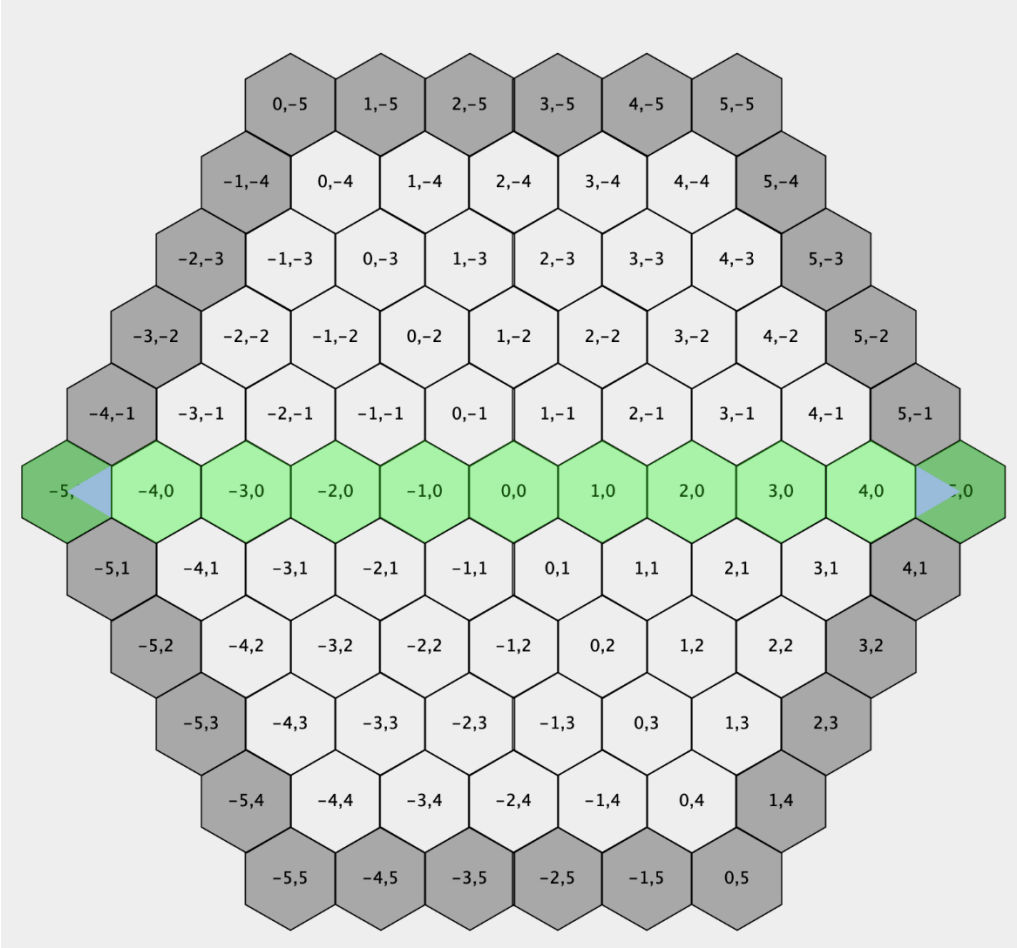
Aim: Testing the handleMouseClick for player 2.

Process: Sets player as 2. Creates co ordinates for atom to be placed and clicking point. Uses the function “handleMouseClick” function to add the atom. I then call the function isEmpty(); which will return a Boolean value wether the guesses are empty or not. AssertFalse means it should be false as I added an atom above as long as the function “handleMouseClick” worked properly.

```
@Test
void testRayNoInteraction() {
    Ray ray1 = new Ray(new Point( x: -5, y: 0),new Point( x: 1, y: 0));

    hexGridPanel.moveRay(ray1, hexGridPanel.atoms);

    assertEquals(new Point( x: 5, y: 0), ray1.getExitPoint(),  message: "Ray should go straight through " +
            "the board without any change in direction");
}
```

Aim: Testing ray path if it comes across nothing

Process: Creates ray with location (-5, 0) and direction (1,0) meaning it is going left. Used moveRay() function to move the ray. Assert equals used to compare the exit point and the co-ordinate it should exit which is just straight left.


0,-5
1,-5
2,-5
3,-5
4,-5
5,-5
-1,-4
0,-4
1,-4
2,-4
3,-4
4,-4
5,-4
-2,-3
-1,-3
0,-3
1,-3
2,-3
3,-3
4,-3
5,-3
-3,-2
-2,-2
-1,-2
0,-2
1,-2
2,-2
3,-2
4,-2
5,-2
-4,-1
-3,-1
-2,-1
-1,-1
0,-1
1,-1
2,-1
3,-1
4,-1
5,-1
-4,0
-3,0
-2,0
-1,0
0,0
1,0
2,0
3,0
4,0
-5,1
-4,1
-3,1
-2,1
-1,1
0,1
1,1
2,1
3,1
4,1
-5,2
-4,2
-3,2
-2,2
-1,2
0,2
1,2
2,2
3,2
-5,3
-4,3
-3,3
-2,3
-1,3
0,3
1,3
2,3
-5,4
-4,4
-3,4
-2,4
-1,4
0,4
1,4
-5,5
-4,5
-3,5
-2,5
-1,5
0,5

```java
@Test
void TestingSize() {
    TwoPlayer.currentPlayer = 2; // Set to player 2
    Point hexCoord = new Point( x: 2,  y: 2);
    Point clickedPoint = new Point( x: 200,  y: 200);
    Point hexCoord2 = new Point( x: 5,  y: 5);
    //Point clickedPoint2 = new Point(500, 500);


    game.handleMouseClick(hexCoord, clickedPoint);

    TwoPlayer.playerTwoGuesses.add(new Atom(hexCoord));
    TwoPlayer.playerTwoGuesses.add(new Atom(hexCoord2));
    assertEquals( expected: 3, TwoPlayer.playerTwoGuesses.size());
}
```

Aim: Test the size of atoms when different methods are used to add atoms.

Process: Sets the player to 2. Creates co ordinates for two atoms and also a clicking point for one of the atoms as that will use the function “handleMouseClick”. I then call the function “size()” which returns the size.

```java
@Test
void RandomPlacement() {
    //making sure if amount of atoms placed matches usual amount.
    assertEquals(HexBoard.MAX_ATOMS, TwoPlayer.playerOneAtoms.size(),  message: "Make sure that" +
            " all atoms are placed.");

    // check to see the ones auto placed are actually in the board
    boolean allValid = TwoPlayer.playerOneAtoms.stream().allMatch(atom ->
            game.hexCoordinates.contains(atom.getPosition()));
    assertTrue(allValid,  message: "All atoms placed by robot are in board.");
}
```

Aim: Making sure all atoms placed in single player are in the hexboard.

Process: First I use assert equals to make sure that all the atoms are placed by comparing the atom.size(); to 6 which is the number of atoms which should be placed. I then create Boolean variable which will check all atoms to make sure they are included in the games hex co-ordinates and if they are not, the Boolean will be false.

```java
@Test
void handleMouseTester() {
    Point hexCoord = new Point( x: 1,  y: 1); //random hex place
    Point clickedPoint = new Point( x: 100,  y: 100); // random place

     game.handleMouseClick(hexCoord, clickedPoint); //clicks on atom to add it


    assertFalse(TwoPlayer.playerOneAtoms.isEmpty(),  message: "Atom should be added to" +
            " playerOneAtoms list");
    //above test is making sure that an atom is in the list of "playerOneAtoms"
}
```

Aim: Make sure that the handleMouseClick works and also adds to player one atoms when player is not specified.

Process: Creates co ordinates and points which is where the atom will be placed. I then call the “handleMouseClick” to add an atom. Assert false is used to make sure that “isEmpty()” returns false as it shouldn’t be as I just added an atom above.