Azalea Rohr
Homework 8
April 1st, 2024
Collaboration: None

Question 1

For this question, I moved both my i32 and f64 points clockwise and counterclockwise to ensure that the code worked to full functionality. Below is the output with my original i32 point being at (3,4) and the f64 point being at (3.5, 2.0). For the i32 point, I first create the point (3,4), then called the clockwise() method to move the i32 point to rotate 90 degrees and then the counterclockwise() method to rotate it 90 degrees counterclockwise. The 'assert_eq!(clockwise_point_i32.x, -4);' checks that the clockwise rotation of the x coordinate is now -4 and 'assert_eq!(clockwise_point_i32.y, 3);' checks that the y coordinate is now at 3. The same test is doen with couterclcokwise with the statements 'assert_eq!(counterclockwise_point_i32.x, 4);' and 'assert_eq!(counterclockwise_point_i32.y, -3);' to check that the counterclockwise rotation is now at (4,-3). I also printed the original point, clockwise rotation and counterclockwise rotation.

For the f64 point, I first created a point at (3.5,2.0) and using the clockwise() and counterclockwise() methods I rotated it clockwise and counterclockwise respectfully. To test that the 90 degree rotations were correct, I used the code 'assert!((clockwise_point_f64.x + 2.0).abs() < f64::EPSILON);' to ensure that the x coordinate becomes -2.0 with a small margin or error by using 'f64::EPSILON'. I did the same for the y coordinate in the clockwise rotation by using 'assert!((clockwise_point_f64.y - 3.5).abs() < f64::EPSILON);' to ensure that the rotation is now 3.5 with a small margin of error. A similar test was done with the counterclockwise method by using 'assert!((counterclockwise_point_f64.x - 2.0).abs() < f64::EPSILON);' and 'assert!((counterclockwise_point_f64.y + 3.5).abs() < f64::EPSILON);' to check that the x and y coordinate moved to 2.0 and -3.5 respectfully. I also printed the original f64 coordinate, clockwise rotation and counterclockwise rotation.

Original i32 Point: Point { x: 3, y: 4 }
Clockwise i32 Point: Point { x: -4, y: 3 }
Counterclockwise i32 Point: Point { x: 4, y: -3 }
Original f64 Point: Point { x: 3.5, y: 2.0 }
Clockwise f64 Point: Point { x: -2.0, y: 3.5 }

Counterclockwise f64 Point: Point { x: 2.0, y: -3.5 }

Question 2

Below is the output for Conway's game of life for a 16 by 16 board that evolves for 10 iterations. I used 'X' to denote live cells and '.' to denote dead ones. For my test, I did 4 separate 3 by 3 boards each with a cell positioned at (1,1) for boards 1-3 and the cell positioned at (2,2) for board 4. For board 1 I am checking where a cell with 2 live neighbors remains alive according to the rules of Conway's Game of Life with the statement 'assert_eq!(calculate_liveness(&board1, 1, 1), 1);. For board 2 I an checking if a dead cell with 3 live neighbors becomes alive, the dead cell at the position (1,1), should become alive at ('1') with the statement 'assert_eq!(calculate_liveness(&board2, 1, 1), 1);'. In board 3 a live cell with 4 live neighbors should die, using 'assert_eq!(calculate_liveness(&board3, 1, 1), 0);' the live cell at (1,1) dies ('0') due to overpopulation. And for board 4, I am checking for a dead cell with 2 live neighbors remains dead, using 'assert_eq!(calculate_liveness(&board4, 2, 2), 0);' the dead cell at (2,2) should remain dead ('0'). Each of these boards tests a different aspect of the calculate_liveness function to make sure that it works according to the rules of Conway's Game of Life.

Generation 1:

```
.X..............
..X.............
XXX.............
................
................
................
................
................
................
................
................
................
................
................
................
```

Generation 2:

```
................
X.X.............
.XX.............
.X..............
................
................
................
................
```

.....
.....
.....
.....
.....
.....
.....
.....

Generation 3:
.....
..X.....
X.X.....
.XX.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 4:
.....
.X.....
..XX.....
.XX.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 5:
.....
..X.....
...X.....
.XXX.....
.....

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 6:
.....
.....
.X.X.....
..XX.....
..X.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 7:
.....
.....
...X.....
.X.X.....
..XX.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 8:
.....
.....

..X.....
...XX....
..XX.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 9:
.....
.....
...X.....
....X....
..XXX....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Generation 10:
.....
.....
.....
..X.X....
...XX....
...X.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....