Genetic Algorithm: Genetic Landers

Team Number: 522

Team Members: Zenan Chen-1888976

Ayush Gupta-1817303

Ira Pantbalekundri-1423854

Problem Statement:

The problem is to evolve the spacecraft's landing path on a complex land figure. The best landers are selected and used to create new landers. The problem is solved using optimum speed for maximizing the fuel efficiency using Genetic Algorithm which eventually increases the number of spacecrafts landing in the safe zone.

Implementation Details:

Genetic code:

In Genetic Algorithm, we have population as the landers or the spacecrafts which are going to be launched on a complex land figure. Chromosomes represent one such solution to the problem. In the problem, chromosomes are the commands which the lander receives before being launched. They include the speed, angle, power and so on.

To begin with the GA, we first need to initialize the population. We are initializing the population to 200 which evolves with every succeeding generation. A fitness and selection function will then determine how close a lander has reached to the destination. Once the algorithm is initiated, it randomly generates a set of commands for every lander and then launches them.

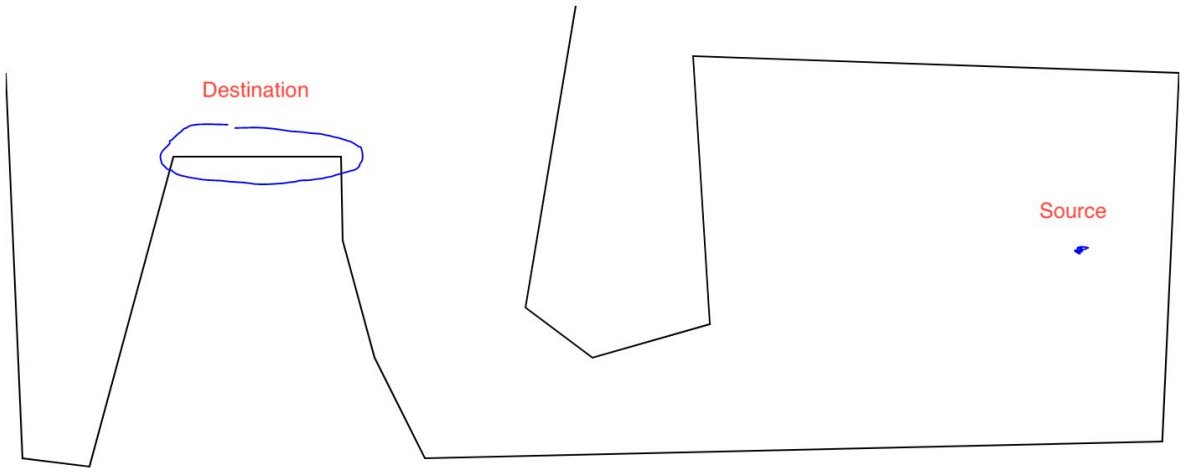


Fig 1: Landscape

The picture above is the land shape designed by us, using Java Graphics2D and Canvas. Our first generation of landers (or called spacecrafts) start flying from the start point with an initial power, and its speed & flying direction angle are generated randomly defined by their own commands (chromosomes). Due to the gravity, after first flying most of them could have crashed into ground. Their routes may look like the below image

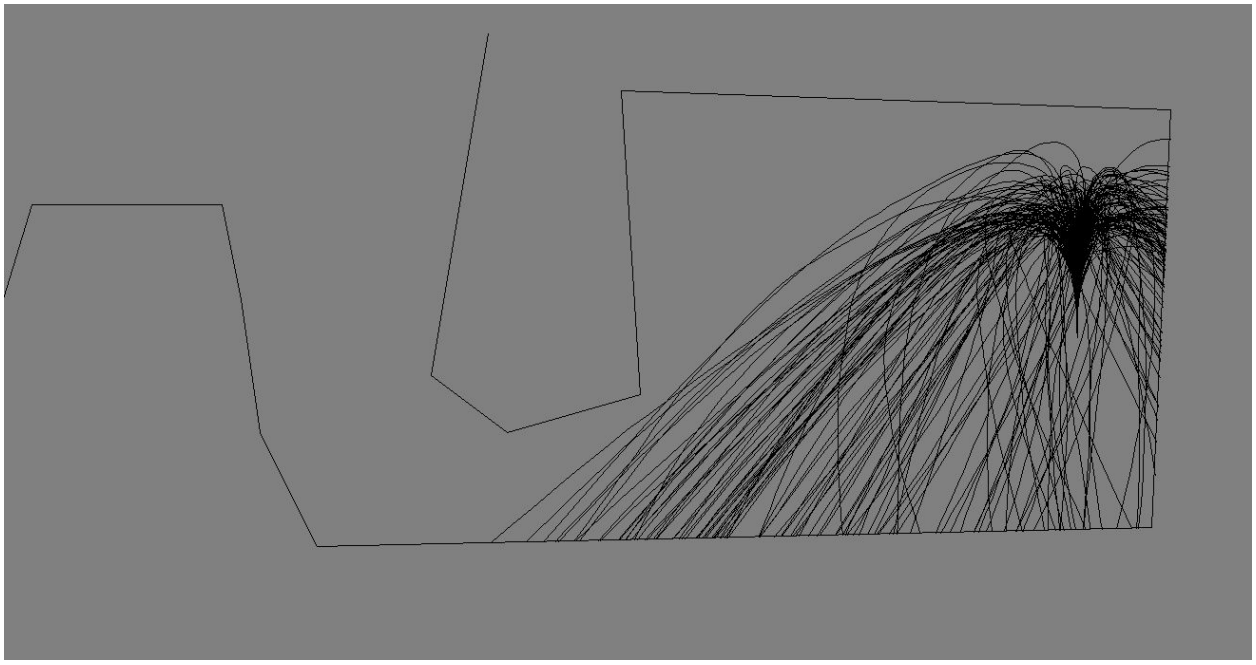


Fig 2: Fly routes of the 1st generation landers

We define 200 landers as the entire population and separate them into 4 sub population in order to speed up the crossover progress, each lander has configurations such like fuel limit, speed, accelerate rate, flying route points, etc. This process is repeated several times as the evolution.

After each flight we score those landers based on their performance: if one got closer to the target landing area with appropriate speed, it gets a higher score; if it crashed somewhere or flew out of the shape, it gets a lower score.

Then first 5 best landers are selected and used to breed new landers and replace the worst landers at same time to keep the total number of populations at 200.
We record their flying routes and visualize it in Canvas. The evolution progress looks like the below image

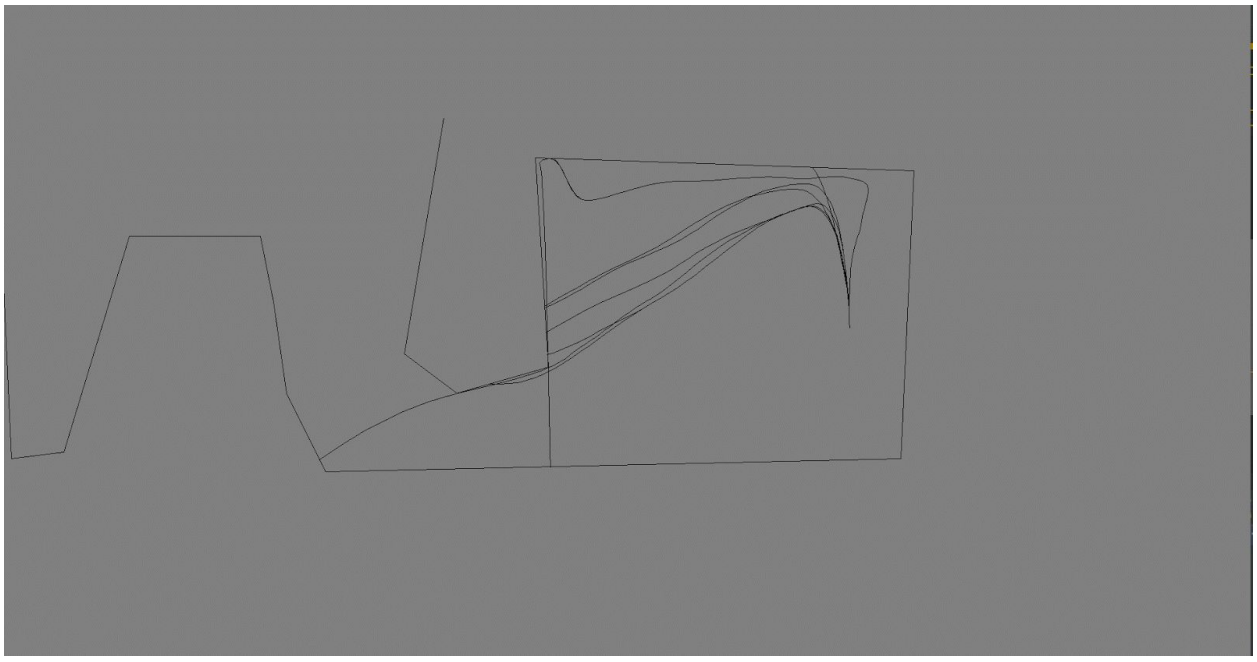


Fig 3: Fly routes after evolution

Fitness function:

The fitness function calculates the best score for every lander by taking into account the following parameters:

- The distance left to travel post-crash (i.e. the difference between the destination coordinates and the crash coordinates)
- Speed (i.e. if the speed exceeds the safety limit, it would account for a lower score)
- Fuel remaining (i.e. the fuel efficiency)

Following is the fitness function used in our algorithm:

```
public void calculateScore(Level level, boolean hitLandingArea) {
    double currentSpeed = Math.sqrt(Math.pow(this.xspeed, 2) + Math.pow(this.yspeed, 2));

    // 0-100: crashed somewhere, calculate score by distance to landing area
    if (!hitLandingArea) {

        double lastX = this.points.get(this.points.size()-2)[0];
        double lastY = this.points.get(this.points.size()-2)[1];
        double distance = level.getDistanceToLandingArea(lastX, lastY);

        // Calculate score from distance
        this.score = 100 - (100 * distance / level.max_dist);

        // High speeds are bad, they decrease maneuvrability
        double speedPen = 0.1 * Math.max(currentSpeed - 100, 0);
        this.score -= speedPen;
    }

    // 100-200: crashed into landing area, calculate score by speed above safety
    else if (this.yspeed < -40 || 20 < Math.abs(this.xspeed)) {
        int xPen = 0;
        if (20 < Math.abs(this.xspeed)) {
            xPen = (int)(Math.abs(this.xspeed) - 20) / 2;
        }
        int yPen = 0;
        if (this.yspeed < -40) {
            yPen = (int)(-40 - this.yspeed) / 2;
        }
        this.score = 200 - xPen - yPen;
        return;
    }

    // 200-300: landed safely, calculate score by fuel remaining
    else {
        this.score = 200 + (100 * this.fuel / this.initFuel);
    }
}
```

Fig 4: Fitness Function

Selection function:

The algorithm generates the score by taking into account the distance, speed and fuel remaining and stores it in an array in descending order.
The scores of the 200 landers are divided in sets of 5 consisting of 40 scores each. Now, the best 5 among 40 scores are selected for crossover. Hence, we get 20 pairs of parents form every top 5 best scores which produce a new generation of landers.

```
else {
    // Evolve each population independently
    for (int p = 0; p < NUMBER_OF_POPULATIONS; p++) {
        for (int i = REPRODUCING_LANDERS; i < LANDERS_IN_POPULATION; i++) {
            // Replace each low-value lander with a combination of two high-value landers
            int combinationCount = REPRODUCING_LANDERS * (REPRODUCING_LANDERS - 1);
            int combination = (i - REPRODUCING_LANDERS) % combinationCount;
            int momIndex = (int) Math.floor(combination / (REPRODUCING_LANDERS - 1));  // The "row"
            int dadIndex = combination % (REPRODUCING_LANDERS - 1);  // The "col"
            if (momIndex <= dadIndex) {
                // Indexes have to be different
                dadIndex += 1;
            }
            int populationOffset = p * LANDERS_IN_POPULATION;
            int childIndex = populationOffset+i;
            int a = 0;
            level.landers.get(populationOffset + i).inheritCommands(
                    level.landers.get(populationOffset + momIndex),
                    level.landers.get(populationOffset + dadIndex)
            );
        }
    }
```

Fig 5: Selection Function

Crossover function:

Every new lander inherits certain traits of the mother and father. Following these events, a new generation consisting of 200 landers is generated which replaces the worst landers at same time to keep the total number of population at 200

Following is the code for the inherit commands:

```java
public void inheritCommands(Lander mom, Lander dad) {
    Lander first = mom;
    Lander second = dad;
    if (Math.random() < 0.5) {
        // Randomly choose a first and a second parent
        first = dad;
        second = mom;
    }
    int minTimestep = Math.min(first.timestep, second.timestep);
    int crossoverFrom = (int) Math.floor(minTimestep / 3);
    int crossoverTo = (int) Math.floor(minTimestep * 2 / 3);
    int crossoverIndex = Helper.getRandomInt(crossoverFrom, crossoverTo);
    for (int i = 0; i < crossoverIndex; i++) {
        this.commands.get(i).setAngle(first.commands.get(i).getAngle());
        this.commands.get(i).setPower(first.commands.get(i).getPower());
        //this.commands.set(i, new Command(first.commands.get(i).getAngle(), first.commands.get(i).getPower()));
    }
    for (int i = crossoverIndex; i < second.commands.size(); i++) {
        this.commands.get(i).setAngle(second.commands.get(i).getAngle());
        this.commands.get(i).setPower(second.commands.get(i).getPower());
        //this.commands.set(i, new Command(second.commands.get(i).getAngle(), second.commands.get(i).getPower()));
    }
    if (Math.random() < 0.2) {
        // Angle
        int i = Helper.getRandomInt(0, this.commands.size() - 3);
        int avg = (this.commands.get(i).getAngle() + this.commands.get(i+1).getAngle() + this.commands.get(i+2).getAngle()) / 3;
        this.commands.get(i).setAngle(avg);
        this.commands.get(i+1).setAngle(avg);
        this.commands.get(i+2).setAngle(avg);
    }
    if (Math.random() < 0.2) {
        // Power
        int i = Helper.getRandomInt(0, this.commands.size() - 3);
        double avg = (this.commands.get(i).getPower() + this.commands.get(i+1).getPower() + this.commands.get(i+2).getPower() / 3);
        this.commands.get(i).setPower(avg);
        this.commands.get(i+1).setPower(avg);
        this.commands.get(i+2).setPower(avg);
    }
}
```

Fig 6: Crossover Function

Supporting Materials:

Unit Test:

Collision test:

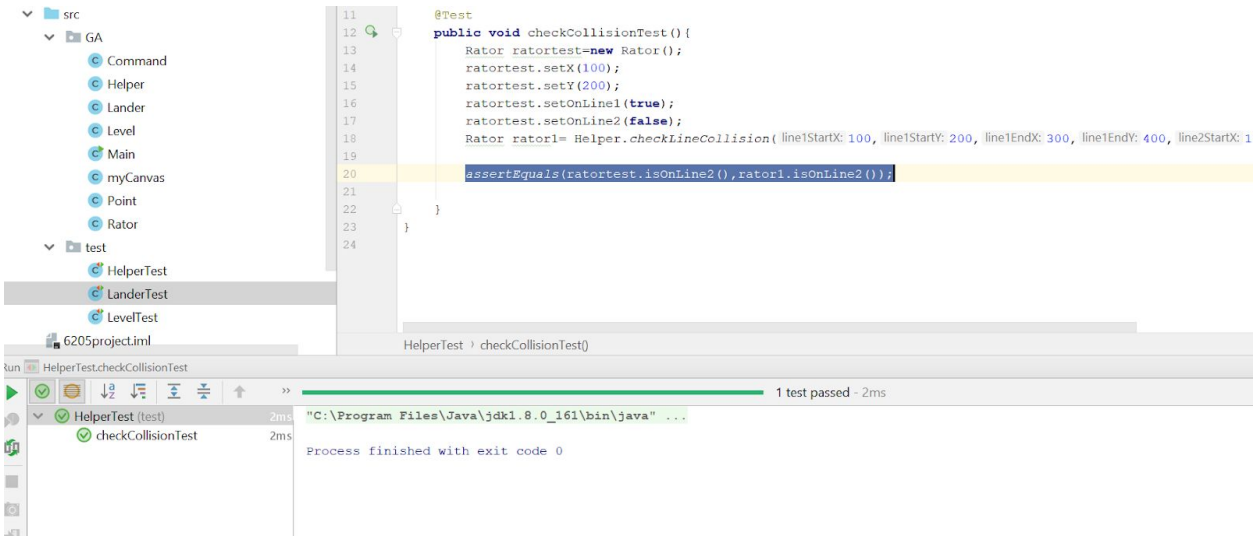


Fig 7: Collision Test

Lander test:

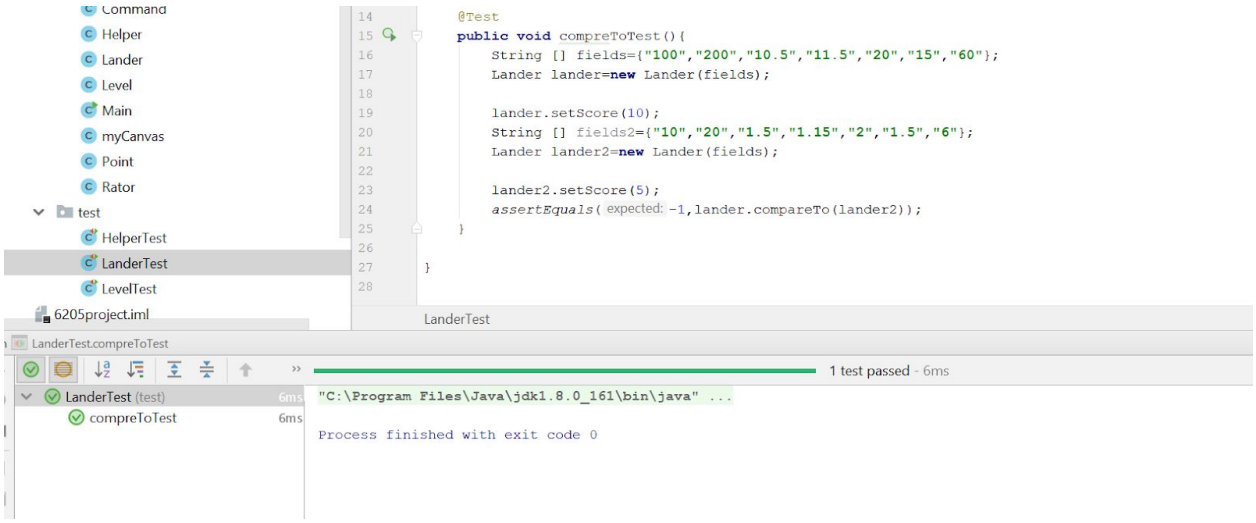


Fig 8: Lander Test

Level test:

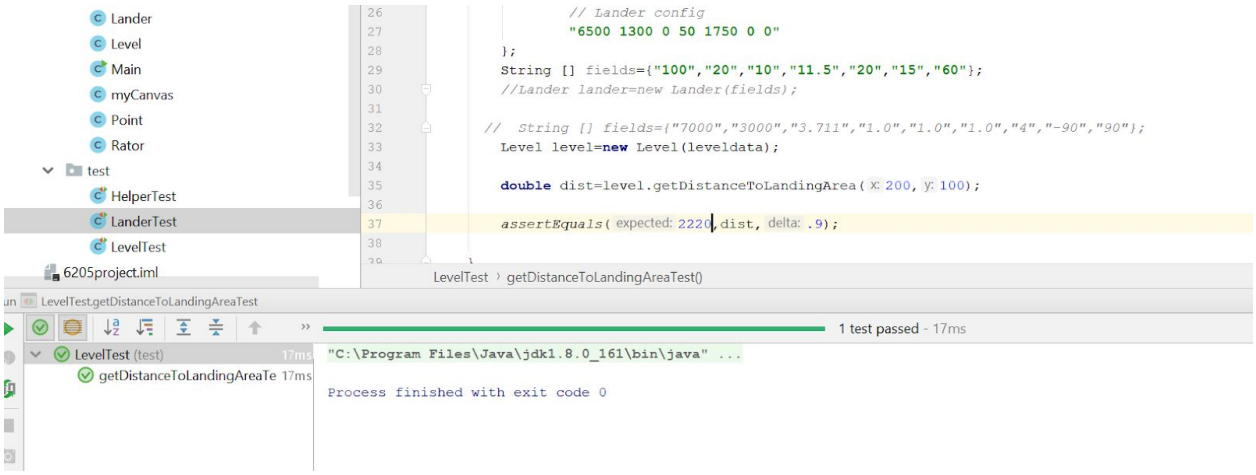
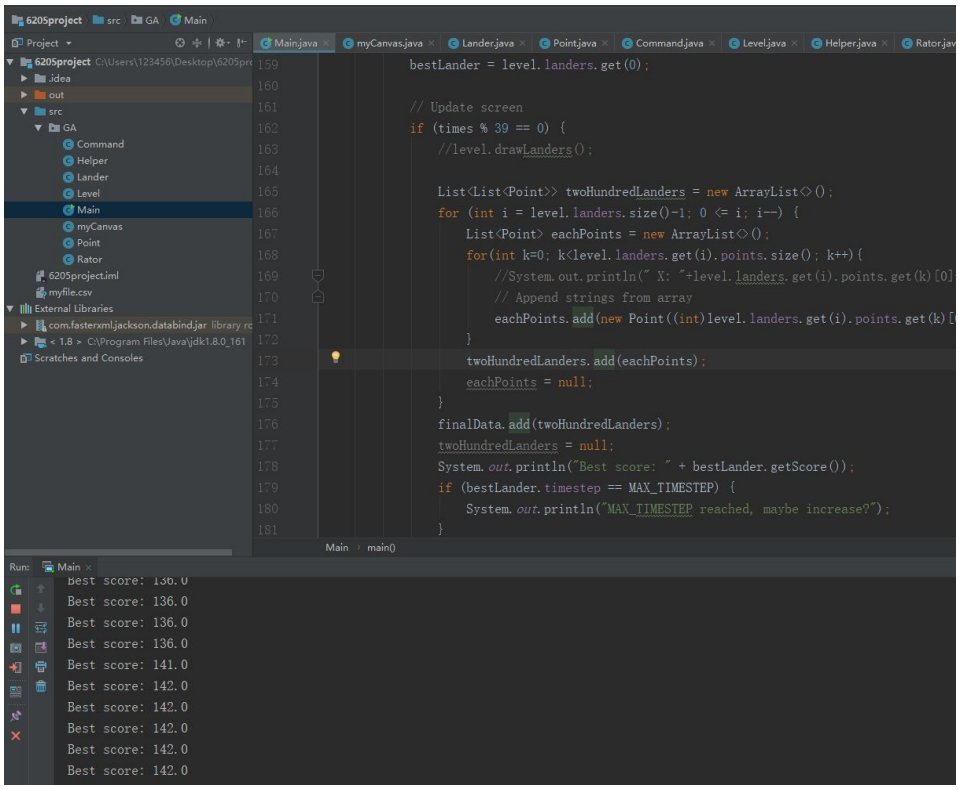
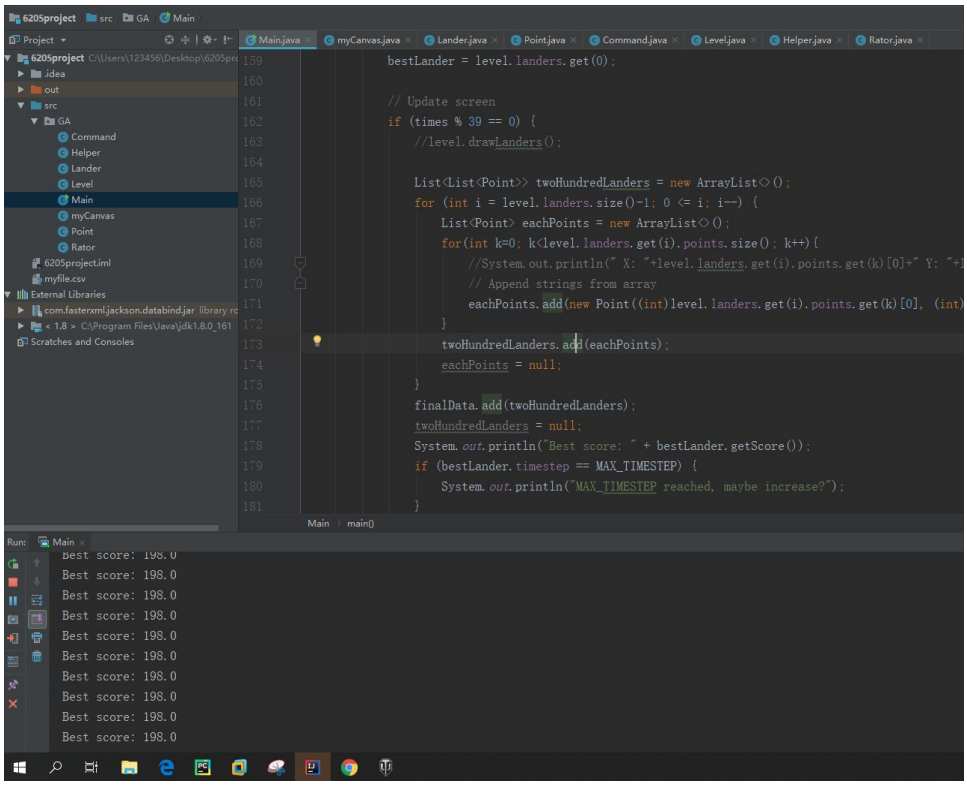


Fig 9: Level Test

Output:

Initially the best score started off at 136 and it progressed to 198





REFERENCES

- https://github.com/fafl/genetic-lander
- https://en.wikipedia.org/wiki/Genetic_algorithm
- https://www.tutorialspoint.com/genetic_algorithms/index.htm