**UnReal** **Battle Field** using **P**article **S**warm **O**ptimization

Project Members: -

1. Kinnar Kansara (001388213)
2. Bhavya Patel (001351292)
3. Yash Thakker (001389338).

[**Particle Swarm Optimization**](https://www.sciencedirect.com/topics/engineering/particle-swarm-optimization) is similar to the [genetic algorithm](https://www.sciencedirect.com/topics/engineering/genetic-algorithm) technique for optimization in that rather than concentrating on a single individual implementation, a population of individuals (a “swarm”) is considered instead. The algorithm then, rather than moving a single individual around, will move the population around looking for a potential solution. This is an example of a heuristic approach, where there is no guarantee of an optimal solution.

Each individual in the swarm has a position and velocity defined, the algorithm looks at each case to establish the best outcome using the current swarm, and then the whole swarm moves to the new relative location.

The position of each particle is represented by XY-axis position, and also, the velocity is expressed by Vx (the velocity of X-axis) and Vy (the velocity of Y-axis). Modification of the particle position is realized by the position and velocity information. Each particle knows its best value so far **(Pbest)** and its XY position. This information represents the personal experiences of each particle. Moreover, each particle knows the best value so far in the group **(gbest)** among **Pbests**. This information represents the knowledge of how the other particles around have performed.

This modification can be represented by the concept of velocity. Velocity of each particle can be modified by the following equation:

**vi (t+1) = wvi (t) + c1r1[XPbest(i) (t) − Xi (t)] + c2r2[XPbest(t)− Xi (t)]**

• i is the particle index

• w is the inertial coefficient

• c 1, c 2 are acceleration coefficients, 0 ≤ c 1, c 2 ≤ 2

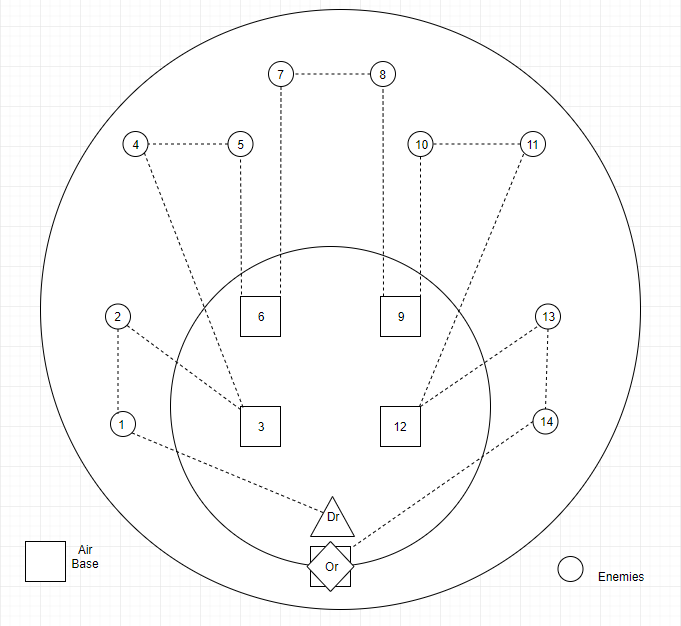
• r 1, r 2 are random values (0 ≤ r 1, r 2 ≤ 1) regenerated every velocity update

**PSO for Unreal Battle Field**

We have tried to implement an unreal battle field with a drone flying over a war zone and attack enemies with explosives. The drone will attack the targets and once it will identify that its explosives are reaching its minimum point, it will fly over to the nearest airbase, reload the explosives and again resume from the point where it left the attack.

Once it has attacked all the enemies it will return back to the airbase it started from. We have tried to implement PSO to find the shortest path the drone will travel to attack the enemies and return back to the origin.

Below is an example which elaborates how we are planning to go ahead with the execution of our battle field with the help of PSO.



In the above battle field, the drone will go and attack the first two targets and comeback to the air base to reload the explosive and go to next target. We are planning to generate the capacity of explosive in a specific range the drone can carry and once it reaches the minimum capacity it will reach out to its nearest air base to reload and will move ahead with the air strike.

The optimal path will be calculated with the help of PSO algorithm and the next move of the drone i.e. will it move towards the next target or the air base will be decided with the current capacity of explosives the drone is carrying.

The targets to attack and the air bases to reload the explosives are being randomly generated in a specific range, and accordingly will decide the optimal path.

**Implementation of Unreal Battle Field using PSO**

The **helper class** of the PSO package contains the shuffle array method in which a random array is generated which is helpful in calculating the velocity in Particle class. The shuffle array is further converted into different data types i.e. int and double.

The pBest value for a particular iteration is calculated in the **Particle class** of the PSO package, in this class the velocity is calculated by randomly generating numbers in the getRandomVelocity method. This velocity is used to calculate the shortest path for the drone to travel and destroy all the enemies.

In the **Swarm class** of the PSO package we have added particles in the swarm and calculated the fitness value in getFitnessValue method based on the previous target value. Once the fitness value is calculated we calculate the gBest from findGlobalBest method depending on that iterations pBest value and display it. Once we have calculated the gBest value we update the position depending on the velocity and according calculate the new gBest value for all other iterations. We have then updated the velocity using the formula stated above and also the optimal strike route based on distance in the Strike route method.

In the **war zone package**, we have the drone target and air base classes along with their directories. In the **drone class** we have randomly generated the payload capacity of the drone which signifies the number of explosives that particular drone can carry. The **drone directory** is used to add the drones in a list which will be used further to optimize which drone will be the most optimal one to launch first.

The **target class** contains the required payload which we have generated randomly, this required payload represents the number of explosives it requires to be exploded. And on the other hand, the **target directory** is used to add the targets in a list which will be used to optimize the route in target explosion.

In the **position class** we assign a range for the latitude and longitude where the air base can be found so that the drone can go and refill the explosives and then we have randomly assigned the air bases in that specific latitude and longitude range. We have also assigned random position to the target in this class, the target details have been fetched from the target directory class.

The **Air base class** contains the name and position, the position is extracted from the position class. These air bases are then added into a list in **air base directory class** where they will be used further by the drone to refill its explosives.

In the **War Zone Simulator class**, we have assigned the values for number of targets, drones, minimum target payload, maximum target payload, total number of particles, total number of iterations and the total air bases. Then we have implemented multi-threading for adding the particles in the swarm which will help us in simultaneous execution for calculation of the pBest value of particles. So, once we have calculated the pBest value we have invoked the findGlobalBest function from the Swarm class to calculate the gBest value from the available pBest.

The **War Zone Simulator Directory class**, we have calculated the distance between the targets and airbases. The optimal strike route is then calculated which helps the drone to decide the optimal path to attack the targets. And once the distance and the optimal path is calculated we print the distance matrix in which we display how far each target and air base is from each other.

In the **Graph class** of the graph package we have tried to display a line graph using JFree chart called particle progress which show cases the number of iterations on x-axis and the fitness value on y-axis for all the particles. In the category dataset method, we have passed the particle array which consist of all the particle information and the result array consist of all the iteration information.

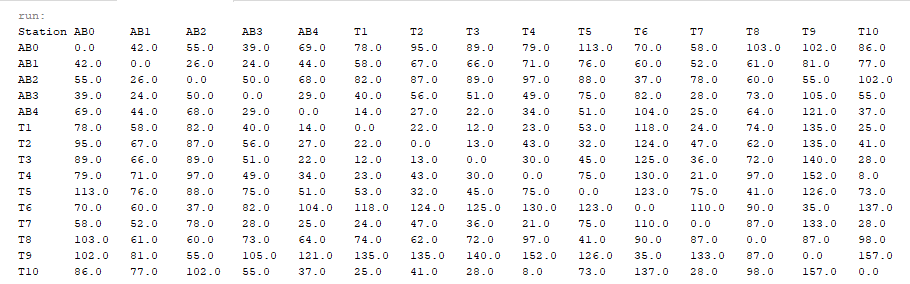
The **AnimationStorage** **class** in the Animation package has a list which contains the parent route which is optimum strike route along which the drone will pass. This route also consists of the distance between each target and air base, which will help the drone find the nearest air base to go and refill the explosives.

The **TimmerAnimationUtility class** is used to update the location of the drone based on the strike route and once the location is updated the init method in invoked to re-paint the further strike route.

The **AnimationBoard class** is used to display the JPanel on which the entire animation is displayed. It takes two arrays in which the x and y co-ordinates are stored for the JPanel. It then takes all the images to be displayed in the loadImage method. On the JPanel our air bases are indicated a square and all the enemies are displayed by a small circle. We have also displayed the strike path the drone will travel by dashed lines. So, once the drone will reach the target and destroy it the target will change its color from white to red.

**Output**

The console will first display the distance in miles of each air base and target with other air bases and targets which will help us to find the optimum strike route and from when and where the drone will have to go and refill the explosives for further attack.



Next, we are displaying the Drone details along with the payload capacity which is generated randomly. The payload capacity is the number of explosives each drone can carry.

----------------------------------------

Drone Details

----------------------------------------

Drone-1 Payload Capacity:6

Drone-2 Payload Capacity:7

Drone-3 Payload Capacity:8

Drone-4 Payload Capacity:9

Drone-5 Payload Capacity:10

Then we have displayed the target details which includes the various targets and the respected payload values i.e. explosives that each target will require to get eliminated.

----------------------------------------

Target Details

----------------------------------------

Target-1 Payload required:3

Target-2 Payload required:2

Target-3 Payload required:2

Target-4 Payload required:3

Target-5 Payload required:3

Target-6 Payload required:1

Target-7 Payload required:4

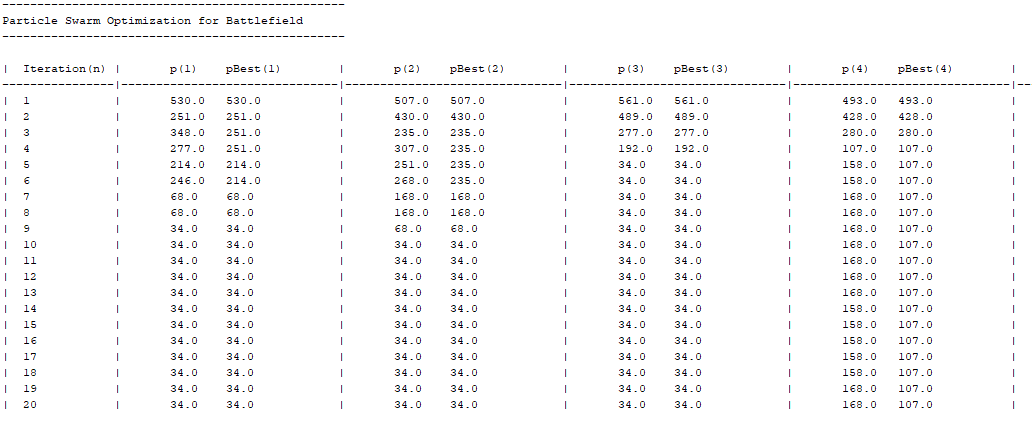
Target-8 Payload required:2

Target-9 Payload required:3

Target-10 Payload required:3

Then we have displayed the Particle Swarm Optimization for Battlefield which has 30 particles and their respected pBest values. We have then calculated the gBest value from the 30 particles and have repeated the procedure for calculation of pBest and gBest values for 20 iterations.

In the table displayed below we have also displayed a particular particles pBest along with the current pBest which is running from the previous iterations i.e. if a particular particles pBest is 30 but that particles pBest in previous iteration is 25 so we have displayed both the current and also the previous pBest.

So, once we have completed the procedure for 30 iterations, we have found an optimal path and the gBest value from all the 20 iterations we have executed.

Then we have displayed the Strike Path which will include the best fitness value from all the 20 iteration that have been executed and the optimal route the drone will have to follow to destroy all the targets.

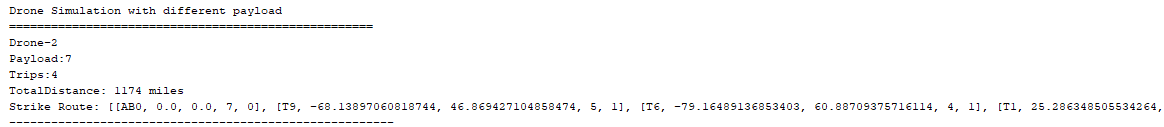
Strike Path

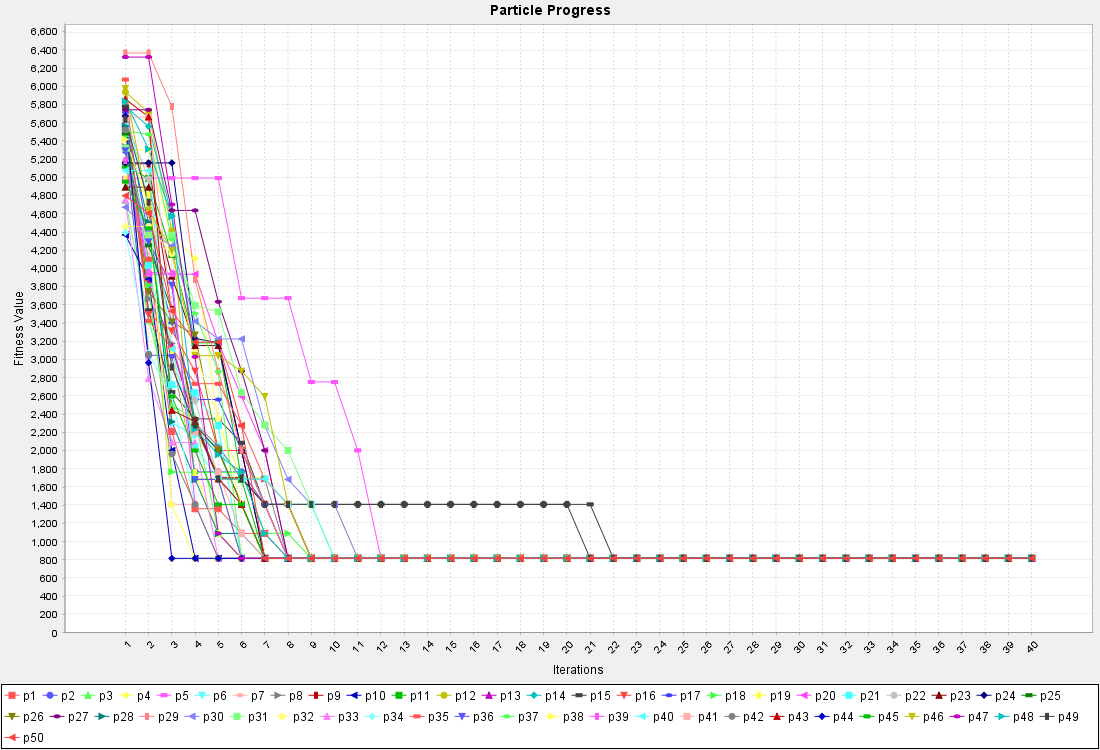
====================================================

gFitnessValue=34.0

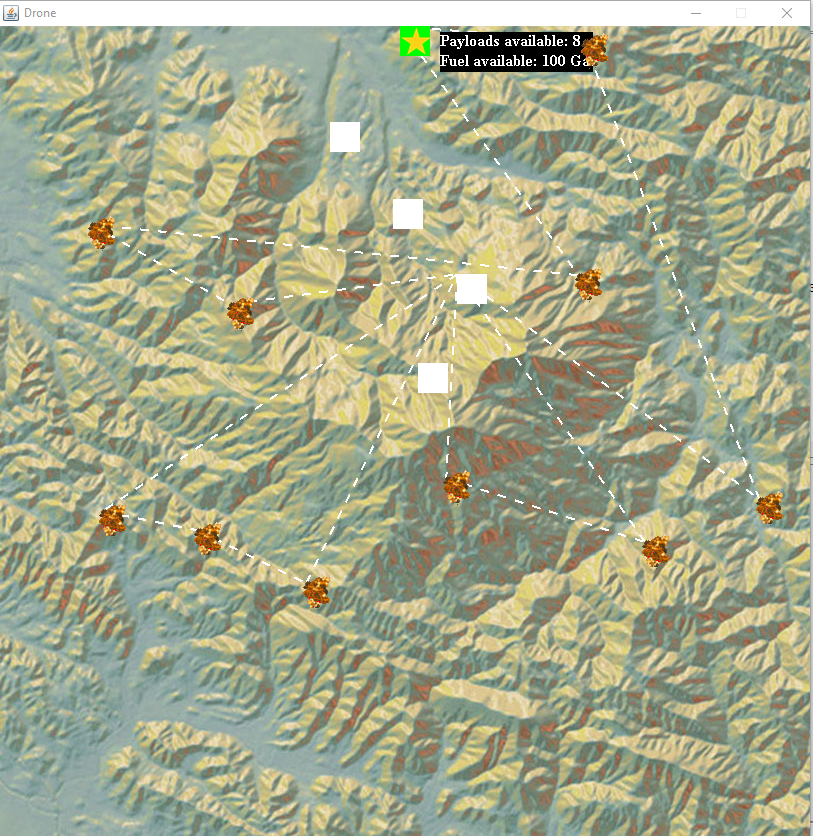
Optimal Strike Route: [2, 3, 1, 4, 5, 6, 7, 8, 9, 10]

Once we have calculated the optimum strike route, we have displayed the path each and every drone will follow based on its payload value and the distance it covers. We have also displayed the trips which determine in the entire air strike how many times the drone will go to the nearest air base to refill the explosives.



**Graph**

In the above graph we have displayed the iterations on Y axis and fitness values on X axis. Here we can observer that all the pBest are redirected towards the gBest.

**UnReal Battle Field**

In the above screen shot we have displayed our battle field after the targets are destroyed, and the drone has reached its origin. We have also displayed the current payload capacity of the drone along with the fuel capacity.

**References**

For PSO: -

* <https://www.sciencedirect.com/topics/engineering/particle-swarm-optimization>

For Animation: -

* <http://zetcode.com/tutorials/javagamestutorial/animation/>
* <https://www.clear.rice.edu/comp310/JavaResources/animation.html>

For Timer Task Execution: -

* <https://www.iitk.ac.in/esc101/05Aug/tutorial/essential/threads/timer.html>
* <https://www.baeldung.com/java-timer-and-timertask>