ASSIGNMENT REPORT

Assignment ID: 3

Student Name: Liu Leqi（刘乐奇）

Student ID: 12011327

DESIGN

Auction

There are two types of auction: Vickrey, and Dutch.

In Vickrey auction, the bidders has only one chance to bid, and the winner is the one with highest bid and the final saled price is the second highest bid.

In Dutch auction, it starts with a high bid and decreases the bid until there is a bidder stops and accepts the bid.

```java
// Conduct Vickrey auction among neighboring cameras
private void handoverEventVickrey(List<Camera> neighboringCameras, ObjectInterest objectToHandover) {
    double myBid = auctioneer.calculateBid(objectToHandover, AuctionStrategy.Vickrey);
    double highestBid = myBid;
    double secondHighestBid = 0;
    Camera winner = auctioneer;

    for (Camera neighbor : neighboringCameras) {
        double neighborBid = neighbor.calculateBid(objectToHandover, AuctionStrategy.Vickrey);
        if (neighborBid > highestBid) {
            secondHighestBid = highestBid;
            highestBid = neighborBid;
            winner = neighbor;
        } else if (neighborBid > secondHighestBid) {
            secondHighestBid = neighborBid;
        }
    }

    // Update Pheromones in Vision Graph
    for (Camera neighbor : neighboringCameras) {
        auctioneer.updatePheromones(neighbor, winner == neighbor);
    }

    // Handover if this camera is not the winner
    if (winner != auctioneer) {
        auctioneer.finalizeHandover(winner, objectToHandover, secondHighestBid);
        Network<Object> net = auctioneer.getNetwork(netName:"pheromoneNetwork");
        net.addEdge(auctioneer, winner, weight:0.0);
        System.out.println("A connection between " + auctioneer.getId() + " and " + winner.getId() + " ocurred");
    }
    System.out.printf(format:"My bid: %.2f - Bid winner: %.2f\n", myBid, highestBid);
}
```

```java
// Conduct Dutch auction among neighboring cameras
private void handoverEventDutch(List<Camera> neighboringCameras, ObjectInterest objectToHandover) {
    double myBid = auctioneer.calculateBid(objectToHandover, AuctionStrategy.Dutch);
    double curBid = myBid;
    double decayRate = 0.05;
    Camera winner = auctioneer;

    while (winner == auctioneer) {
        int idx = RandomHelper.nextIntFromTo(from:0, neighboringCameras.size() - 1);
        Camera randomCamera = neighboringCameras.get(idx);
        if (randomCamera.getRandomBid() > curBid) {
            winner = randomCamera;
            break;
        }
        curBid = curBid * (1 - decayRate);
    }

    // Update Pheromones in Vision Graph
    for (Camera neighbor : neighboringCameras) {
        auctioneer.updatePheromones(neighbor, winner == neighbor);
    }

    // Handover if this camera is not the winner
    if (winner != auctioneer) {
        auctioneer.finalizeHandover(winner, objectToHandover, curBid);
        Network<Object> net = auctioneer.getNetwork(netName:"pheromoneNetwork");
        net.addEdge(auctioneer, winner, weight:0.0);
        System.out.println("A connection between " + auctioneer.getId() + " and " + winner.getId() + " ocurred");
    }
```

```
    System.out.printf(format:"My bid: %.2f - Bid winner: %.2f\n", myBid, curBid);
}
```

Utility Calculation

According to the instruction, the utility of a camera can be calculated as below:

$$
\begin{aligned}
U_i(O_i, p, r) &= \sum_{j \in O_i} u_i(j) - p + r \\
&= \sum_{j \in O_i} [c_j \cdot v_j \cdot \phi_i(j)] - p + r
\end{aligned}
$$

For calculating the part: $\sum_{j \in O_i} [c_j \cdot v_j \cdot \phi_i(j)]$, where c_j is the confidence, v_j is the visibility, $\phi_i(j)$ is the characteristic function indicating whether the camera is holding the object, we implemented methods as below:

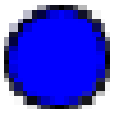
```
public double calculateUtility(ObjectInterest objectOfInterest) {
    // Example: Calculate utility based on object importance and threat level
    // double importance = objectOfInterest.getImportance(); // Example: 0 to 100
    // double threadLevel = objectOfInterest.getThreadLevel(); // Example: 0 to 100
    // double totalPheromoneStrength =
    // pheromoneStrengths.values().stream().mapToDouble(Double::doubleValue).sum();

    // Utility calculation
    double confidence = calculateConfidence(grid.getLocation(objectOfInterest));
    double visibility = calculateVisibility(grid.getLocation(objectOfInterest));
    double track = calculateTrack(objectOfInterest);

    double utility = confidence + visibility + track;

    return utility;
}

private double calculateVisibility(GridPoint objectLocation) {
    if (!isInCameraView(objectLocation)) {
        return 0.0;
    }
    double distance = grid.getDistance(objectLocation, grid.getLocation(this));
    double maxDistance = this.fovRadio;
    return (distance <= maxDistance) ? 1.0 - (distance / maxDistance) : 0.0;
}

private double calculateConfidence(GridPoint objectLocation) {
    double visibility = calculateVisibility(objectLocation);
    double confidence = Math.sqrt(visibility);
    return confidence;
}

private double calculateTrack(ObjectInterest obj) {
    return (trackedObjects.containsKey(obj)) ? 1.0 : 0.0;
}
```

For p (the sum of all payments paid in that tick) and r (the sum of all payments received), we calculate them during the auction:

```java
// Finalize handover by transferring tracking responsibility
public void finalizeHandover(Camera winner, ObjectInterest objectToHandover, double pricePaid) {
    System.out.println("Camera " + id + " hands over object to Camera " + winner.id + " at price " + pricePaid);

    Network<Object> net = getNetwork(netName:"camerasNetwork");
    // Remove the object from current camera's tracking list and network
    trackedObjects.remove(objectToHandover);
    net.removeEdge(net.getEdge(this, objectToHandover));
    objectToHandover.setTracking(track:false);
    this.r += pricePaid;

    // Add the object to the winner's tracking list and network with its bid value
    winner.trackedObjects.put(objectToHandover, pricePaid);
    objectToHandover.setTracking(track:true);
    winner.p += pricePaid;

    // utility
    double utility = calculateUtility(objectToHandover);
    net.addEdge(winner, objectToHandover, utility);
}
```

Simulation

The parameters and simulation results can be seen in the source code files. Following are the icons indicating the characters.



refers to the cameras.

refers to the objects interested.

Setting parameters

The parameters can be set in the simulation runtime GUI. Here is the meaning of the parameters.

parameter	meaning
auctionStrategy	the strategy used in auction, 0 for Vickrey, 1 for Dutch
isRandom	whether the distribution of the cameras is random

Following are the fixed parameters for every simulation.

parameter	value
environment size	(height, width) = (64, 64)
random seed	389419806
stop tick	1000

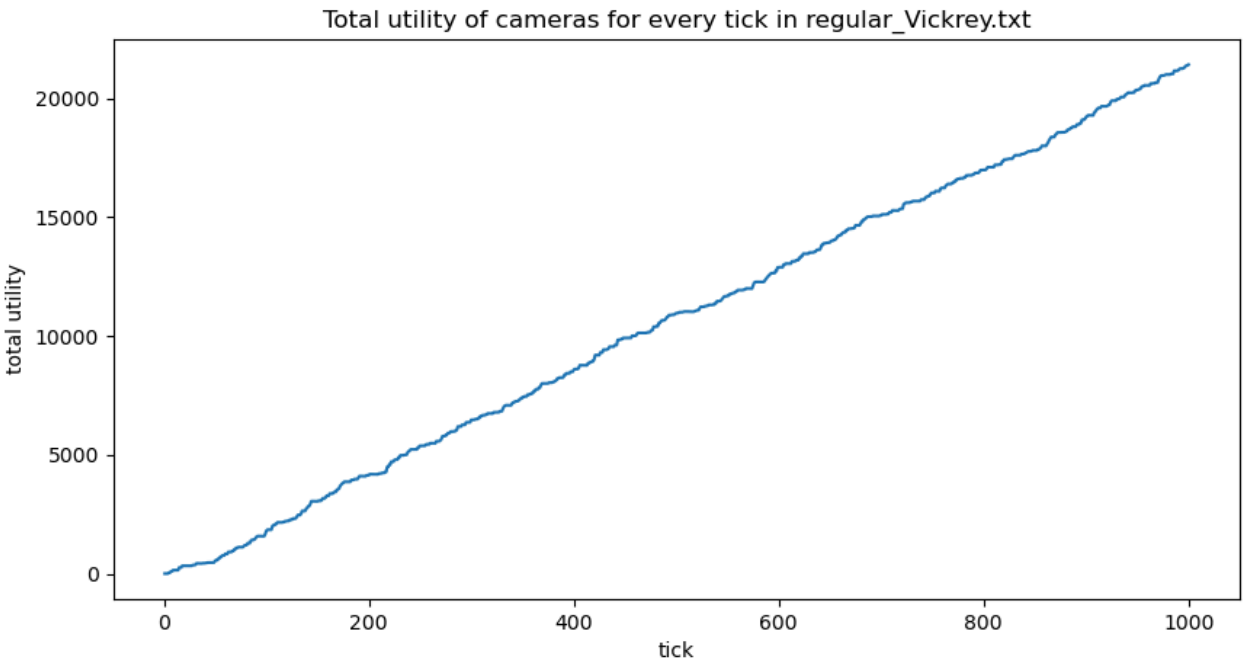
Structure of the source code files

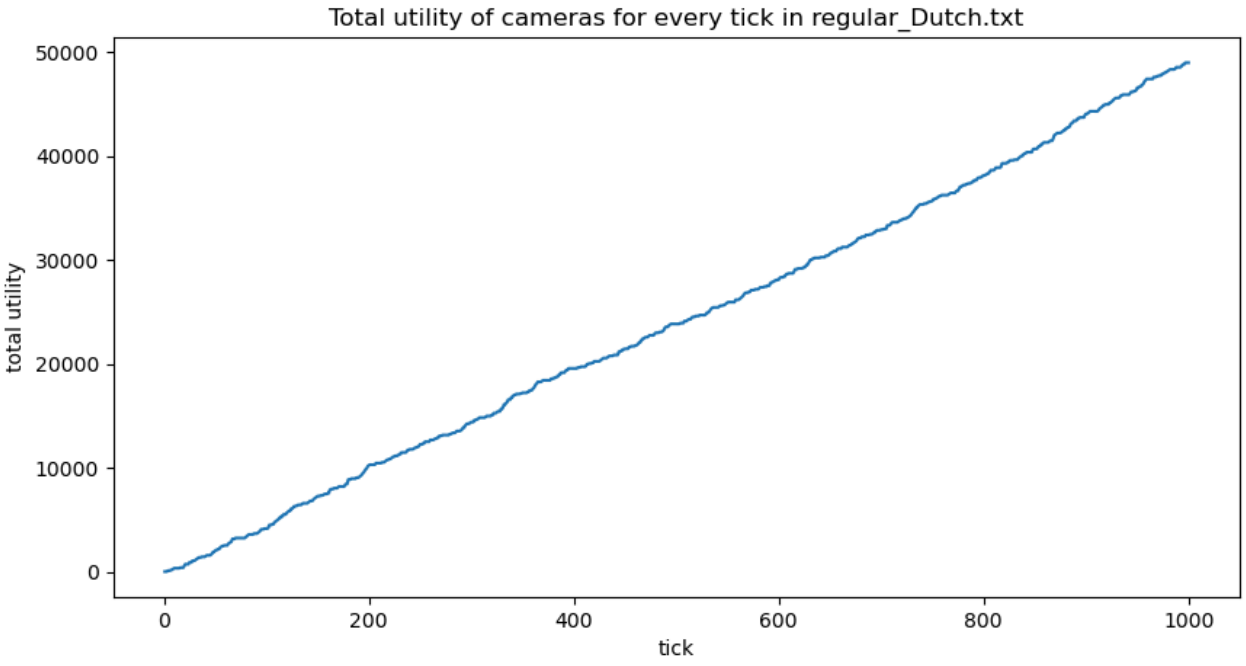
The source code files are under the folder `src_code`. And its structure and descriptions are as below.

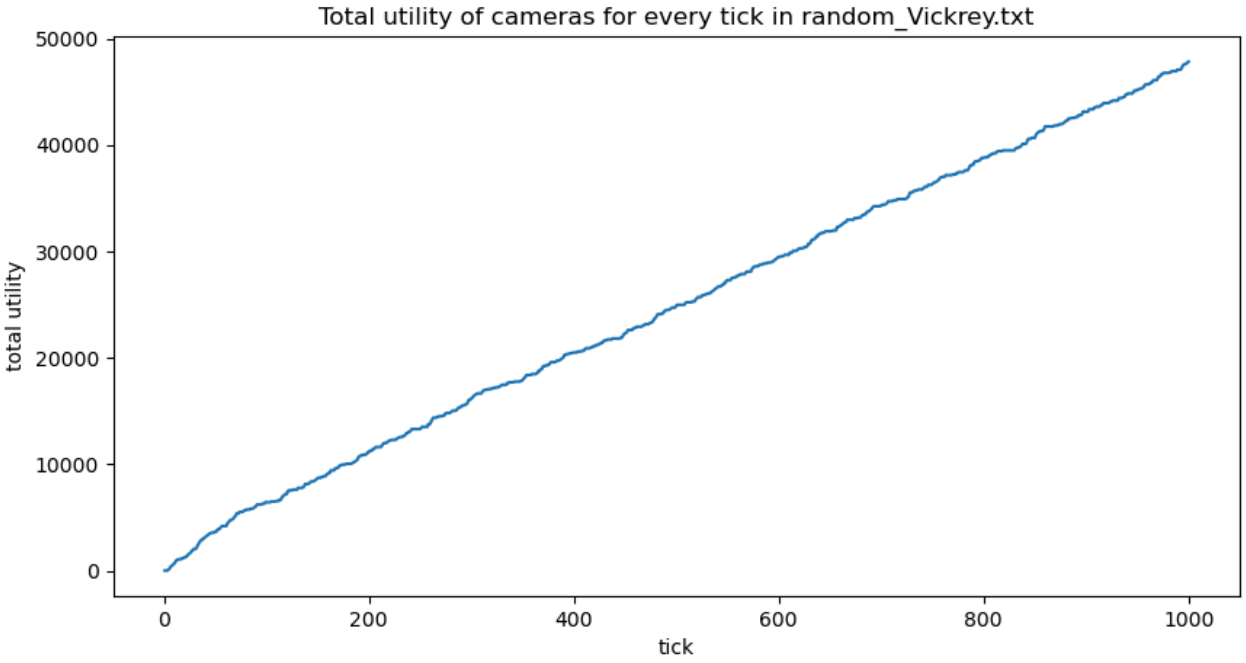
```
src_code
    ├─records           // simulation records
    ├─src               // source code
    │  └─multicameraTracking
    └─multicameraTracking.rs      // styles and parameters
```

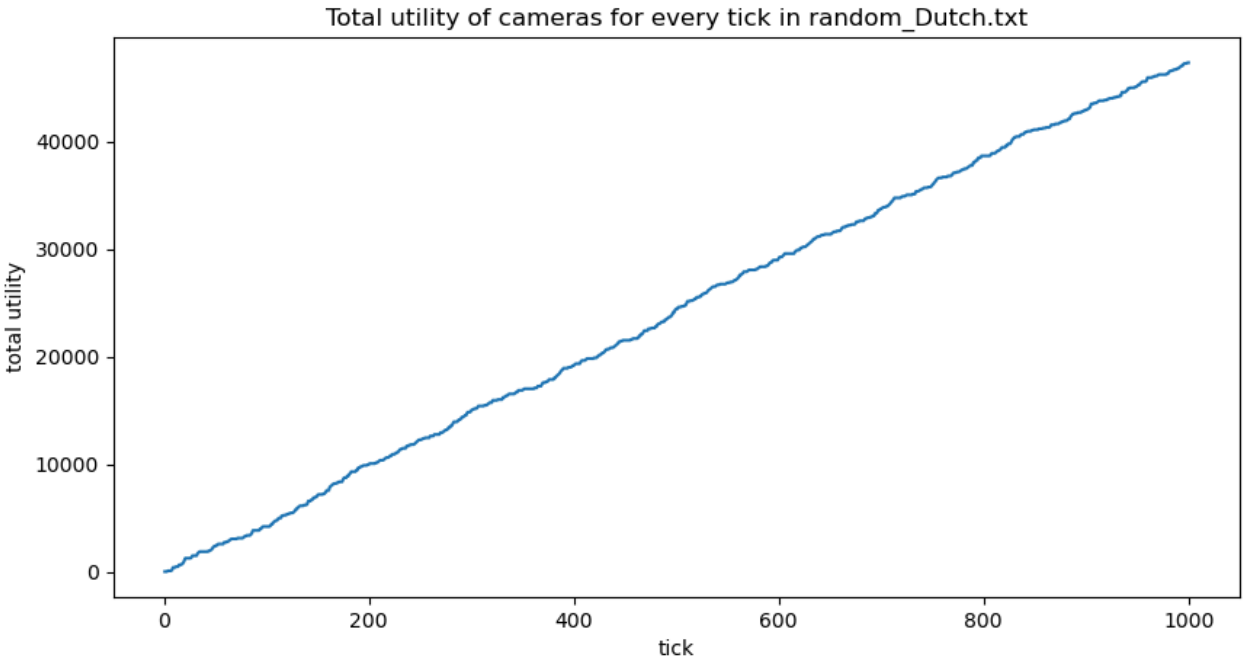
Results

The running records can be seen undet the `records` folder. Here we have plot the figures as below:




Total utility of cameras for every tick in regular_Dutch.txt
total utility
tick
0
10000
20000
30000
40000
50000
0
200
400
600
800
1000


Total utility of cameras for every tick in random_Vickrey.txt
total utility
tick
0
10000
20000
30000
40000
50000
0
200
400
600
800
1000



PROBLEMS

Eclipse is hard to use. But there is no support for IDEA or vscode due to the integration of Eclipse according to the developers' response.