UNIVERSITÀ DEGLI STUDI DI MILANO
FACULTY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Master in
Computer Science

SIMULATION
FINAL REPORT ON AGENT-BASED SIMULATION FOR STIROAPP

Teacher: Prof. Alberto Ceselli

Final report written by:
Samuele Simone
Matr. Nr. 11910A

Academic year 2022-2023

Contents

Chapter 1

Introduction

The idea behind this project was to try to validate a startup idea by making use of simulation models. Specifically by leveraging an agent-based simulation. In the report therefore we will go over what StiroApp is, how it works, the models that have been used as well as the technical implementation. Through the use of KPIs we will study the performance indicators of the model. After that there will be a chapter dedicated to experiments and finally the conclusions that can be drawn from it.

1.1 StiroApp - The story behind

StiroApp was born from the desire to bring to the market of applications, and more generally of services, a system that can connect two categories of people: Workers and Buyers. In more detail, the service seeks to make the process of ironing/washing one's clothing faster, more streamlined and more economical. I started by analyzing people's desire/problems by going to create user-stories necessary to make the product as user-centric as possible. As an example I reported one user story for a better understanding of the problem.

1.1.1 User story - Max

- **Context**: Max is a 27-year-old boy. He has been living alone for about a year. He moved to Milan for work reasons. He's a video game enthusiast.
- **Problems:**
  - Because of his busy life, He does not have time to iron/wash his clothes
  - It is a time-consuming, tedious and even difficult activity

1.2 Understanding the problem - More deeply

After a series of dutiful analyses and interviews at the front to examine the problem as best as possible here is what can be extracted in a more abstract way representing the basic principles for which is Stirapp should be born.

- Only a few millennials make use of laundries
- Daily laundry/ironing cannot be delegated to laundries
- This type of daily activity (ironing/washing) is generally carried out by generation X (1965 - 1980) as opposed to generation Y (1980 - 1994) and Z (1995 - 2010)
- Delivery/collection times of clothes are high

1.3 Proposed solution

The approach taken to solve these problems was therefore to offer an application for people in such a way as to connect the two categories into which we can classify users:

- Workers: Those who intend to use the application for the purpose of earning money by working for Buyers, thus ironing or washing their clothes.
- Buyers: Those who intend to use the app as a service where they can get their laundry cleaned and ironed.

1.3.1 Starting features

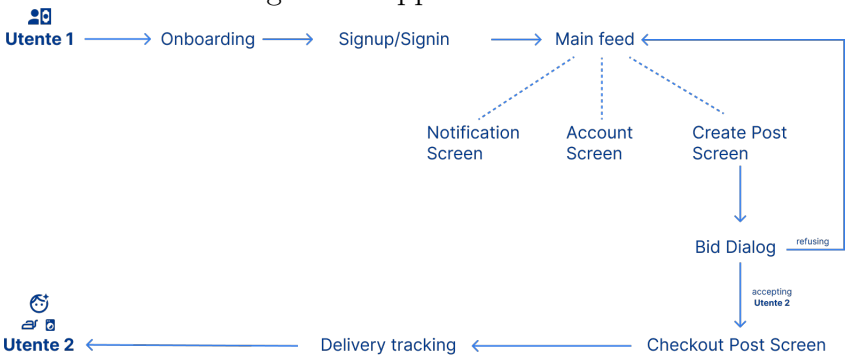
The app can be divided into two major features:

- Create posts for your own clothes by specifying the type of service you want (example: washing, ironing, both) and your needs.
- Wash, iron other users' clothes in such a way as to earn money.

1.3.2 App Flow

Below I show how the app should work. As an example during the agent-based simulation, some changes were made so as to focus more on the important parts of the app such as post creation or the worker system.

Figure 1: App flow Schema



- **Utente 1 (Buyers)** enter in the app after the onboarding in which he can understand the main features of the app
- **In-app onboarding** is the process of teaching users how to use an app to achieve their goals
- **Signup-Signin**. The user must be logged or registered into the system before using the app
- The user will be redirect into the Main Screen of the app. Then with a bottom navigation bar he can choose in which page enter such us Notification screen, Account screen and the Create Post screen.
- **Bid system** is a hypothetical part of the app in development
- **Checkout post screen** is the confirm page after the accepted price
- **Delivery tracking** is an important page for the user in order to check where is his order
- **Utente 2** he is the Worker that is allowed to do the laundry or offer other type of services specified in the app during the create post screen

1.3.3 UX and UI for StiroApp

Before we get our hands on the code, it is important to study the user experience (UX) and user interface (UI) to finalize the creation of a user-friendly application. So here I report some images of the work done:

Figure 2: Representation of the main features of the application through UX

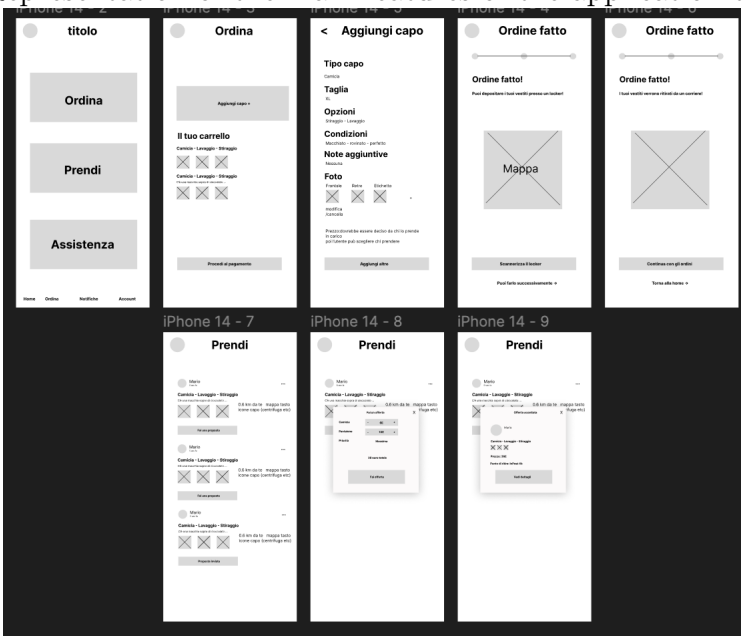


Figure 3: An example of Card UI for the post creation

1.3.4 Mockup

To give a real representation of StirApp to the reader I created a mockup. According to Decode Agency [1], an app mockup is a detailed representation of your app design. It contains all the final UI elements such as typography, copy, colors, and visuals like icons and photos. Sample content will also be used, but dummy text is also acceptable.

Figure 4: Main Feed Mockup

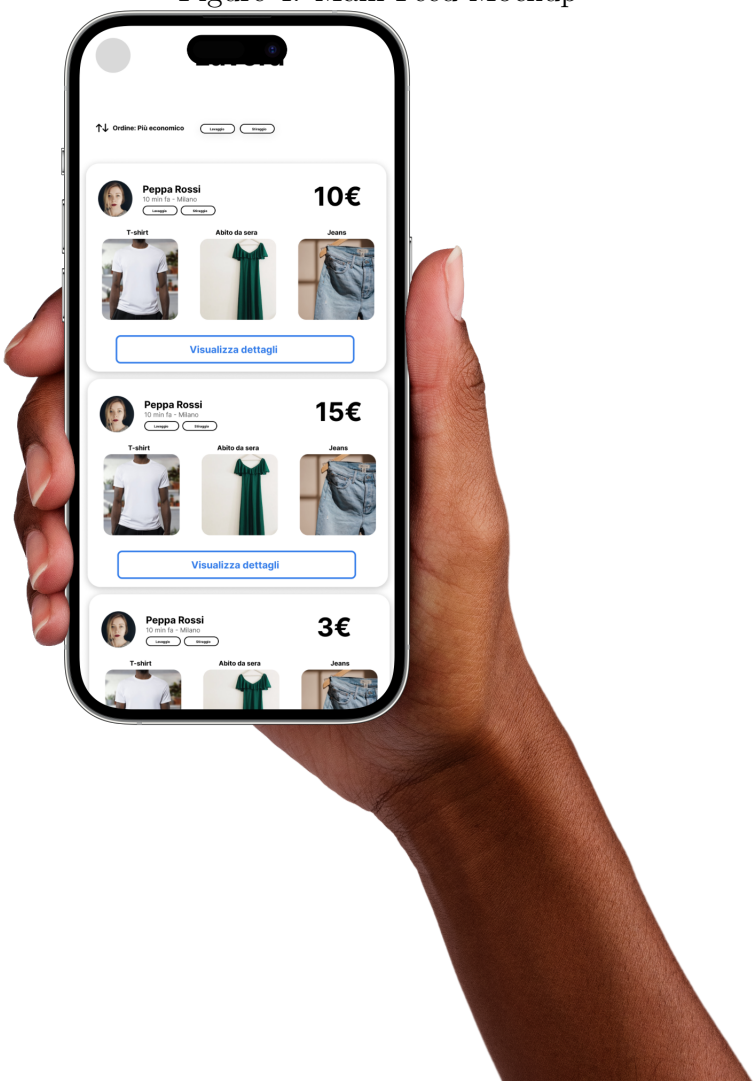


Figure 5: Detail Item Screen Mockup

1.4 A look into the market

By doing a market analysis I obtained those main pillars:

- Buyers find this activity (washing/draining) time-consuming
- Workers find this activity (washing/ironing) necessary and have the skills to do it
- Buyers prefer to find someone who can take care their clothes
- Workers make a profit for fulfilling these tasks

1.5 Business Model

To conclude this introductory part and to make sure that the project can have an economic source of revenue we need to proceed in writing a business model. My business model is based on the number of valid transactions within the application. A valid transaction is when the work is completed by both parties.

- Fixed cost + variable percentage based on post cost paid by buyers.

- Applicable for every valid transaction
- In the future, cleaning products (internal e-shop)
- Advertising in some list item (non-invasive)

Currently within the simulation only point 1 and 2 has been carried out. The remaining ones, on the other hand, are to be applied in such a way as to try to increase revenue more and have a higher margin than the costs incurred. Costs are defined as the following:

- Infrastructure costs of the service
- Costs of team management and administration.
- Costs on privacy and damage of garments

Chapter 2

Model

In order to simulate the traffic over the app and to estimate revenues, costs and other KPI that are described in the Chapter 4, I used the agent-based modeling. As well described from the Columbia University website [2], Agent-based models are computer simulations used to study the interactions between people, things, places, and time. They are stochastic models built from the bottom up meaning individual agents (often people in epidemiology) are assigned certain attributes.The agents are programmed to behave and interact with other agents and the environment in certain ways.

The agents that are involved in the model are:

- **Worker_iron**: Represents the worker within the application
- **Buyer**: Represents the buyer, i.e., the one who creates the posts with their clothing
- **Rider**: He is in charge of picking up the goods and delivering them to the respective worker/buyer.
- **Scheduler**: He/she is in charge of handling incoming posts and directing them

We will see during this Chapter how I modeled these agents for StiroApp.

2.1 Worker_iron agent

Below I will go on to describe the logic of the Worker_iron agent. The technical details will be discussed in Chapter 3. Basically the worker as soon as it is generated waits for some order to be assigned to it, in a **Waiting state**. Through a branch the worker by condition wonders whether it has received the order or not.

In the first case then he can start working by entering the **Working state** and after a Uniform Discrete random variable $\mathcal{U}(a, b)$ where $a = 2, b = 60$ minutes, he enter in a **ReadyForDelivery state** Then, with a 10 minutes Timeout transition enter in a branch and through a Bernoulli random variable $\mathcal{B}(p)$ where $p = 0.7$ he decides whether to finish his task or re-enter the **Waiting state**.In the second case, on the other hand, he enters the **Thinking state** and then after a small timeout point to the same transition branch where there is the Bernoulli random variable.

2.2 Buyer agent

The Buyer as soon as it is generated is in a **Waiting state**. In fact here, through a transition to the branch you want to check if that specific agent is new or already created and therefore is waiting for his order. If he is new then he can create a post with his clothing by entering the **CreatePost state**. After that through a Timeout of 10 min he enters the **Waiting state** and also through a Bernoulli random variable $\mathcal{B}(p)$ where $p = 0.7$ he decides whether to finish his task or re-enter the **Waiting state**. If,instead, he has already a pending order and it's notified as delivered he can again through the same branch if finish or re-enter.

2.3 Rider agent

The rider plays a key role within my simulation. In fact it involves a slightly more complex logic in that its task depends on the type of order received. Initially it is in a **Waiting state**. As soon as it receives an order it goes into the **AssignedOrder state**. Then it has to figure out through the properties of the received order whether it is a pickup or a delivery. In fact the rider first has to go to the buyer and receive the clothes for which it wants to perform a service and from there, the rider has to deliver it at the chosen worker. Once this is done through a Bernoulli random variable $\mathcal{B}(p)$ where $p = 0.7$ decides whether to continue working enter the Waiting state again or stop working. If it continues working it may happen to receive the delivery order and then go to a worker who is in the ReadyForDelivery state, pick up the clothes, and return them to the buyer who owns those clothes.

2.4 Scheduler agent

This agent, on the other hand, acts as a conduit between agents by handling the exchange of messages (specifically, we will see that we will be dealing with the dispatch of an object of type Post). It presents two states, a **Wait state** and

a **Dispatch state**. It simply presents two transitions and allow you to manage the event queue and based on the type of post received sort the dispatches to the correct recipient.

Chapter 3

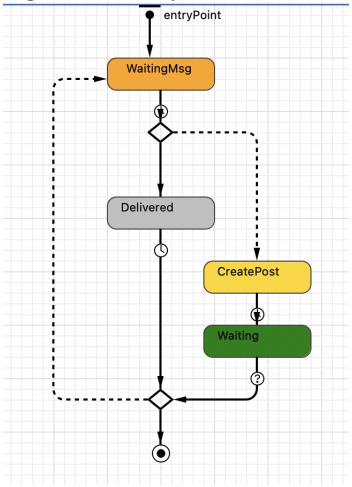
Implementation

In practice to implement the models described above, I have used AnyLogic software. Specifically in this chapter I will show the various flowcharts and the more specific transitions that require further study.

3.1 Buyer agent implementation

Let's start with the implementation of the Buyer. The most important part to

Figure 6: Buyer statechart



analyze is within the CreatePost state. In fact, there is some Java code here that is responsible for the operation of the orders.

```
Post p = new Post();
p.buyer = this;
p.whoIs = 0;
```

```
p.numOfCloths = uniform_discr(1, 10);
p.postPrice = Math.round((uniform(0.5, 20) * p.numOfCloths) *
    100.00) / 100.00;
post = p;
send(p, main.scheduler);
```

As you can see from the code I created a Post class where inside there are a number of attributes.

- **buyer**: who is the buyer and so in this case I pass him the **this** pointer, so himself being in the Buyer agent
- **whoIs**: this is an integer attribute that allows me to figure out whether the order was placed by the Buyer (0) or the Worker (1)
- **numOfCloths**: for the cloth numbers I relied on a Uniform Discrete random variable $\mathcal{U}$ of parameter $a = 1, b = 10$.
- **postPrice**: same for post price by estimating on parameter b = 20
- **status**: it's useful for understanding if the order is completed or is currently under working
- **rider**: assign a rider for that specific post
- **worker**: assign a worker for that specific post

This operation, on the other hand, must be commented out send(p, main.scheduler), since it is critical for handling the exchange of messages (and thus Posts). Then the conditional transition are two: From the branch to delivered if the post (variable inside the Buyer agent) is not null and the status = 1.

```
post != null && post.status ==  1
```

Same store from Waiting to the branch if status = 1.

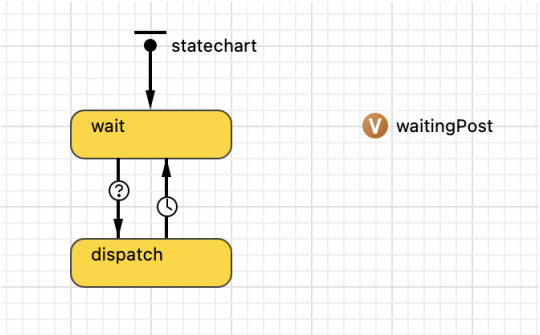
3.2 Scheduler agent implementation

Another important agent is the Scheduler agent as we saw in the previous Chapter. However let's have a look inside the statechart. As you can see there are two transition in which one is conditional. Here the code for the trigger condition:

```
waitingPost.size() > 0
```

so if the size of the queue isn't empty. Then if the condition is triggered there is this action to be executed:

Figure 7: Scheduler statechart



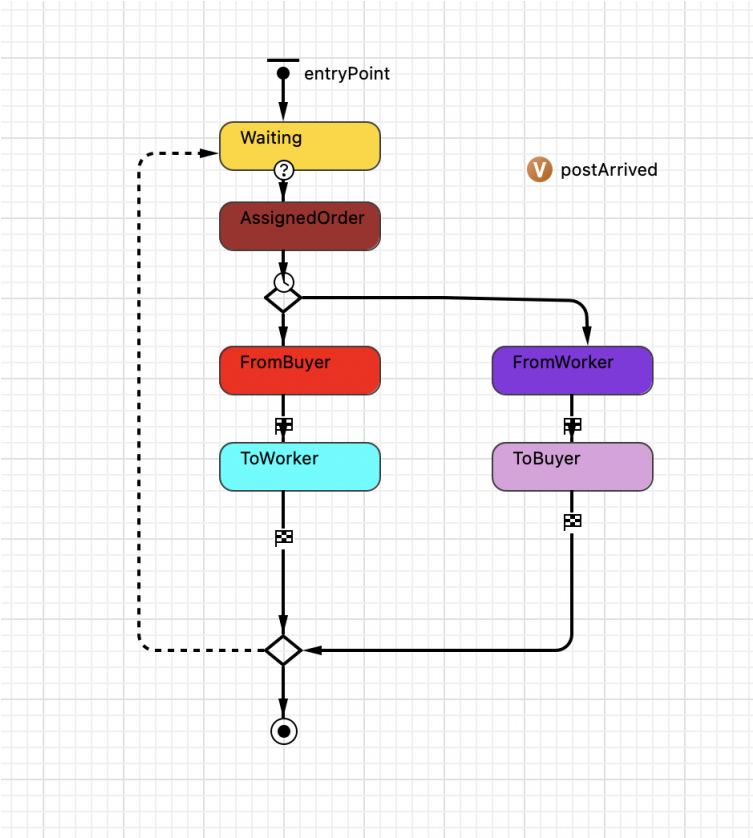

```
Post currPost = waitingPost.get(0);
waitingPost.remove(0);
if(currPost.whoIs == 0){
  Workers_iron w = main.workers_irons.findFirst(wi -> wi.inState(
   Workers_iron.Waiting));
  currPost.worker = w;
  Rider r = main.riders.findFirst(ri -> ri.inState(Rider.Waiting))
   ;
  if(r != null){
    currPost.rider = r;
    send(currPost,r);
  }

}else{
  Rider r = main.riders.findFirst(ri -> ri.inState(Rider.Waiting))
   ;
  if(r != null){
    currPost.rider = r;
    send(currPost,r);
  }

}
```

So I'll extract from the queue the first order and I remove it. Then I check if the order is from the Buyer, so it means that we need to search and find the first worker that is in the Waiting state, assign it to the Post attribute worker and the find the first rider that is in a Waiting state. If so, we assign the rider to the currentPost and then send currentPost to the rider. Instead, if the order is produced by the Worker, we just find a free rider that can take the processed order and send back to the Buyer.

3.3 Rider agent implementation

In order to make the simulation work I need this agent. Here the statechart: As we

Figure 8: Rider statechart



discussed previously the important thing here is to understand how the rider will move and how to trigger the conditions. Indeed we have two different flow: one is from **FromBuyer state** to **ToWorker state** state and here the rider will deliver the laundry from the Buyer to the Worker that need now to iron/wash his cloths. On the other hand we have the flow from **FromWorker state** to **ToBuyer state** that means that the Worker has finished his job and now the rider can catch the laundry and deliver to the owner (Buyer). When the rider is the **AssignedOrder state** through the condition

```
postArrived.whoIs == 0
```

we execute this action

```
moveTo(postArrived.buyer);
```

Note that each state block has his own color. That's useful to recognize in which state the rider enter during the simulation by change the shapeColor using this code:

```
shapeBody.setFillColor(red);
```

When the rider is arrived for example to the Buyer , he will pass in the other state like ToWorker state only by the transition that is trigger by agentArrival. (It is represented as a checkered flag). When this is true the action is the follow:

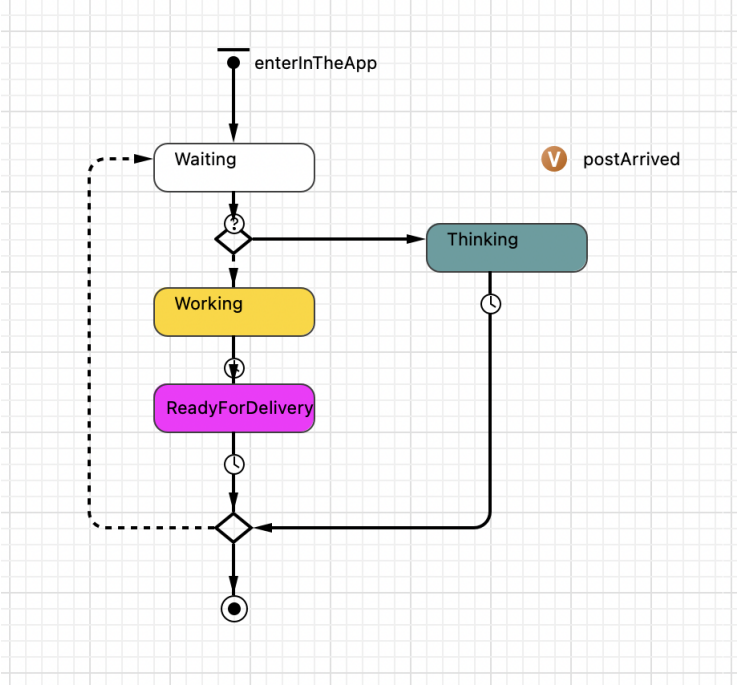
```
moveTo(postArrived.worker);
```

Same story but in the opposite way for the other branch. After the deliver the rider will set the postArrived to null and with the Bernoulli random variable as discussed in the Model Chapter we can continue the flow.

3.4 Worker agent implementation

Then to complete the flow, we need the Worker agent that has the duty of complete the service that the Buyer asked. Here the statechart: The only thing to discuss

Figure 9: Worker_iron statechart



here is the conditional transition that allow us to enter inside the branch. Indeed:

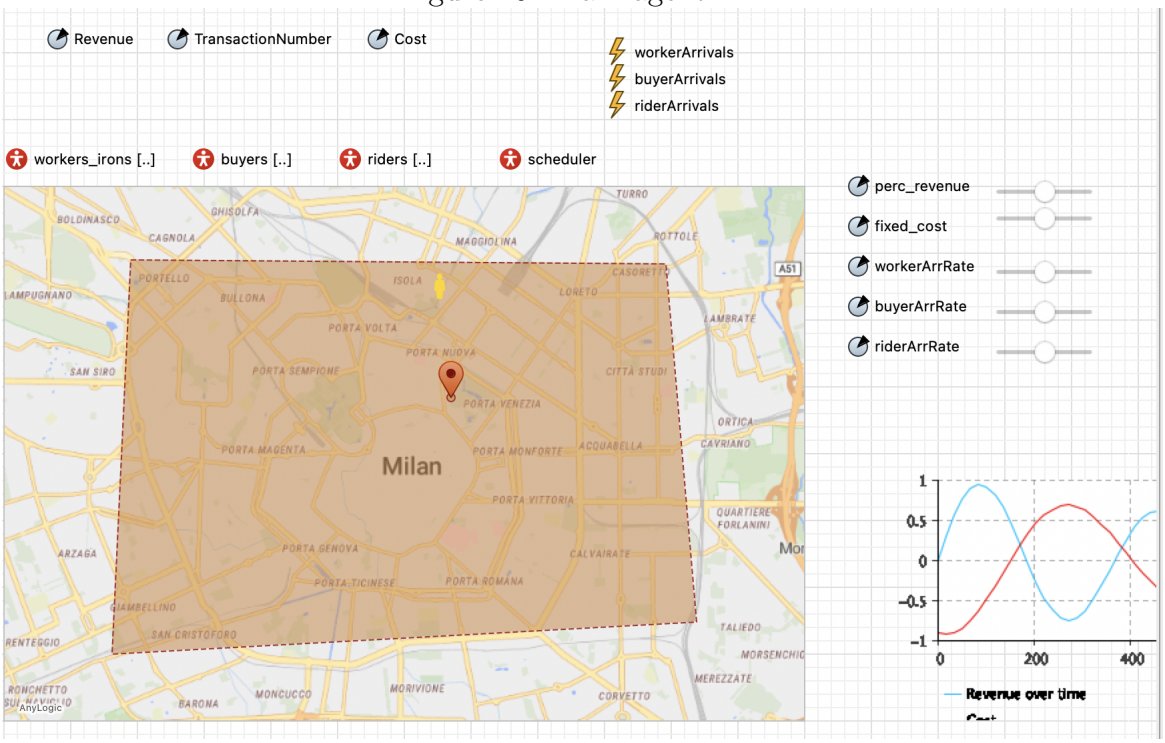
```
postArrived != null
```

then the Worker can enter in the Working states otherwise he will enter in the Thinking state.

3.5 Main agent implementation

Finally let's talk about the special agent of the simulation that is called **Main agent**. It represents the "dashboard" of the simulation. As you can see the

Figure 10: Main agent



first thing that appear is the GIS map. It is centered in Milan, in which the simulation will run. With the help of the gis region I established a wide area within worker,buyer,rider can spawn. In the top-right corner there are 3 different arrivals, one for each agent, excluding the Scheduler. With this code they can spawn in the GIS Region, just change the agent that we want to spawn:

```
Rider r = add_riders();
Point p = gisRegion.randomPointInside();
Position q = new Position();
q.setLocation(p);
r.setPosition(q);
```

Then let's consider the trigger type. They are all Rate and the amount is customizable with the sliders on the right side. Just riders are Rate/per day instead the other are Rate/per hours. To conclude all are population of agents and just the scheduler is a single agent.

Chapter 4

KPI

In the top of the Figure 10 there are 3 different parameters:

- Revenue
- TransactionNumber
- Cost

These are my KPI. The goal of this simulation is to maximize the Revenue KPI and compare it with the Cost KPI that I want to minimize. Indeed all the startup/industries wants to increase their productivity and their earnings by comparing it to the cost of all the infrastructure. In this case the Costs are simplified and they are estimated with 64 euros/per day , so the cost of each rider. The Revenue is counted when a valid transaction occurs. To be valid the flow Buyer → Worker and Worker → Buyer must be completed. Revenue and TransactionNumber are updated in the **ReadyForDelivery state** with the following code:

```
main.TransactionNumber++;
main.Revenue += postArrived.postPrice * main.perc_revenue + main.
    fixed_cost;
```

So,as described in the subsection 1.5, I will use two parameters like perc_revenue and fixed_cost in order to increase the earnings. So by playing with the sliders we obtain several different scenarios that are fully shown in the Chapter 5.

Chapter 5

Experiments

In this Chapter we will explore different scenarios obtained by changing parameters, number of population agents and we will try to maximize the Revenue as the KPI of the system.

5.0.1 Original setup

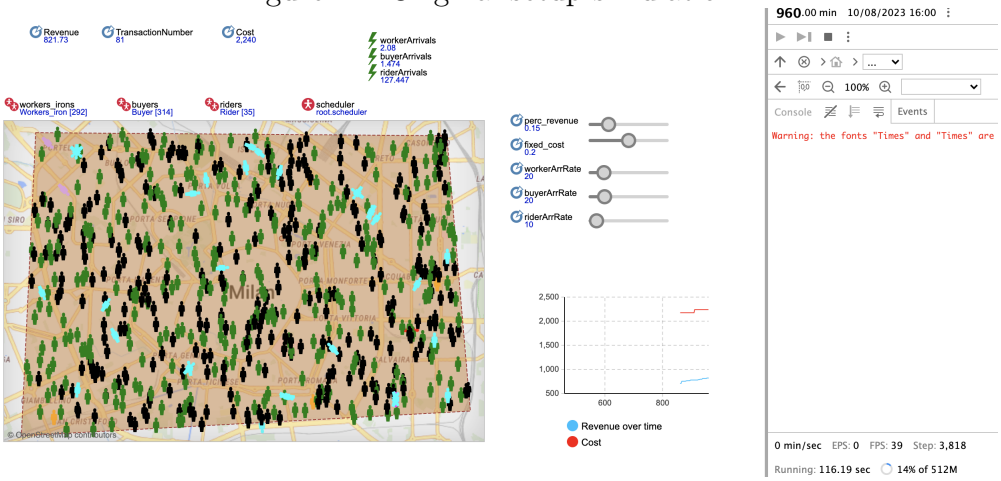
From the simulation panel I setup the simulation duration to 960 minutes that correspond to 16 hours. Then the other parameters:

- perc_revenue : 0.15%
- fixed_cost :0.20€
- workerArrRate: 20 per hour
- buyerArrRate: 20 per hour
- riderArrRate: 10 per day

Here the results: As you can see from the top KPI or form the chart in the bottom-right side the Revenue are below the Cost. So there will be a lost over the first day. So we need to change some parameters and check if the situation will be better or not. For doing that I applied the What-If scenario different time as reported below.

5.0.2 What-If scenario 1:

Figure 11: Original setup simulation



Chapter 6

Conclusion

ci

Bibliography

[1] Decode Agency. What is an app mockup? https://decode.agency/article/app-mockup/#:~:text=with%20proper%20wireframing-,What%20is%20an%20app%20mockup%3F,dummy%20text%20is%20also%20acceptable., 2022.

[2] Columbia University. Agent-based modeling. https://www.publichealth.columbia.edu/research/population-health-methods/agent-based-modeling#:~:text=Agent%2Dbased%20models%20are%20computer,epidemiology)%20are%20assigned%20certain%20attributes., 2023.