3D Traffic Simulation:

Self-Driving Vehicles at Intersections

(Zhijing Jin - 2017 Fall)

Introduction

Self-driving cars have received much attention in recent decades. A number of studies into intersection management of automatic cars have been conducted. For example, Dresner (2004) proposes a reservation-based intersection control mechanism which requires a central controller that collects requests from all vehicles and returns signals of whether they can proceed or need to wait. Other research depends on vehicle-to-vehicle and vehicle-to-infrastructure communication devices (Lee, 2012; Zohdy, 2012). However, having a central controller or standardized communication devices on individual vehicles may be too unrealistic in many actual situations, and it is hard to collect information on vehicles' precise intentions at the intersections. As an alternative, a distributed system may fit in more situations. (1)Giving each independent agent the decision making ability is a safer and (2)computationally easier way, as agents only need to keep track of the objects on trajectories. (3)It also avoids the bandwidth problems of having a central controller receiving signals from all vehicles. (4)What's more, as the training is based on images from camera sensors, we can take into account pedestrians or special obstacles on the road. (5)We have a fifth advantage of more comprehensive data. Instead of using data from a certain number of experimental self-driving cars, we will mount data from static sensors in a city (for example,

on light poles overlooking intersections). The static sensors will tabulate 24/7 position information on all agents and then feed it into a visualization environment.

In this project, I will first establish a standard data format for traffic data, and then build a visualization environment for all kinds of traffic data at intersections. This simulator will benefit future needs for visualization, debugging and verification. It can also work with a data generator and become a game platform for reinforcement learning, taking advantage of the collision detection ability of Unity, an open platform game engine that is widely used in research.

Summary of Project

In this project, I visualizes various car movements at intersections. The visualization can be shown in Unity, an open platform game engine that is widely used in research. This work can be applied to illustration of raw numerical traffic data, visualization of model effects for the purpose of debugging, and demonstration of the final effects.

In addition, a server is established for the communication of different platforms. This server facilitates high-frequency updates between the programs that analyzes data and the Unity visualization.

An illustration video can be found here:

Demo - Car@Intersection - Unity https://www.youtube.com/watch?v=TViK7s2HJ-4

Methodology

The main methodology includes python programming and establishment of a 3D environment with multiple agents in Unity. The Python codes need to read in certain parameters of the settings, such as the size of the intersection, the number of lanes per way, the width of the lane, the trajectories of vehicles and their speed. It then needs to generate proper data types for each information that can be interpreted in Unity environment.

Setting up the Unity 3D environment requires an understanding of game objects, scenes and scripts. The main idea is to create a scene, instantiate appropriate game objects, and write scripts for their behaviors.

Implementation Details

Part 1: Setting up a server with Python Flask

To enable the frequent communication between Python-generated data and Unity game engine, a server needs to set up to receive POST and GET requests.

With Flask, a web framework in Python, a local server is established at localhost:5000. The index page keeps all the real time updates of data.

To communicate with the server, python programs can post on localhost:5000 through the requests library, while Unity scripts just need to get website data through System.Net namespace. This update can be more frequent than 24Hz.

Part 2: Constructing the road infrastructure

The infrastructure encompasses an intersection area and four roads with certain directions, lane width and rules.

There are two purposes for designing the infrastructure: demonstration needs and traffic rule implementation (making the lane an object can prevent the car from getting out of the lane).

On the python side, the program takes in data for the global parameters, such as the 3D location of the center, the width of roads, the number of lanes in each way and so on. Mathematical calculation is included in the program and it will output the position, length, width and height for each road segment.

On the Unity side, once it receives data of a list of road segments. It automatically create them as a list of cube objects.

Part 3: Designing the Car Movement

The car movement is mainly designed in python. There are mathematically calculated trajectories of linear and circular routes. And there is also a template for arbitrary routes.

For a user-defined trajectory, the python files can read in an image and generate the route. Then according to the route and a certain speed, it will generate car position updates every certain interval.

This car position update is posted on localhost:5000 so that Unity can read in this information at a much quicker speed than from file I/O.

Documentation

A video on implementation details can be found here:

Demo - How it Works - Unity Car@Intersection

https://www.youtube.com/watch?v=anKKEzA4l4A&t=23s

Part 1: Unity side

The overall structure of the Unity project is as follows:

./Assets

- _Scenes
- Resources
  - DataForSettings
  - RoadData

- VehicleData
  - Materials
- Scripts
  - Cars
  - Json_InAndOut
  - Lines
- Road Pack
- Vehicle Pack

1. Visualization effects: Vehicles and Roads

1.1 Where are the prefabs of vehicles and roads?

Path:

FancyIntersection/Assets/RoadPack/Prefabs

FancyIntersection/Assets/Vehicle Pack/Prefabs

Dependency:

None

1.2 How to customize them?

For cars, you can use other assets of car models. After importing that asset, you just need to create objects of that type and attach scripts to them. Please refer to the script that is currently attached to the car objects for reference.

For road segments, you may need to change the look of the cube objects. Then you just need to create a new material and use it on the cube objects.

2. How does Unity receive data?

2.1 Where are the related files?

./Assets

- _Scenes
- Resources
    - DataForSettings
        - RoadData
            - lanes.json
        - VehicleData
    - Materials
- Scripts
    - Cars
        - **DesignedMovement.cs**
        - **PlayerCommands.cs**
        - **PlayerControl.cs**
        - **go_forward.cs**
        - **CarGenerator.cs**
        - **CarTemplate.cs**

- Json_InAndOut
    - **ReadJson.cs**
- Lines
    - **LaneGenerator.cs**
    - **LineTemplate.cs**
    - **LineDrawer.cs**
- Road Pack
- Vehicle Pack

Path:

Scripts/Cars/DesignedMovement.cs

Scripts/Cars/PlayerCommands.cs

Scripts/Cars/PlayerControl.cs

Scripts/Cars/go_forward.cs

Scripts/Cars/CarGenerator.cs

Scripts/Cars/CarTemplate.cs

Scripts/Lines/LaneGenerator.cs

Scripts/Lines/LineTemplate.cs

Scripts/Lines/LineDrawer.cs

Scripts/Json_InAndOut/ReadJson.cs

Dependency:

Resources/DataForSettings/RoadData/lanes.json

http://localhost:5000

2.2 How are roads constructed?

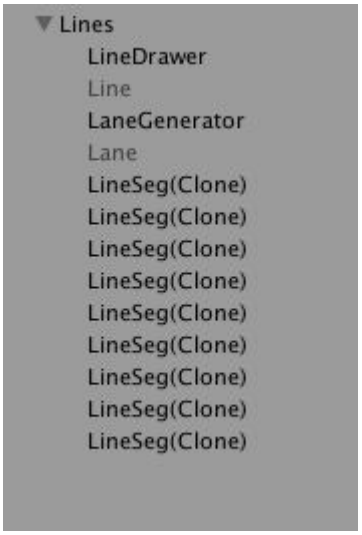
The roads are basic GameObjects in Unity. All road segments are children of a parent object “Lines”. They are instantiated according to position and scale information in lanes.json.

To read the lanes.json file, it calls a method to load json files from a path “DataForSettings/RoadData/lanes.json”.

The json reading method is written in “ReadJson.cs”. The contents in lanes.json is a dictionary with one element. The key of the element is “LineList”; the value is a list of dictionaries whose keys are “position”, “scale” and “traffic_direction”.

The script reads each line segment as an object of LineTemplate. The structure of this object is written in “Scripts/Lines/LineTemplate.cs”. Each LineTemplate has “position”, “scale” and “traffic_direction” as properties.

The object “LaneGenerator” is an empty object which only contains script “LaneGenerator.cs”. The methods in the script will be executed once Unity display starts.



2.3 How are Cars constructed?

In this stage, we manually create a car object and link a C# script to it. For example, one of the car moves according to “DesignedMovement.cs”. The other moves according to the user commands.

In “DesignedMovement.cs”, the car updates two properties according to localhost:5000. One of them is the position; the other is its rotation. The rotation is automatically the 3D direction of the change in position.

Part 2: Python side

The overall structure of the python files is as follows:

./	./Python/	./Python/DataGenerator/
- Python	- <dir> data # data for python files - <dir> DataGenerator # the module - <dir> documentation - <f> main.py - <f> launch_server.py	- __init__.py - global_assignments.py - carmove.py - infrastructure.py - ArbiTraj.py - server.py

1. Infrastructure

1.1 Where are the codes for infrastructure?

Module path: ./Python/DataGenerator/infrastructure.py

Dependency:

For calling the infrastructure.py: ./Python/main.py

The data will be stored to:

./unity/FancyIntersection/Assets/Resources/DataForSettings/RoadData/lanes.json

1.2 How to run them?

python ./Python/launch_server.py

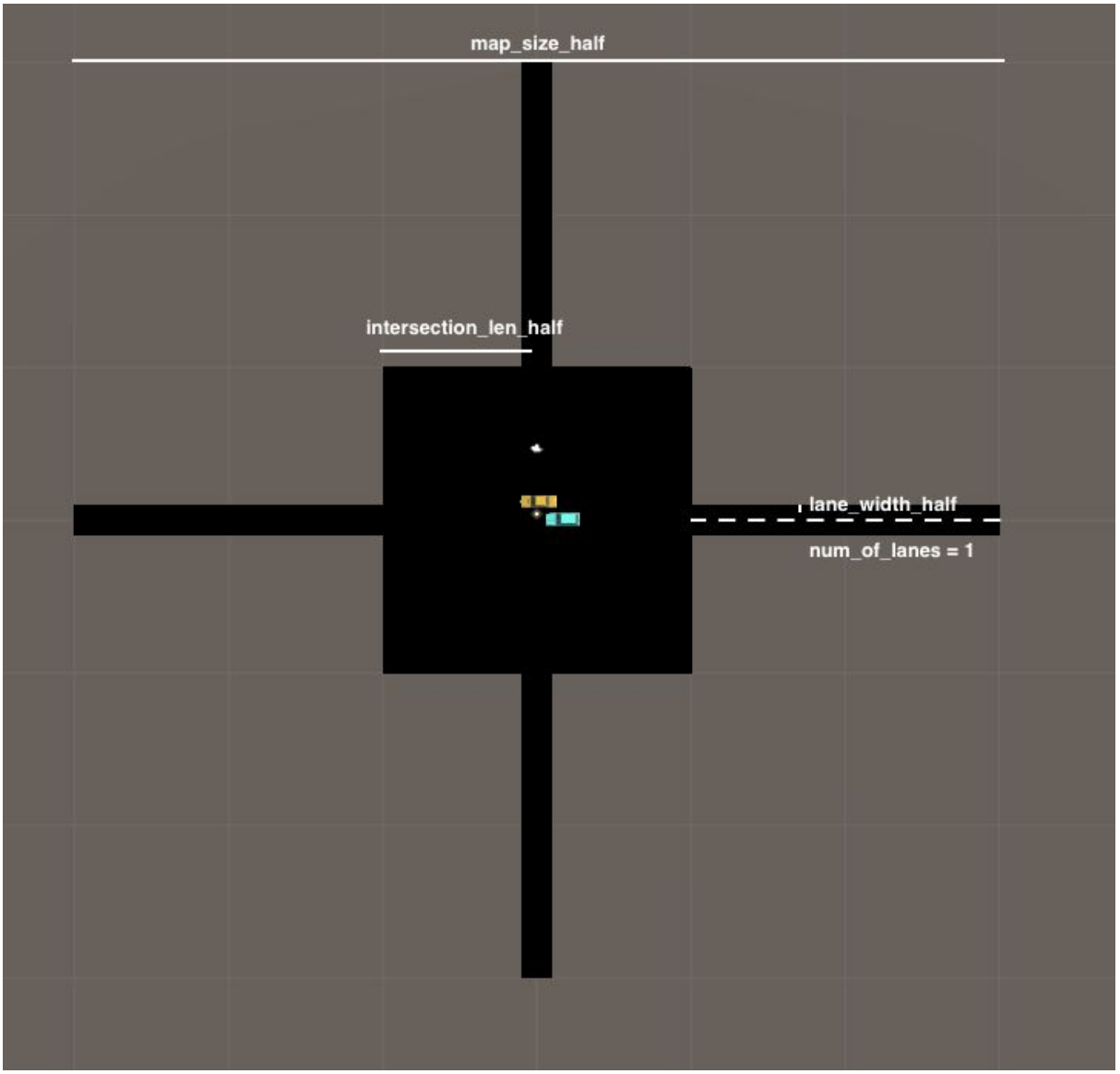
python ./Python/main.py

1.3 How to customize?

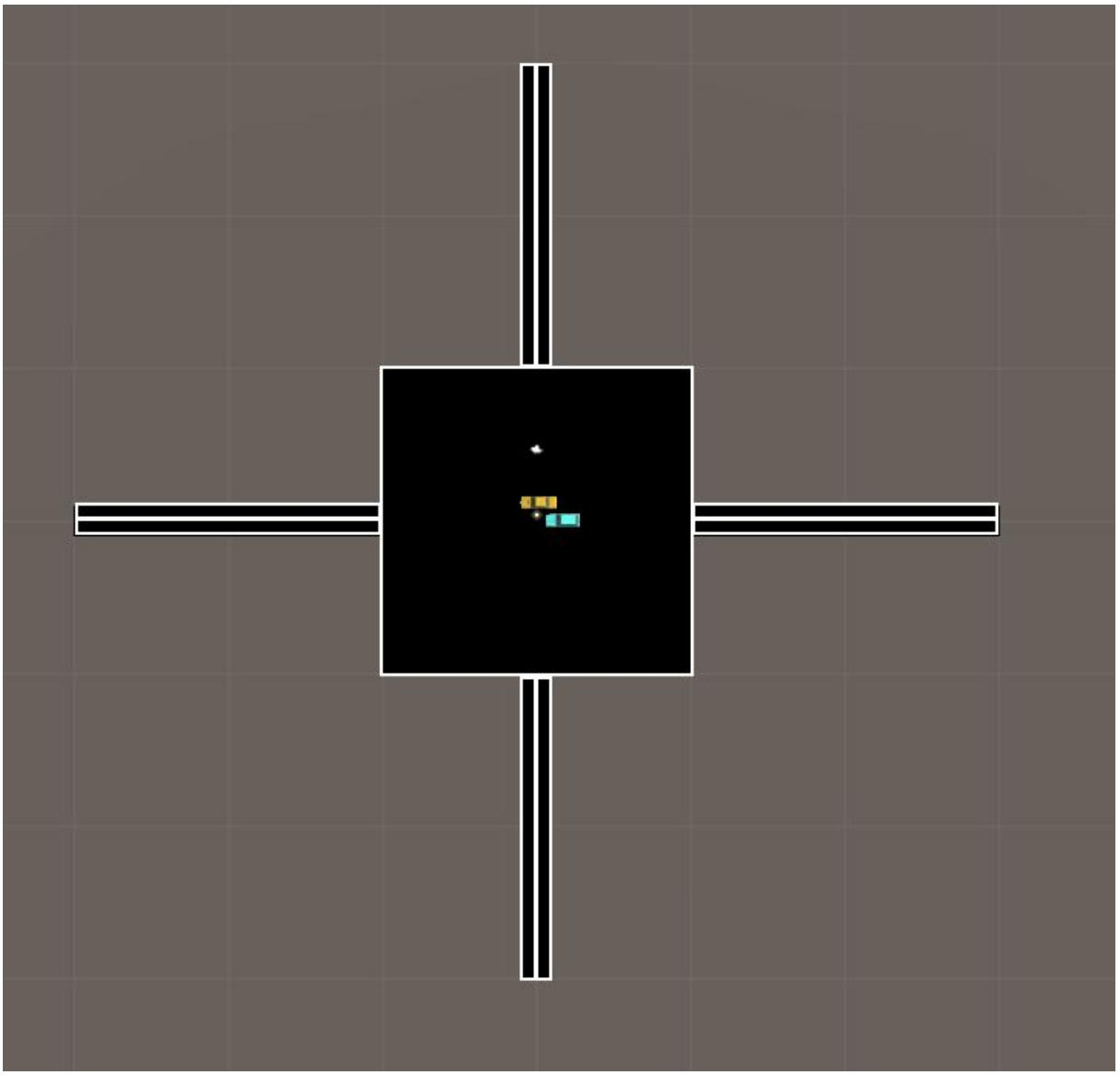
(a) Set parameters for the intersection

```
31      kwargs = {"map_size_half":30, "center":np.zeros(3),
32          "intersection_len_half":10, "lane_width_half":0.5,
33          "road_thickness":0.1, "num_of_lanes":1 }
```

In ./Python/main.py, some key arguments are specified before creating the intersection as is shown in the above picture. These arguments correspond to the following features of an intersection:



By specifying them, the python file will automatically calculate the position and scale of each line segment.



Unity will create 9 line segments accordingly.

(b) Location of the information for all line segments

In ./Python/main.py, you can specify the location where the information will be stored as a json file.

```
34      file_infra = '../unity/FancyIntersection/Assets/Resources/DataForSettings/RoadData/lanes.json'
35      set_infrastructure(file_infra, kwargs)
```

As default, it is stored in

'./unity/FancyIntersection/Assets/Resources/DataForSettings/RoadData/lanes.json',

and will be read by Unity later.

(c) Variables for each lane

Up till now, the variables for each lane includes “position”, “scale” and “traffic_direction”. This can be seen in the json file “lanes.json”:

```
{
    "position": {
        "x": 20.0,
        "y": 0.0,
        "z": 0.5
    },
    "scale": {
        "x": 20.0,
        "y": 0.0,
        "z": 1.0
    },
    "traffic_direction": 2
},
```

This can be customized in “./Python/DataGenerator/infrastructure.py”. Modifications can be made to the “class Lane”:

```
 8  class Lane:
 9      def __init__(self, position, scale, traffic_direction):
10          self.position = position
11          self.scale = scale
12          self.traffic_direction = traffic_direction
```

2. Trajectories of vehicles

2.1 Where are the codes for the trajectory?

Module path: ./Python/DataGenerator/carmove.py

Dependency:

./Python/DataGenerator/ArbiTraj.py

./Python/data/example_trajectory1.png

./Python/data/arbi_traj_2d.npy

Calling the module: ./Python/main.py

The data will be sent to: localhost:5000/

2.2 How to run them?

python ./Python/launch_server.py

python ./Python/main.py

2.3 How to customize?

(a) Set linear/circular trajectories:

For each Linear Trajectory, the two parameters needed is the start point and end point; for each Circular Trajectory, the location of the center and magnitude of radius are needed.

In “main.py”, A list of trajectories, along with global settings, can be passed to generate the car movements:

```
39
40      trajectories = [
41          carmove.Line_Trajectory(np.array([0.5,0,-kwargs['map_size_half']]),
42              np.array([0.5,0,-kwargs['intersection_len_half']])),
43          carmove.Circular_Trajectory(kwargs['center'], kwargs['intersection_len_half'],
44              -np.pi/2, np.pi/2),
45          carmove.Line_Trajectory(np.array([0.5,0,kwargs['intersection_len_half']]),
46              np.array([0.5,0,kwargs['map_size_half']]))
47          ]
48      # for an arbitrary traj:
49      # arbi_route = np.load("./data/arbi_traj_2d.npy")
50      # trajectories = [carmove.Arbitrary_Trajectory(arbi_route, \
51      #   np.array([0.5,0,-kwargs['map_size_half']]), np.array([kwargs['map_size_half'],0,0.5]))]
52      send_carmoves(trajectories, kwargs)
53
```

(b) Set arbitrary trajectories:

Files:

./Python/main.py

./Python/DataGenerator/carmove.py

./Python/DataGenerator/ArbiTraj.py

./Python/data/example_trajectory1.png

./Python/data/arbi_traj_2d.npy

Arbitrary trajectory class is written for cars moving in an irregular route. It is more applicable to real-world car routes.

To create an arbitrary trajectory, you need to have a one-pixel route in a monochrome image, such as example_trajectory1.png. Then the python file “ArbiTraj.py” will generate an array of 2D points in numpy format. This array of trajectory information will be stored in arbi_traj_2d.npy.

Then, carmove.py reads in the trajectory information from arbi_traj_2d.npy, scales and translates it into an array of 3D points in the Unity setting. The "Arbitrary_Trajectory" class in carmove.py takes in the array of 2D points and returns a transformed array of 3D points.

3. Frame Rate and Moving Speed

3.1 Where are the codes?

Module path: ./Python/DataGenerator/carmove.py

Calling the module: ./Python/main.py

The data will be sent to: localhost:5000/

3.2 How to run them?

python ./Python/launch_server.py

python ./Python/main.py

3.3 How to customize?

(a) Two variables: Speed and frame rate

There are two ways to change how the movement of vehicles are. In carmove.py, two variables are included in the function test_car_movementtest_car_movement(): "speed" and "framerate".

Speed decides how frequent we sample a point in the trajectory, and the set of points are stored in “moves”. Then we send the movement to the server at localhost:5000. The frequency that we update the movement to the server is called framerate, because it is the same rate that Unity updates a car’s location. So framerate is about how frequent the car position is updated, and speed is how far a car travels between every two updates.

Future Work

The purpose for building this visualization tool is to facilitate deep learning for self-driving cars. So far, my supervisor and I have devised two approaches:

(1) Semi-Supervised Learning

The input of this learning will be visual data from streets.

We can first build a CNN to analyze the input and get embeddings of several channels. Each channel is a “type” of information, such as car information, pedestrian information, road information, light information, driver intention information and so on.

This CNN will then be fed into an LSTM layer which takes the previous 3 frames and generate a decision. The decision includes both the instant acceleration and instant change in direction.

(2) Reinforcement Learning

Although the aforementioned method seems promising, there are a lot of difficulties in retrieving a good amount of clean data for supervised learning.

To complement this, we will also try reinforcement learning. By reinforcement learning, we only need to simulate driving games in Unity and pipe out the data. This will need support from Unity's ML-agent, which enables us to make interactions with the game while a neural network is being trained.

The states will be a CNN encoding of the car's vision, the policy will generate an action with a certain randomness, and then it will receive feedbacks from the new state and reward. The reward for getting hit is a large negative number and the reward for each action is -1, so that the car will try to pass the intersection in the shortest time to maximize the reward.

Conclusion

This project is an important step toward the machine learning for self-driving cars. It can both make normal car movements and visualize a variety range of trajectories. There needs to be

future improvements to scale this visualization and apply it to other kinds of intersections. More adjustments will be made according to the training process of the neural network.

Acknowledgements

I sincerely thank Camera Culture Group at MIT Media Lab which offers me this project. Professor Ramesh and Pratik supervised my work, and my PhD mentors, Tristan and Guy, gave me a lot of guidance and inspirations while I was working on this project. I was also inspired and motivated by many nice people in the lab.

References:

Dresner, K., & Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2* (pp. 530-537). IEEE Computer Society.

Lee, J., & Park, B. (2012). Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems*, *13*(1), 81-90.

Zohdy, I. H., Kamalanathsharma, R. K., & Rakha, H. (2012). Intersection management for autonomous vehicles using iCACC. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on* (pp. 1109-1114). IEEE.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, *11*(3), 387-434.