UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Departamento de Ciências de Computação

A Smart City Simulation Framework

Tiago Marino Silva

São Carlos - SP

A Smart City Simulation Framework

Tiago Marino Silva

Supervisors: Richard Bunk and Slawomir Nowaczyk

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina Ciências de Computação do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação –

ICMC-USP para obtenção do título de Bacharel em Ciências de Computação.

Área de Concentração: Software Engineering and Computational Intelligence

USP – São Carlos
10/07/2022

Abstract

Simulations are used as tools to study the complex behavior of geographical and social phenomena. Many simulations are built from scratch or using a framework. These frameworks can be limiting, allowing users to focus only on a few domains. Therefore, we propose an open source framework for smart city simulations, which will speed up the development process and accommodate multiple domains through the contribution of the users. This is achieved through the right selection of the license and tools to build the framework. The framework expands on the Unity Editor, adding basic functionalities for building a self-organized agent simulation. The final result is a flexible framework that allows for multiple Artificial Intelligences (AI) to govern the agent's behavior while having other aspects of the simulation not governed by AI.

Index

CHAPTER 1: INTRODUCTION

1.1. Context and Motivation

Smart city is a city that uses sensors and digital solutions to optimize its services and networks for the benefit of its inhabitants and business [European Union, 2022]. It is common to simulate new solutions for smart cities before implementing them in the real city, since the data collected from these simulations may help researchers and other parties to understand the complex dynamics generated by it.

Because simulations can differ on many scales and domains, parties usually build their own tools and programs to help them achieve the simulation scenarios they need. But, building these types of tools is a complex task in itself and takes valuable time and resources which could be spent on simulating and analyzing the studied object instead. Also, there are limitations to the complexity of the simulation a party can build alone. With this in mind, we introduce the concept of an open smart city simulation framework. A framework provides a set of code and tools that facilitates the creation of a new system inside a certain domain. To achieve this, the framework uses components, patterns, and an architecture that the user can modify, reuse, and expand upon to achieve the desired solution faster.

Our framework allows developers to build a smart city simulation, by using self organized agents, and to share it between parties. The desired scenario is a real time simulation running on an open server, where parties can connect and input their own agent artificial intelligence and observe their behavior in the shared scenario. As a first step to achieve this scenario and as a proof of concept, this project focuses on building an open source framework with self organized agents in an offline environment. While being open source, users can contribute by creating and sharing new components and data, allowing others to build more complex simulations and to integrate data they wouldn’t have integrated otherwise.

This project was built in a five month time frame, from planning to implementation and report. It is part of an exchange program between the University of São Paulo and the University of Halmstad, and a bachelor’s thesis.

1.2. Purpose and Objectives

Our objective is to build an open source framework for simulating smart cities. It comes with an example simulation scenario which is based on a portion of the city of Halmstad, using Open Street Map (OSM) data, and uses the Needs Artificial Intelligence (AI) for the agent's decision making. This framework was developed using the realtime development platform Unity [Unity Technologies, 2022] and its component based architecture, extending the editor. The use of this platform will be highlighted in Chapter 3.

For the framework, being open source is important to allow different parties to contribute between themselves, by sharing components, systems, ideas and data. For example, one party can create the components for simulating the implementation of an electric scooter service in the city of Halmstad, while another party is simulating the implementation of a bicycle service. Perhaps, the code for creating these different services is similar and thus can be reused between the projects. Also, with both of them contributing, they can simulate having both services available at the city simultaneously, achieving a simulation scenario which wouldn’t be possible otherwise.

1.3. Work Organization

In Chapter 2, we describe the basic terminology used for the project, as well as the related work in literature. Next, in Chapter 3, we describe the development method, the framework and the demonstration. Finally, in Chapter 4, we present the conclusion and proposals for future work.

CHAPTER 2: RELATED WORK

2.1. Initial considerations

This chapter begins by describing relevant concepts, techniques, and terminology required to build the project. Then, it brings the state-of-the-art of the work relevant to our project and how they relate to and help with the proposed solution.

2.2. Relevant Concepts and Techniques

Since we are developing a framework, we use the Object Oriented Programming (OOP) paradigm, in which the concept of framework was created. For physics and rendering, we use a real-time development platform, Unity. It uses the C# language for scripting. For a framework, it is appropriate to provide clean code, which can be achieved by following S.O.L.I.D principles, utilizing design patterns when applicable and other guidelines presented in the book Clean Code [Martin, Robert C. 2009]. Finally, to demonstrate the capabilities of the framework, we implement an example simulation using Needs Based Artificial Intelligence and the A* pathfinding algorithm for our agents' behavior.

2.2.1. S.O.L.I.D Principles

S.O.L.I.D Principles is an acronym for five design principles in OOP:

- Single-Responsibility principle: states that a class or procedure should have only one responsibility, which is defined by the level of abstraction of it;
- Open–closed principle: states that a class should be closed for internal modifications, but open for extensions;
- Liskov substitution principle: states that if two classes implement the same interface, it should be possible to use either of them and get the expected behavior;
- Interface segregation principle: states that interfaces should be granular to a point that every class implementing that interface needs all the methods from that interface;

- Dependency inversion principle: states that high-level modules should be unaffected by changes in low-level modules and should be easily reusable.

They intend to make object-oriented designs more flexible, maintainable, and readable.

2.2.2. Needs Based Artificial Intelligence

Needs Based Artificial Intelligence is the algorithm behind the first The Sims game, which is described in [Zubek, R. 2010].

The algorithm receives as input the agent's internal state, the simulation state, and the possible sequence of actions and their rewards, and outputs the sequence of actions the agent will perform. To choose a sequence of action, the algorithm scores each sequence of action based on their rewards, and picks the one with the highest score or a random one weighting them by their score.

2.2.3. A* algorithm

The A* algorithm is a pathfinding algorithm commonly used in AI. The algorithm was described first in [P. E. Hart, N. J. Nilsson and B. Raphael 1968]. It uses a heuristic and a cost function to determine the total cost of traversing each node, then, from lowest to highest cost from the open nodes, it traverses the node's neighbors until it reaches the destination node or traverses all nodes.

2.2.4. Agents

Agents are used to represent entities in a simulation, such as humans, autonomous cars, or retailers. They commonly share these features: autonomy, heterogeneity, and being active [Crooks, Andrew T., and Alison J. Heppenstall 2012].

Agents can have a pre-defined behavior, as seen in agents from "Conway's Game of Life" [Gardner, Martin 1970], where every interaction of the agent is defined as a series of if and else statements. Or they can be self-organized, which means they don't have a scripted behavior, but a set of actions they can accomplish, and they decide these actions in an unpredictable manner, using optimization functions, AI and/or randomness.

Self-organized agents can adapt to the environment, and create situations and behavior which weren't predicted by the developers. This adaptability and unpredictability is beneficial for studying complex topics like sustainability, and enables an easier expansion of the simulation scenario and components than with pre-defined behavior, since it is not required to script the behavior of agents with every new component added.

2.3. Related Work

To create new policies for a city, city planners must consider all aspects of the city. Since when modeling from one perspective, a found solution can affect other aspects negatively. For example, adding a new road will reduce traffic flow, but will also increase the pollution in nearby areas. Thus, City Simulation Software (CSS) was invented to satisfy the integration of different perspectives to support decision-making. The software focuses on four main topics: Energy, Environment, Mobility, and Urbanization. It combines four different modeling tools and their outputs to generate static and dynamic heatmaps that display how source transport emissions are affected and formed by the cities geographic data [M. Daniel et al. 2021]. Thus, a simulation that considers more aspects of the city has a higher quality of use. Hence, the project being open source allows it to grow in complexity, covering more aspects with time, making it a valuable resource for city planners. Similar to this work, another simulation platform has been developed but uses uncertainty as a tool to approximate reality. So some uncertainty is added to the sensor's captured data, following a defined randomness distribution, and to the event layer, by having random events like accidents. This platform implements a layered architecture, utilizing different open source solutions for the simulation [Shuyang Dong, Meiyi Ma, and Lu Feng 2021]. As described in [Shuyang Dong, Meiyi Ma, and Lu Feng 2021] work, modeling the simulation with uncertainty approximates reality and should be allowed in our framework. Thus, agent actions must be cancellable and the same type of agent should have different behavior based on its internal state and maybe some randomness.

The use of open data, e.g. from OpenStreetMap (OSM) [OpenStreetMap Foundation, 2022] can be found in many simulations, especially when considering microscopic traffic simulation. A popular package for modeling intermodal traffic systems is SUMO, which also comes with supporting tools for tasks such as route finding and emission calculation [Eclipse Foundation, 2022] [Ma, X.; Hu, X.; Weber, T.; Schramm

2021] [Shuyang Dong, Meiyi Ma, and Lu Feng 2021] [L. CODECÁ, J. ERDMANN and J. HÄRRI 2018]. This package has been used to create a parking management framework for smart cities simulations, communicating with SUMO via python scripts [L. CODECÁ, J. ERDMANN and J. HÄRRI 2018]. To design a use case scenario for our framework, we must account for the use of OSM open data. Also, the framework should be flexible to permit these types of communications, so these softwares can be reused inside of our framework, allowing for the user to save development time.

For defining the behavior of the agents of the simulation, one can use a bottom-up or top-down approach, which results in simulating individuals that exchange information or simulating a bigger aggregation of these individuals' behavior at once, respectively. For our framework, we decided on the bottom-up approach, allowing users to define the behavior of individuals. This approach allows for unexpected situations to occur during simulation, giving unexpected insights into the intricacies of the study object [Crooks, Andrew T., and Alison J. Heppenstall 2012]. This approach also is more flexible when accounting for changes in the simulation, since you can model agents that adapt to the environment. These kinds of adapting agents are known as self-organized agents, and should be achievable in our framework since they are valuable resources when studying sustainability. This value in sustainability studies comes from the fact that these agents create a series of situations the researcher could not have anticipated.

2.4. Final Considerations

In conclusion, this project is developed by using OOP, Unity, and open data from OSM. In the next chapter, Chapter 3, we will describe the method for designing and implementing the framework and discuss some design decisions.

CHAPTER 3: METHOD

3.1. Initial Considerations

This chapter first describes the steps taken to build the framework and the example demonstration. Next, it describes in detail the implementation of the framework and the concepts used to increase its flexibility and maintainability. Finally, it describes the algorithms used in the example demonstration and some final considerations.

3.2. Method Description

The objective of this project is to develop an open source framework for building smart cities simulations. To achieve that, we designed our framework inside a Game Engine platform, Unity. It allowed us to focus on the simulation aspect of the framework, instead of physics, rendering, and user interfaces, which were handled by the engine. After choosing the platform the framework would be built upon, we designed the framework's architecture and implemented the solution and demonstration. Finally, we designed two use case scenarios that should be achievable with our framework. Another important step was choosing the right license.

3.3. Method Description

3.3.1. Framework Design and Implementation

Our framework is designed based on four important concepts:

1. Agents:

   Our agents can perform two types of tasks: exchange resources and translate.

   Their behavior is driven by an AI, which queues tasks for the agent to perform.

2. Resources:

   Resources are quantifiable objects the entities can carry and exchange between themselves.

They can represent material objects, e.g. money and iron, or immaterial, e.g. energy and electricity.

This allows simulations inside our framework to study sustainability since the user can insert limitations to the production of these resources.

3. Advertisers:

   Advertisers provide services for the agents, by broadcasting them to those who meet their requirements. These services can be anything the user implements, and involve exchanging resources and translating the agent. While interacting with agents, advertisers might store that interaction's data.

4. Transportation:

   Transportation systems are what allow agents to translate around the simulation environment. These systems return a list of available methods of transportation the agent can use when pathfinding to their destination. These methods can be walking, taking a bus, riding an electric scooter, and others the user implements.

Being an extension of Unity, our framework follows the component-based architecture used by the engine. Thus, we break our logic into smaller components that can be applied to objects to create different behaviors when combined.

3.3.1.1 Data Abstraction

An important design principle for developing applications such as games and simulations is separating the data from the logic of your application. This allows the designers to tweak and experiment with different parameters without the cost of compilation and the danger of introducing new bugs.

Unity allows developers to define objects that can be easily serialized into an asset file and referenced throughout the Editor. To define a type of object, one must implement a class that inherits from *UnityEngine.ScriptableObject*, then they can create instances of that class as asset files [Unity Technologies, jul. 5th 2022].

The framework allows users to create new types of resources by simply creating new asset files from the *Resource* class. This pattern of utilizing scriptable objects as an

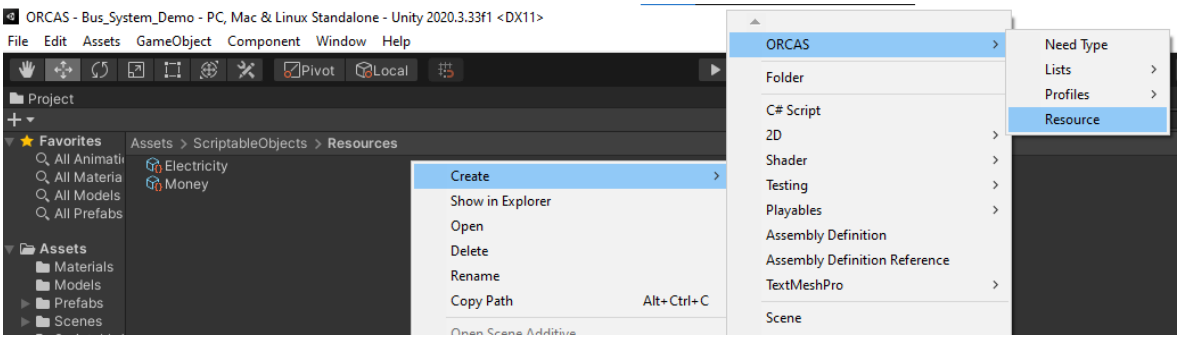
enum was described by [Hipple R., 2017]. Source Code 1 demonstrates the *Resource* class and Figure 1 demonstrates how to create a new resource asset file. After creating the file, the user has to rename it to their specific resource, reference Figure 2. Then, the user-created resource can be referenced in components and other *Scriptable Objects*, as demonstrated in Figure 3.

Source Code 1 - Resource

```
5       [CreateAssetMenu(fileName = "New Resource", menuName = "ORCAS/Resource")]
6       public class Resource : ScriptableObject {}
```

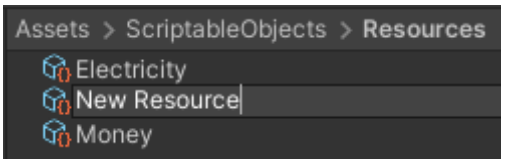
Source: this report's author

Figure 1 - Resource Creation



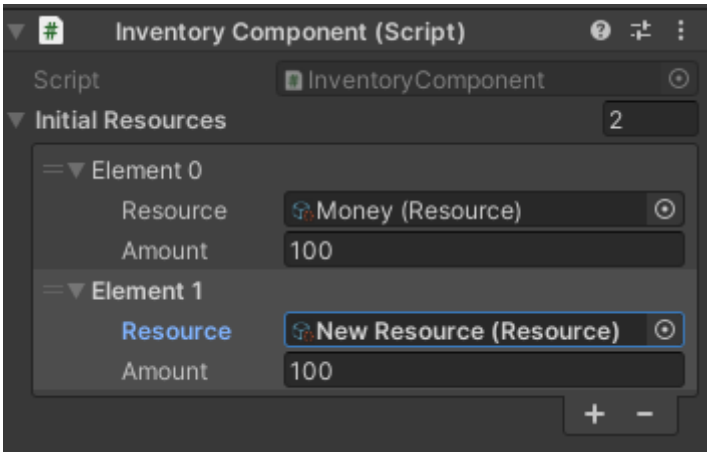
Source: this report's author

Figure 2 - Resource Renaming



Source: this report's author

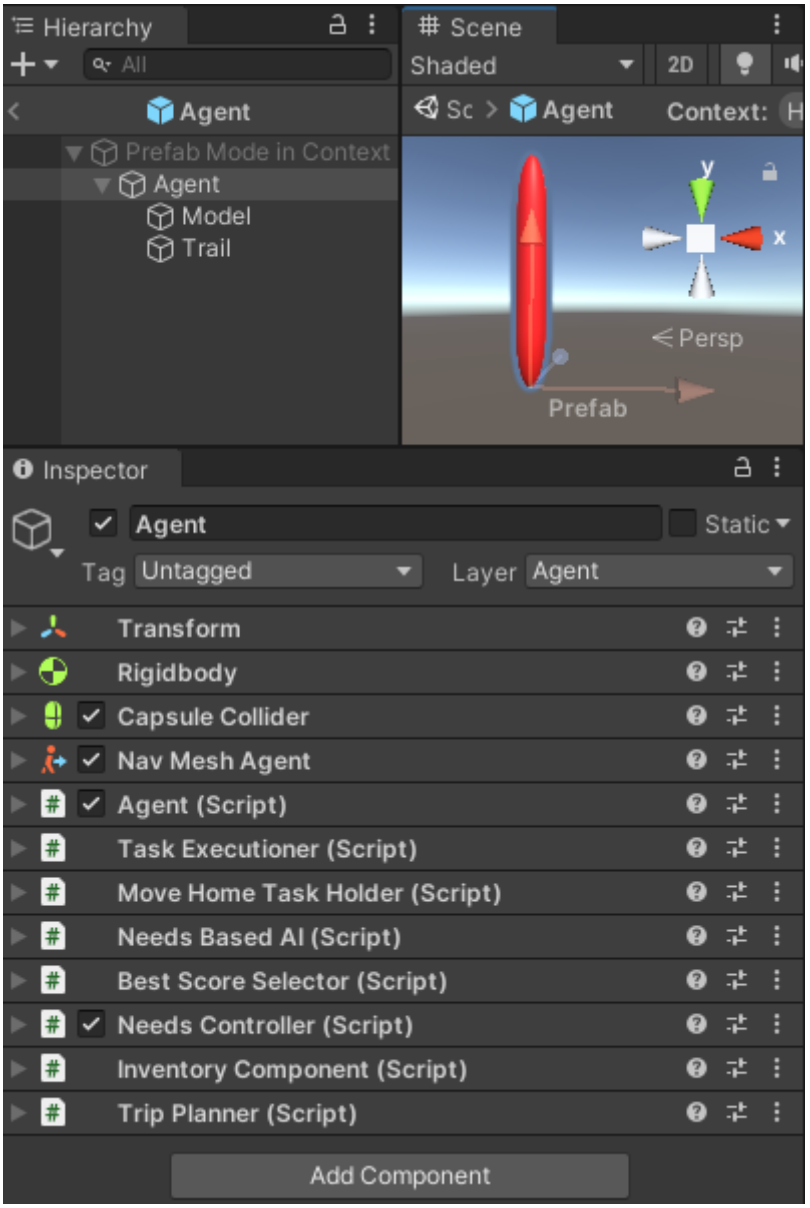
Figure 3 - Referencing Resource in Inventory Component



Source: this report's author

Another feature utilized in the framework is the Prototype design pattern, through the use of Unity's prefabs [Unity Technologies, jul. 5th 2022]. Unity's prefab system allows users to create asset files that describe a *GameObject*, with its hierarchy and components. This way, the user can create, for example, his own agent through the editor or use the framework-provided agent prefab and modify it for their needs. Figure 4 displays the view of the framework's default agent. Notice how it utilizes both Unity and custom components. Also, the agent's logic is composed of different components, instead of being coupled to one single class. It allows various implementations of the agent's behavior which will be demonstrated in Section 3.3.1.3.

Figure 4 - Default Agent



Source: this report's author

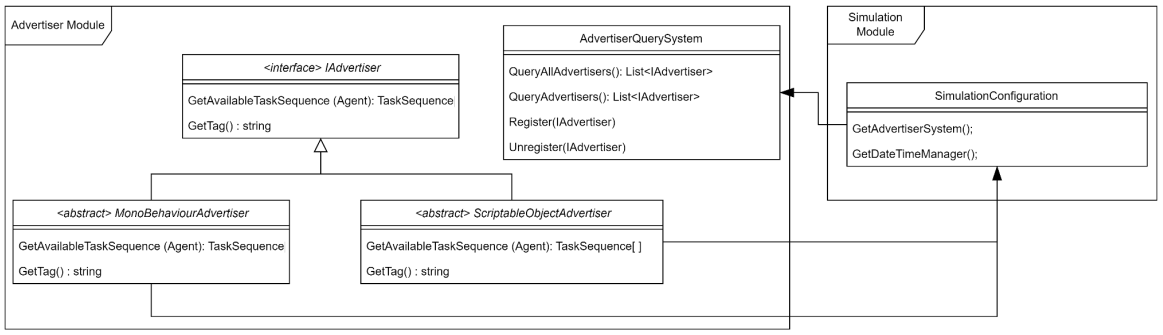
3.3.1.2 Advertisers Interface and Components

The advertiser is a simple interface containing two methods, represented in Figure 5. These methods allow other components to identify advertisers by tag and to get notified of which task sequence they are advertising.

To manage available advertisers at the simulation runtime, we use the *AdvertiserQuerySystem* and *SimulationConfiguration* classes. Communication between

these classes is represented by the diagram in Figure 5. Notice how we have two implementations for the *IAdvertiser* interface, those are *MonoBehaviourAdvertiser* and *ScriptableObjectAdvertiser*. These classes are Advertisers that register and unregister themselves to the SimulationConfiguration's *AdvertiserQuerySystem* when they are enabled / disabled and they inherit from their respective Unity classes. The user can implement these abstract classes to add new advertisers to the simulation without worrying about passing down references through his program or they can implement the *IAdvertiser* interface and be responsible for registering it to the desired AdvertiserQuerySystem.

Figure 5 - Advertiser System Diagram



Source: this report's author

The AdvertiserQuerySystem is used by the agent to listen to advertisements from the simulation. When an agent wants to decide its next course of action, it queries the available advertisers and decides, through its AI, which sequence of tasks to perform.

3.3.1.3 Agents Interface and Components

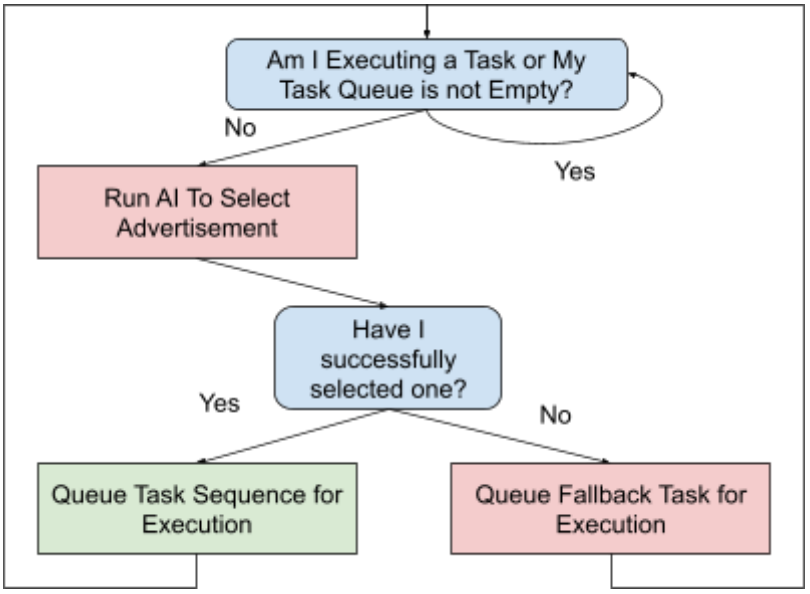
The agents logic is partitioned into four main components:

1. Task Executioner
2. Agent AI
3. Trip Planner
4. Inventory

Each component is responsible for a part of the agent's behavior. The *Agent* class itself is responsible for keeping a reference to these components, allowing them to be

accessed by other systems interested in the agent's public data. It also runs the main loop described in Figure 6.

Figure 6 - Agent Main Loop



Source: this report's author

The main component of the Agent is the *Task Executioner*, which manages the agent's task queue and allows it to execute tasks in the First In First Out (FIFO) order. A task is an implementation of the abstract class *Task* and contains the logic for running the specific task in hand, aborting it, and notifying other components that the task has finished executing successfully or not. Tasks can run in multiple frames, due to their logic being contained in an IEnumerator function. Reference Source Code 2 for more in-depth implementation details.

Source Code 2 - Abstract Task Class

```
7       public abstract class Task
8       {
9           public event Action<bool> OnExecutionEnded;
10          protected CancellationToken _cancellationToken;
11
12          public abstract IEnumerator Perform(GameObject agent);
13          public void Abort()
14          {
15              _cancellationToken.RequestCancellation();
16          }
17
18          protected void InvokeOnExecutionEnded(bool success)
19          {
20              OnExecutionEnded?.Invoke(success);
21          }
22      }
```

Source: this report's author

The second most important component of the Agent is the AI. Agents of the simulation can have different implementations of the *IAgentAI* interface, which allows multiple parties to collaborate on the same simulation. The implementation of the AI is responsible for receiving the available advertisements for the agent and deciding which one the agent should select or notifying if none were selected, resulting in fallback task execution. By using an interface and reducing the responsibility of the AI to this one simple function, the framework follows the Interface Segregation Principle and Liskov Substitution, allowing users to implement their own AIs and to change between agents which one is being used.

The third component of the agent is the Trip Planner, it is responsible for calculating the path and its cost when the agent wants to translate. The logic for pathfinding is an adaptation of the A* algorithm, where the G cost is the cost of time and resources of traversing the node and the heuristic is the distance between the current node and the destination. It basically receives a destination position, the agent data, and the current time, and calculates the path or the cost for that trip. The user can implement their own *ITransportationSystem* and subscribe it to the agent's transportation system list so it will be considered during the trip planning phase.

The last component of the agent is the Inventory component. This component is responsible for managing the resources the agent has. Through its interface, other components can add or remove resources from the inventory. Although it was created for the agent, it can be used by any other GameObject in the simulation that needs to carry resources with it. For example, you can add an inventory component to a Factory advertiser and have the factory manage its money and electricity resources to operate.

3.3.2 Demonstration implementation

To demonstrate the usage of the framework, we implemented a use case environment based on The Sims' Needs AI [Zubek, R. 2010]. In this environment, the agents have needs that need to be fulfilled, which are represented by scalar values from [0, 100]. These needs decay over time and must be attended to. The AI scores the advertisements based on the need it fulfills, the resource it rewards and takes, and the cost of the transportation. This behavior is described in Section 3.3.2.1.

Also, we implemented two forms of transportation system, walking, and bus. Thus, when planning a trip, the agent can decide to walk to the destination, or walk to a bus stop, take a bus, and so on. The differences between both systems are that walking from point A to B is slower than taking a bus from A to B, but when walking the agent goes directly to its destination, while when taking a bus it has to go first to a bus stop and then the bus can only take him from one stop to another.

3.3.2.1 Needs Based AI in our context

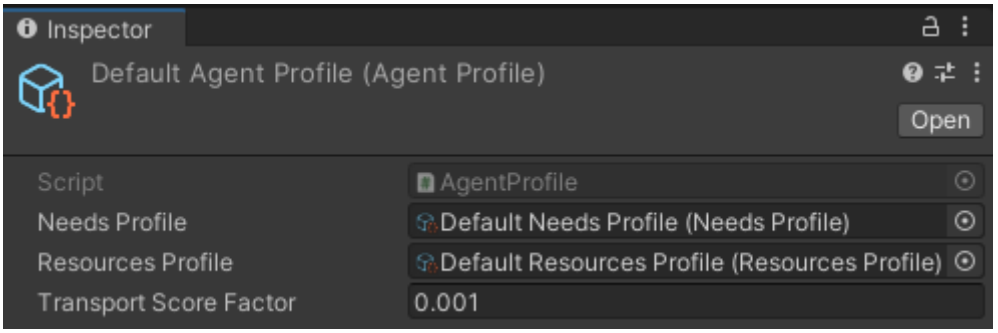
As the name implies, the original needs-based AI takes into account only the needs that will be modified when scoring advertisements. To include other forms of scoring advertisements, we created an interface *IReward*, which returns a numeric value representing the score of that task. Then we implemented three types of rewards: NeedReward, ResourceReward, and TransportationReward. These rewards change their cost based on the needs modified, resources consumed/gained and cost of transportation, respectively.

To allow for different behavior in agents, we implemented Profiles, which describes an agent's preferences when dealing with needs and resources. There are two types of profiles, needs profile and resources profile. They both are implemented as *Scriptable*

Objects to allow for the creation of instances of these classes as asset files. Then we have an Agent profile, which contains both these profiles and a scalar factor for scoring Transportation Reward. Figures 7, 8, 9 represent the data of these profiles.

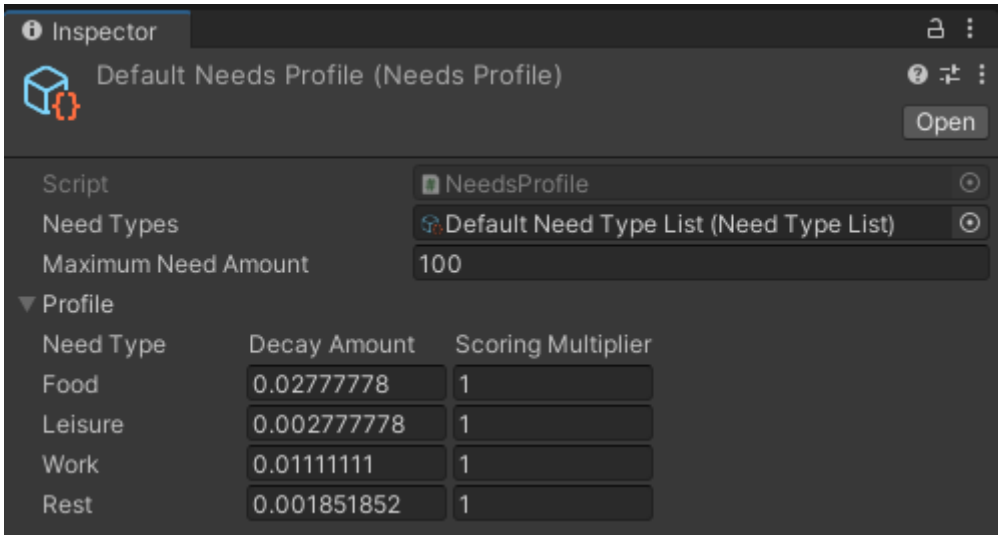
At the end of the Needs Based AI algorithm, we have a list of advertisements sorted by their scores, then we can choose a strategy to select which advertisement the agent will select. This allows us to select the best-scored advertisement or add randomness by selecting a random one based on the weight of its score. Thus, the framework utilizes the Strategy Pattern to decide which advertisement the agent will select and these strategies are defined as components, so the user can change them on the agents' prefab to add variation. This pattern is described in Figure 10.

Figure 7 - Agent Profile



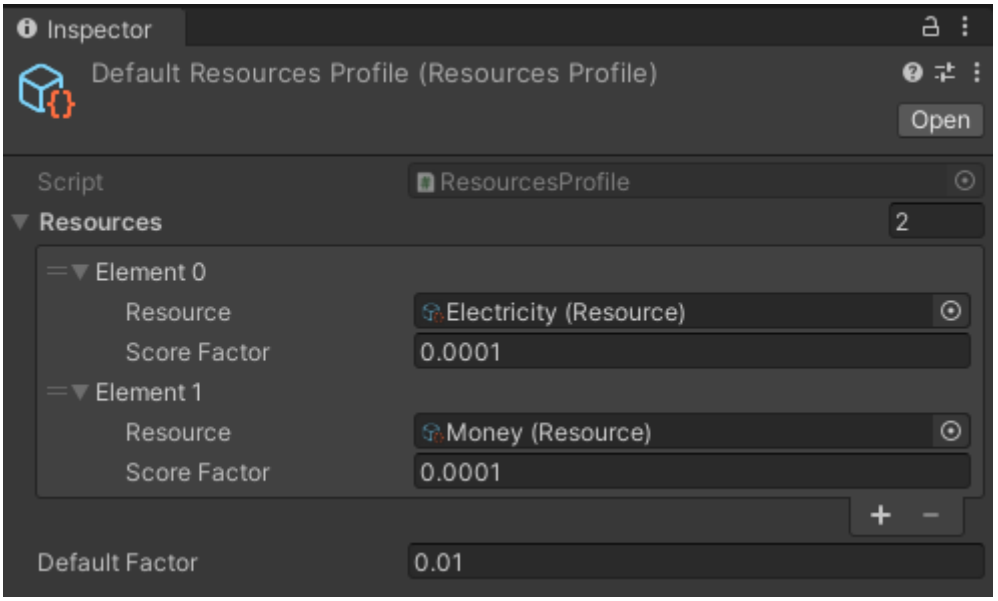
Source: this report's author

Figure 8 - Needs Profile



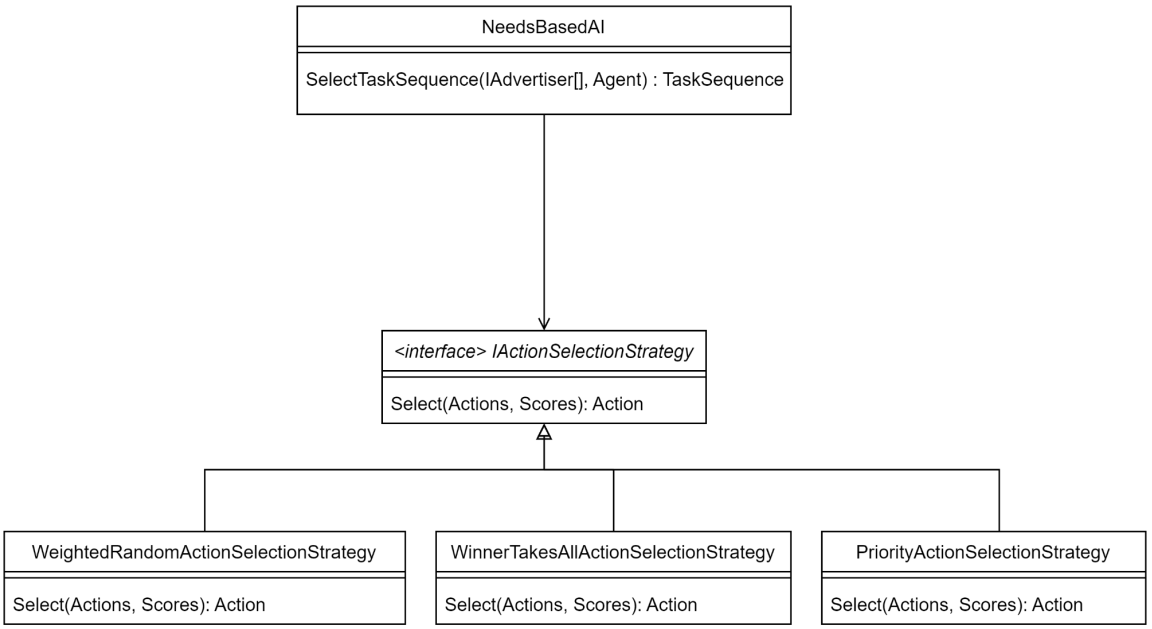
Source: this report's author

Figure 9 - Resources Profile



Source: this report's author

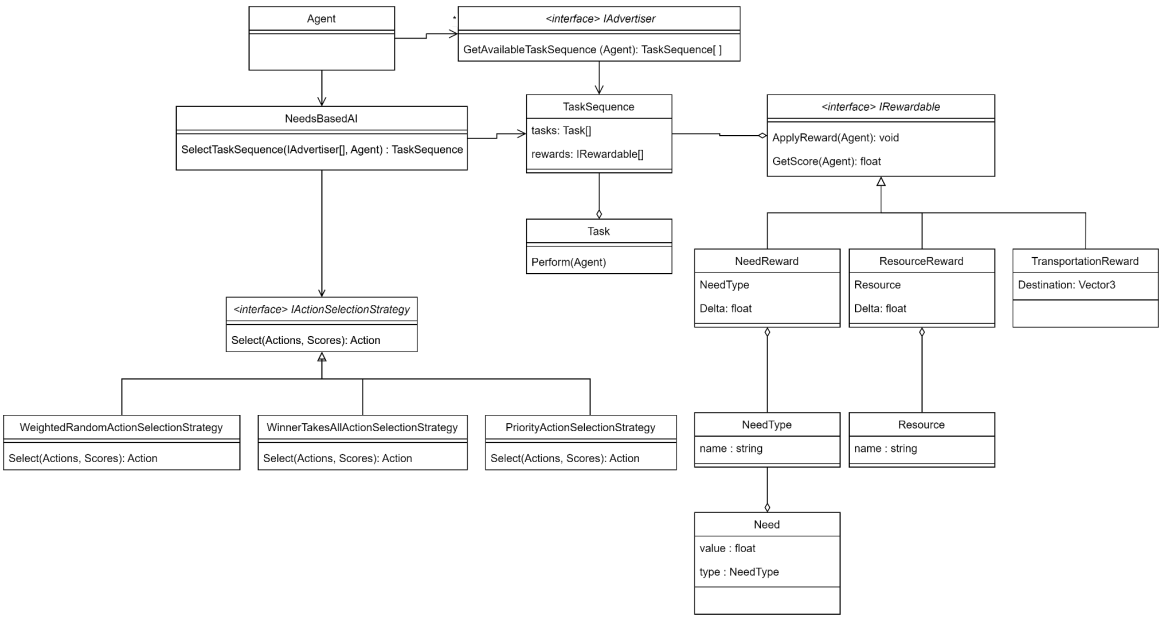
Figure 10 - Advertisement Selection Strategy Pattern



Source: this report's author

A diagram describing the Needs Based AI approach for the agents is modeled in Figure 11.

Figure 11 - Needs Based AI Class Diagram



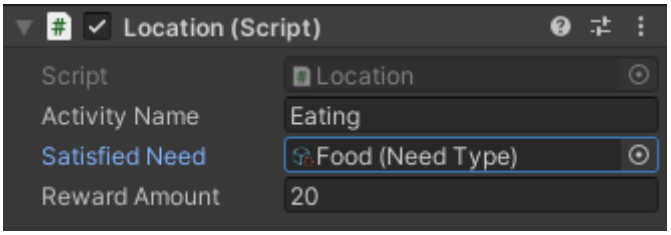
Source: this report's author

3.3.2.2 Implemented Advertisers

For the demonstration we have two types of advertisers, being both MonoBehaviourAdvertiser implementations: Location and WorkPlace.

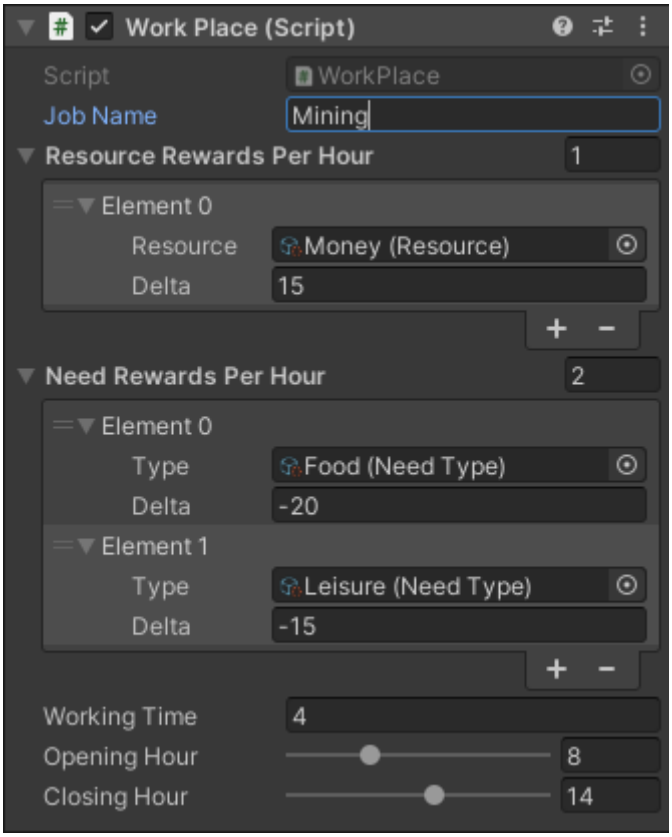
The Location Advertiser can advertise a sequence of tasks that will reward the agent with a specific need for an amount. The need and amount it rewards are specified in the component serialization, which allows a lot of variation. An example of a Location component is displayed in Figure 12. The sequence of tasks the agent performs is a move task, which moves it to the position of the Location GameObject, and a work task, which has the agent wait 1h (in simulation time) and then be rewarded the promised need amount. On the other hand, the WorkPlace advertiser is more complex and allows multiple rewards per hour the agent has to work, these rewards are both in resources and needs. Also, the WorkPlace has an opening and closing hour, which means it only advertises tasks when it is open. All these parameters can be configured, as shown in Figure 13.

Figure 12 - Location Component



Source: this report's author

Figure 13 -WorkPlace Component



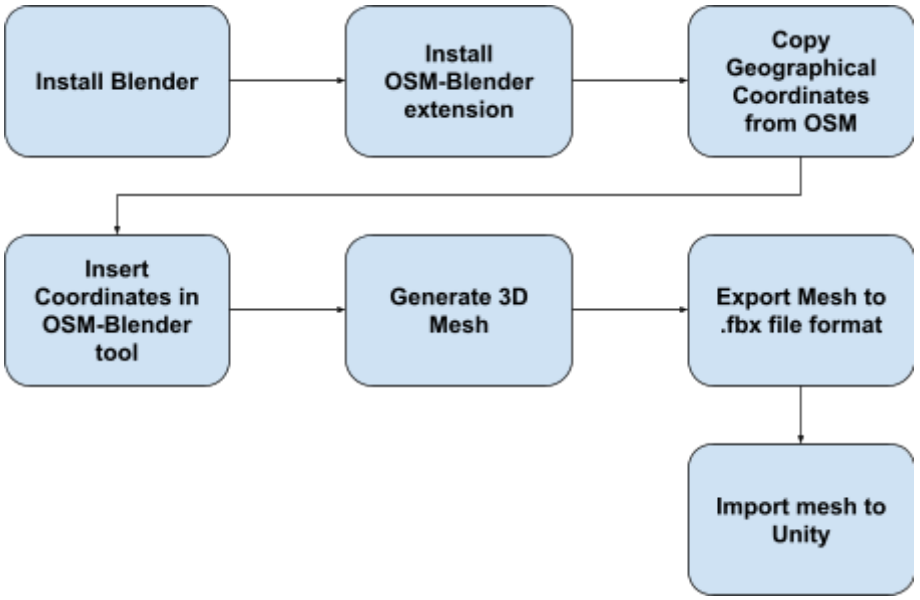
Source: this report's author

3.3.2.3 Importing Geographical Information System Data

As terrain for our main demonstration, we imported a slice of the city of Halmstad, Sweden. This terrain was generated as a 3D mesh in Blender, using the extension OSM-Blender [Gumroad, 2022], which reads data from OpenStreetMap (OSM) to generate the mesh.

The process is described in Figure 14.

Figure 14 - Generating Terrain Mesh

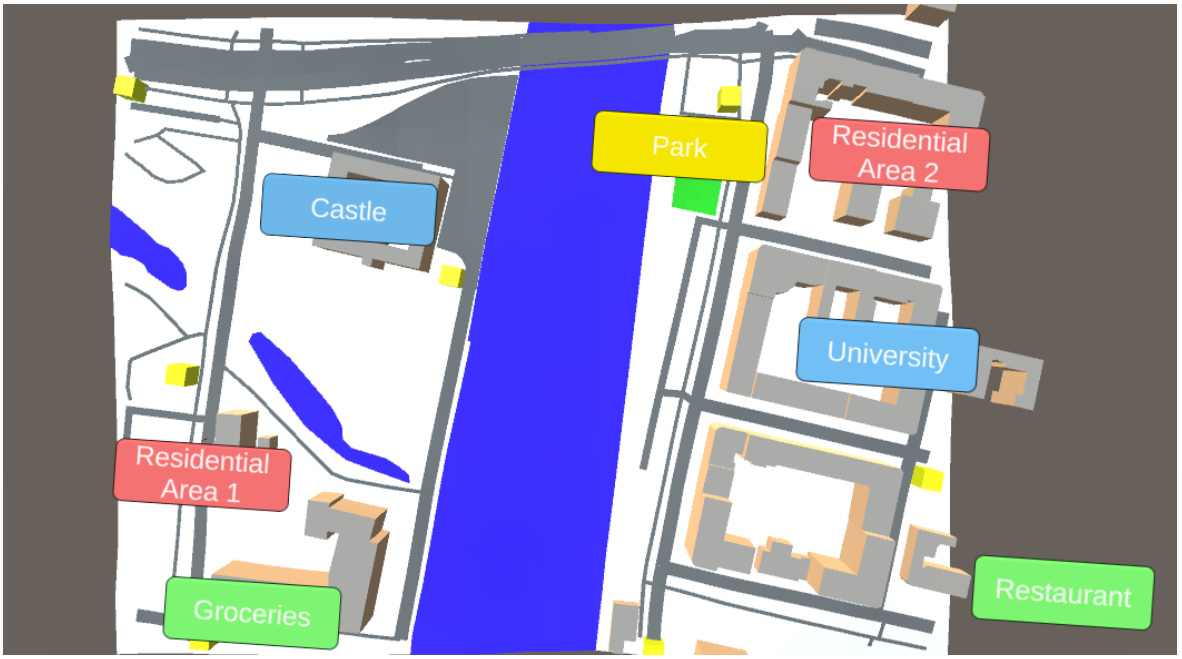


Source: this report's author

3.3.2.4 Demonstration results

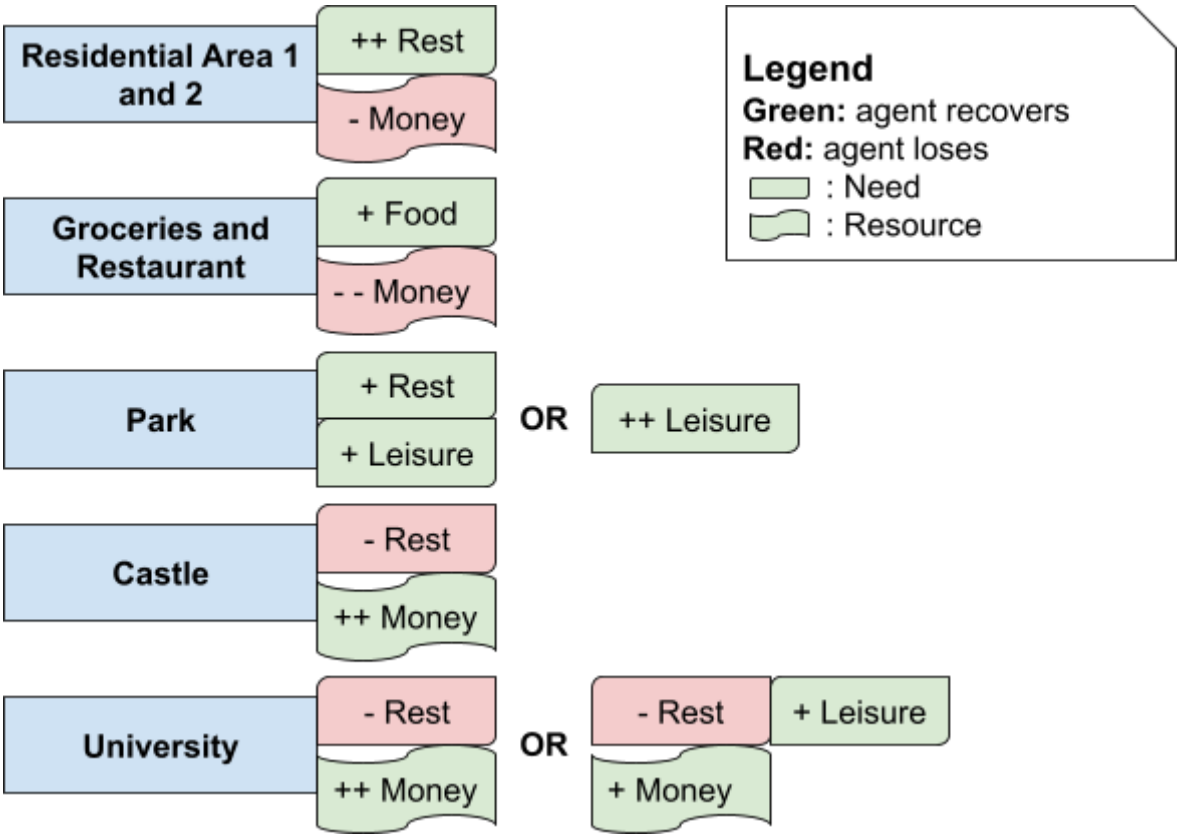
The demonstration scenario was built with a few advertisers that fulfill the three needs of agents on our simulation: food, leisure and rest. The areas containing advertisers are displayed in Figure 15, and the advertisements of each area are described in Figure 16. We have two agents on our simulation, each with a different profile: lazy and hungry. The lazy agent prioritizes the need of rest, while the hungry agent prioritizes the need of food. After running the simulation for approximately 1200 seconds, we can visualize the agents' needs over time in Figures 17 and 18.

Figure 15 - Demonstration areas



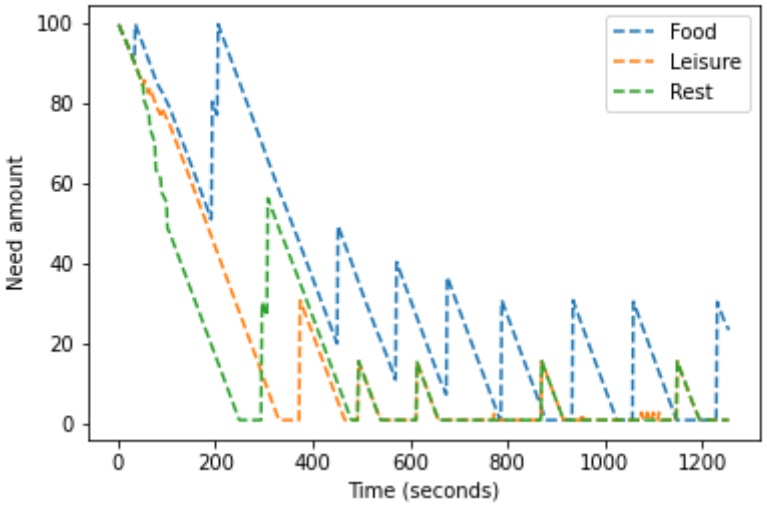
Source: this report's author

Figure 16 - Demonstration Advertisers



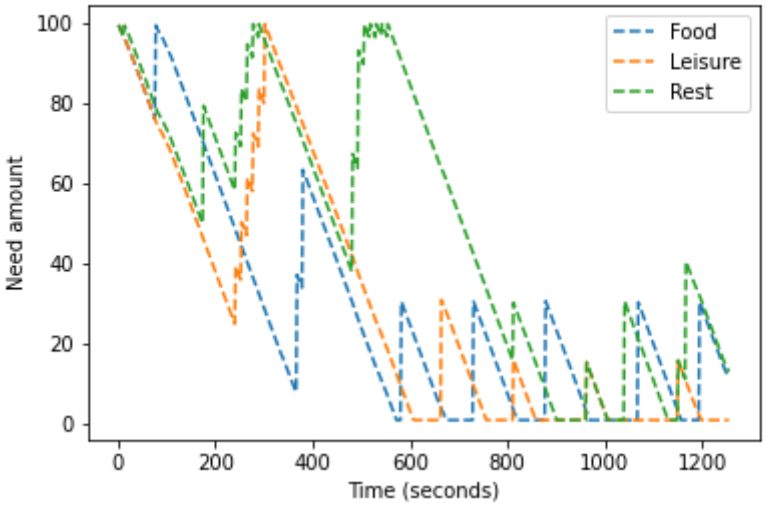
Source: this report's author

Figure 17 - Hungry Agent Needs Over Time



Source: this report's author

Figure 18 - Lazy Agent Needs Over Time



Source: this report's author

3.3.3. Use Case Scenarios

As mentioned in Section 3.2, we designed two use case scenarios for the users of the framework. These scenarios are described as follows:

3.3.3.1 Bus Schedule

In order to reduce the covid spread, the current administrator of Halmstad wants to change the bus schedule. First, they'll need to implement bus stops with the original city's bus schedule and tweak a few parameters to simulate a good enough approximation of the real city. Then, they can add/remove buses and bus stops and change the schedule to observe how the agents react. When achieving a good simulation scenario, the data collected can be used to further evaluate the implementation of the new bus schedule. All these steps should be achievable with the framework.

The objective of this scenario is to maximize the number of passengers on the bus by a threshold while having the following constraints:

- Limited available amount of buses
- A bus only operates at one bus stop at a time
- A bus takes time to travel from one stop to another
- The time is relevant to the distance between two stops
- Limited amount of agents that can fit per bus
- Agents should use the bus only when it's possible to arrive on time
- Agents should own enough resources (money)
- Bus Advertiser should have enough resources to operate

3.3.3.2 Park Planning

The city administration wants to build a new park, but there are plenty of available spaces to build on. To choose the best location, they can simulate the new park in different places and observe how the agents react. The data collected would be the number of human agents that visited the park, the reasons (leisure or passing through), and the date/time of the visits. Having this data will help evaluate the best location to build the park and the expected date/time peak of people at the park.

The objective of this scenario is to maximize the weekly average of leisure visits while having the following constraints:

- Agents consume limited resources to reach the park
- Park working hours

- Agents schedule (allow only for limited visit times)

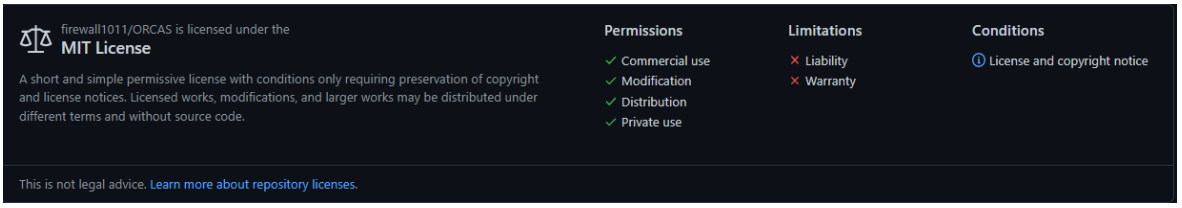
3.3.3.3 Note on Use Case Scenarios

Although these use case scenarios were designed, they were not implemented due to time constraints. But, they served as parameters for how the framework should be designed to allow for their creation.

3.3.4. Choosing the License

Since the objective is to have multiple parties contribute to the project and allow private companies to utilize it, we chose the MIT License [Github, 2022], described in Figure 19. It is a permissive license that only requires the preservation of copyrights and license notices. It allows users of the framework to distribute their work and modification under different terms and without the source code. Also, it permits commercial and private use. The project can be found at https://github.com/firewall1011/ORCAS.

Figure 19 - MIT License



Source: https://github.com/firewall1011/ORCAS/blob/main/LICENSE

3.4. Result Analysis

The framework utilizes design patterns such as Strategy and Prototype to implement its features and allow for user customization. Also, the data serialization features allow for easy modification of the simulation parameters and the component architecture allows for a variation on objects based on their components and their data. As described in Sections 3.3 and 3.2, the use of S.O.L.I.D Principles results in a more robust software that is flexible and maintainable.

From a user experience perspective, a user not used to Unity may need some time to adapt to the patterns and the editor. But, an experienced user should feel comfortable managing assets such as prefabs and scriptable objects, as well as designing their objects through components.

Building a simulation environment with the current state of the framework is a long task because it requires manually placing objects around the world, like agents and advertisers. As mentioned in Section 4.3, automatizing the instantiation of objects in the world based on the mesh file imported from OSM would considerably speed up the environment creation of the simulation, leaving more time for parameter tweaking and testing.

From the demonstration scenario, discussed in Section 3.3.2.4, we can observe the results in Figures 17 and 18. The hungry agent maintains the food need higher than the other needs, and due to geographical distance, the leisure need was left unattended when compared to the others, since the only advertiser that recovers a considerable amount of leisure is far away from both Groceries and Restaurant advertisers, as shown in Figure 15. On the other hand, the lazy agent tried to keep the rest need high, earning some leisure need simultaneously because of the Park advertiser. But, at the end, it opted for the offer that recovered more rest, instead of recovering both needs. Thus, although the agents use the same AI algorithm, they can achieve different behavior based on the environment, which is composed of advertisers and transportations, and on their profile. This demonstrates that different simulations can be built just by tweaking these initial parameters.

3.5. Difficulties and Limitations

During the development of the project, we encountered some difficulties and limitations. These are listed below:

3.5.1. Choosing the tools

While choosing the appropriate tools for development, we considered both state-of-the-art Game Engines: Unreal Engine 4 (UE4) [Epic Games, 2004-2022] and

Unity [Unity Technologies, 2022]. Since visual scripting allows for non-developers to script simple behaviors and has a lower learning curve than programming, and, for future work, having a robust network solution was important, we initially chose UE4 as our main platform for the framework. But, due to hardware limitations, it was impossible to conduct the work in it, since when trying to use C + +, the compilation times were from 5 to 30 minutes and when developing through visual scripting alone, it takes a considerably longer time than through normal code. Thus, in the middle of the project, we changed platforms and had to reimplement the framework in Unity and C#.

3.5.2. Project Management

At the beginning of development, I underestimated the value of planning and managing the project. That slowed down the development process and could be avoided with more effort in that area.

3.5.3. Testing

Testing AI can be a hard task to achieve, but it is a valuable process to include during development because it provides confidence that the code is producing the expected results. Since we didn't implement tests for the AI, a lot of time was lost debugging and understanding the behavior of the agents.

3.6. Final Considerations

To conclude this chapter, we built a framework capable of simulating agents in any environment, but to build such an environment is a hard task. Some known improvements are mentioned in Section 4.3, where we discuss future work possibilities. The result of the framework is a flexible program, while being flexible requires more work on the user side to build the end result, it also gives more liberty and expands the possible domains the framework can be used for. The next Chapter, Chapter 4, discusses the contributions of this project on a professional and personal level, some considerations regarding the bachelor's program, and proposals for future work.

CHAPTER 4: CONCLUSION

4.1. CONTRIBUTIONS

The presented work contributes to the areas of software engineering and simulation by introducing a framework capable of building self organized agent simulations in the context of smart cities and taking the first step on achieving the open smart city framework. Also, by being open source, contributions can expand the domains covered by the framework and allow for more complex simulations to be built and different parties to interact.

On a personal level, this project has helped me develop my scope and time management skills, as well as practice software design patterns and principles.

4.2. CONSIDERATIONS REGARDING THE BACHELOR PROGRAM

The bachelor's program in computer science helped me understand the intricacies of the computer, from a hardware and software perspective. This knowledge allows me to learn new technologies and concepts with ease and understand how they work on a surface level without looking deep into them.

For this project, the course Open Source taught me how licenses work and how sharing software code can be valuable. While SSC0770 - Introduction to Digital Games Development, MAC0472 - Agile Methods Lab, and MAC0413 - Topics in Object Oriented Programming developed my interest in software engineering and clean code. During and after these courses, I've studied and applied concepts of software design and engineering in all my projects and that helped me improve as a software developer.

The most important aspect of graduation were the student groups, events, and opportunities that the university provided. For software developers, it is important to program at all times, it is through analyzing our mistakes that we can improve. Thus, being part of the Fellowship of the Game (FoG) group was important for me, both on an academic and social level. As a member, I always had a side project to work on in a team

and I also had the opportunity to lecture classes in the course Introduction to Game Development and for the local community. Another opportunity I had was to coordinate the development of the game for the event SEMCOMP 24, being a coordinator helped me develop soft skills and managing skills, which were important for this project as well since I had to plan and manage my time.

The weak points found in the bachelor's program were the following:

- SSC0124 - Object-Oriented Analysis and Design: could focus more on critically analyzing different solutions for the same software, instead of focusing on a "perfect" solution. Also, I felt that the importance of the discipline was not clear, since to that point all our softwares had to do was pass on the run codes tests. As an improvement, it is possible to design and implement real software during the course, with a client.
- 7600105 - Basic Physics I: the first part of the course is a revision of the contents we had at high school, then we learn concepts that we never use again throughout the course. It would be more interesting if we had learned concepts used in computer science, like the concepts required to build a physics engine, and maybe develop a basic one during the course.
- SSC0721 Software Testing and Inspection: the course doesn't cover test-driven development, neither has us developing tests for a real application. This lack of project during the course results in a shallow understanding of how to test software and how valuable having a good test base can be.

Besides that, the course gives us a solid foundation for understanding how the machine reads and runs our code. Some special courses I'd like to highlight are:

- SSC0770 - Introduction to Digital Games Development: the course makes students go through the whole process of designing, developing, and publishing a game, which creates a great learning experience.
- First and Second semester courses are important for learning memory management and how the C programming language works. This serves as a foundation for learning and using other languages.

- SSC0713 - Evolutionary systems applied to robotics: the course has the students developing an evolutionary system from the beginning, while the professor acts as a supervisor who encourages and guides them in their learning process.
- SCC0215 - File Organization: the course has students working with binary data, which is an important learning experience for serializing and deserializing data and debugging an application.

Finally, some suggestions are as follows:

- What lacks in most courses is applying the concepts the students are learning. That can be solved by having them develop an interdisciplinary project, in which they can apply everything they are learning in their current semester.
- Having more software engineering courses. Our neighbor bachelor program at IME has the courses MAC0472 - Agile Methods Lab and MAC0413 - Topics in Object Oriented Programming, which are great examples of how to inspire students to design better software and have them learn through experience.

4.3. FUTURE WORK

A simple proposal for future work is to document the framework and build more example simulations to help welcome users and inspire them to use the framework and contribute. Moreover, more complex tasks can be done to improve the framework:

The first idea is to build a distributed architecture for the agent's behavior, where the AI would run on the user's machine but would connect to a server running the rest of the simulation world. This approach is similar to a game server, but instead of human players, the players (agents) would be driven by AI. This solution allows multiple parties to contribute to each other, while still keeping their code private, and allows for the scaling of multiple agents since the heavy AI logic would be run on multiple machines.

The second idea is to facilitate the process of creating a new simulation environment from OSM data. This can be done by generating the mesh with individual objects and reading their properties to determine which components and data it should attach to it. For example, when importing a city, it can look for buildings marked as churches and attach a Location component that advertises a praying task. This would considerably speed up the process of creating new simulation environments. These properties and data would have to be mapped by the user, but then the manual work of attaching the components and looking for the buildings would not be necessary anymore.

Finally, future contributions can be managing the framework's repository. This means analyzing and accepting pull requests from contributors, building a community and helping users debug, and use the framework.

REFERENCES

Crooks, Andrew T., and Alison J. Heppenstall. "Introduction to agent-based modelling." Agent-based models of geographical systems. Springer, Dordrecht, 2012. 85-105.

Eclipse Foundation, 2022. Sumo About Page. Disponível em: <https://www.eclipse.org/sumo/>. Acesso em: 14 Mar. 2022.

Epic Games, 2004-2022. Home Page. Disponível em: <https://www.unrealengine.com/en-US>. Acesso em: 10 Mar. 2022.

European Union. European Commission Website, 1995-2022. Página sobre Smart Cities. Disponível em: <https://ec.europa.eu/info/eu-regional-and-urban-development/topics/cities-and-urban-development/city-initiatives/smart-cities_en>. Acesso em: 20, Jul. 2022.

Gardner, Martin. The fantastic combinations of John Conway's new solitaire game "life". Scientific American, United States, (October 1970), 223, 120-123, Out. 1970.

Github, 2022. License Page. Disponível em: <https://choosealicense.com/licenses/mit/>. Acesso em: 26 Mar. 2022.

Gumroad, 6 May 2022. Documentation Page. Disponível em: <https://github.com/vvoovv/blender-osm/wiki/Documentation>. Acesso em: 9 Jun. 2022.

Hipple R., Ryan Hipple. Unite Austin 2017 - Game Architecture with Scriptable Objects. Youtube, 20 Nov. 2017. Disponível em: Unite Austin 2017 - Game Architecture with Scriptable Objects. Acesso em: 24 Abr. 2022.

L. CODECÁ, J. ERDMANN and J. HÄRRI, "A SUMO-Based Parking Management Framework for Large-Scale Smart Cities Simulations," 2018 IEEE Vehicular Networking Conference (VNC), 2018, pp. 1-8, doi: 10.1109/VNC.2018.8628417.

M. Daniel et al., "City Simulation Software: Perspective of Mobility Modelling," 2021 Smart City Symposium Prague (SCSP), 2021, pp. 1-7, doi: 10.1109/SCSP52043.2021.9447384.

Ma, X.; Hu, X.; Weber, T.; Schramm, D. Experiences with Establishing a Simulation Scenario of the City of Duisburg with Real Traffic Volume. Appl. Sci. 2021, 11, 1193. https://doi.org/10.3390/app11031193

Martin, Robert C. 2009. Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ: Prentice Hall.

OpenStreetMap Foundation, 2022. About Page. Disponível em: <https://www.openstreetmap.org/about>. Acesso em: 14 Mar. 2022.

P. E. Hart, N. J. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968, doi: 10.1109/TSSC.1968.300136.

Shuyang Dong, Meiyi Ma, and Lu Feng. A smart city simulation platform with uncertainty. Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems. Association for Computing Machinery, New York, NY, USA, 2021. 229–230.

Unity Technologies, 2022. Home Page. Disponível em: <https://www.unity.com>. Acesso em: 25 Mar. 2022.

Unity Technologies, 5 Jul. 2022. Scriptable Object Documentation. Disponível em: <https://docs.unity3d.com/Manual/class-ScriptableObject.html>. Acesso em: 15 Mai. 2022.

Unity Technologies, 5 Jul. 2022. Prefabs Documentation. Disponível em: <https://docs.unity3d.com/Manual/Prefabs.html>. Acesso em: 9 Jun. 2022.

Zubek, R. 2010. "Needs-based AI." In A. Lake (ed.), Game Programming Gems 8, Cengage Learning, Florence, KY.