A Multi-Agent Approach to Simulate Autonomous Traffic with Games: How to Transform GTA-SA/SA-MP in Your Simulation Platform

Jônata N. Cirqueira*, Pedro C. Mesquita†, Rodrigo R. Novaes Jr.‡ and Sandro R. Dias§
Centro Federal de Educação Tecnológica de Minas Gerais
Belo Horizonte, Brazil
E-mail: {jonatanc0511*, pedrocesar1410†, rodrigo.novaes.jr‡}@gmail.com, sandrord@cefetmg.br§

***Abstract*—Urban mobility is among the main problems of the contemporary society. In this context, the vehicle automation technologies stand out in several aspects, such as reduction in accidents, congestion and emission of pollutants. Considering that, this work seeks to upgrade the simulation platform composed by Grand Theft Auto: San Andreas, and its modification San Andreas: Multiplayer (GTA-SA/SA-MP) that, according to the literature, is an appropriate environment for implementing autonomous vehicles networks. By using available structures in this environment, it is possible to perform three-dimensional simulations in many scales, ranging from a single village to an extensive intercity map. From the available structures in game, it was used a set of navigational nodes, which identify all the available routes for the vehicles in the roads. We used the navigational nodes to generate a weighted directed graph, to which we applied Dijkstra's and A* search algorithms. From that, it was observed that the vehicles were able to calculate and follow the best route from a source to a target node in any place of the map, obtaining a realistic environment to simulate and test solutions for the traffic, such as the autonomous intersection management protocol, which will be implemented in future.**

Keywords–Autonomous vehicles; multi-agent system; game; VANET; IoT.

I. INTRODUCTION

Urban mobility has proved to be one of the main problems we face in society. According to Texas A&M Transportation Institute, in 2017 the delays of industrial deliveries, along with the fuel consumption, caused a congestion cost of US$ 179 billion in the 494 urban areas of United States. Moreover, an average auto commuter wasted 21 gallons of fuel only at traffic congestion. The report also attested that average auto commuters spent 71 hours of extra travel time in the same year [1].

Besides monetary cost, many fatalities happen in the traffic. According to the World Bank Group, about 1.25 million people die on world's roads every year, while in average 20 or 50 million people are seriously injured. In America, traffic has been the leading cause of death since 1975 [2].

In addition, traffic congestion is a major aggravating factor for environmental problems. Vehicle emissions became the dominant source of air pollutants, raising the risks of morbidity and mortality, specially for commuters and individuals who live near roadways. When traffic flow is slow, regular speedups and slowdowns can increase travel time and diminish the dispersion of pollutants, elating by four times the emission of CO, HC and NO_x [3].

One of the emerging solutions for these problems is the traffic automation, currently leaded by the development of autonomous vehicles. They are proving to be effective and efficient, mainly due to their ability of taking deterministic and accurate decisions. Researches indicate that a vehicular network composed by 5% of self-driving cars already demonstrates significant advantages in the traffic flow [4].

Many simulation platforms were built to test autonomous traffic problems and solutions. Having a variety of them is important to validate different scenarios and perspectives. The two main perspectives for autonomous vehicles simulations are two-dimensional (2D) and three-dimensional (3D). The 3D perspectives are closer to reality, containing more details and problems to solve, like relief dynamics, consumption cost when driving uphills and downhills, among others [5]. However, due to the complexities and costs in building 3D software, most of the simulations are 2D, like the Texas University's AIM project [6][7].

Considering that, games like *Grand Theft Auto: San Andreas* (GTA-SA) [8] and its modification *San Andreas Multiplayer* (SA-MP) [9] team up to form a good environment to simulate multi-agent systems. This platform leverages many of the game's features, like an extensive 3D map containing multiple cities, physical models for mechanics and collision, different vehicle models, weather manipulation and many others [10].

Previous works have managed to successfully build an autonomous vehicles network inside GTA-SA/SA-MP [10]. Some of them could even build autonomous intersection management protocols, which is one of the hottest topics in autonomous traffic problems [11]. However, these works were expanded only to a small village in the game, being unable to scale all solutions for the whole game map [10]. Therefore, this work sought to implement an autonomous traffic system using the entire GTA-SA's map, whose detailed goals can be described as:

- Use the game's structures to identify vehicle's paths along the game map;
- Apply routing algorithms to understand if the game provides a consistent environment for simulations.

We could adapt GTA-SA/SA-MP's features to simulate autonomous traffic in the whole game map. Also, we found paths data for different objects, like pedestrians and boats, opening possibilities to interact with human agents and to develop sensor networks for sea navigation.

In Section I, the reader had a vision of the existing problems traffic automation want to solve, as well as some tools that can be used to achieve that. In Section II, we will present the base concepts and methodologies applicable. Section III describes the implemented methodology in a way to achieve the described goals. In Section IV, we expose the results and data acquired from this work. In Section V, we do a critical analysis on the data, as well as briefly explore future works.

 ISBN: 978-1-61208-795-5

II. CONCEPTS & METHODOLOGY

This section will use theoretical concepts to present methods to achieve our goals. In Section II-A, we do an analysis on how to use games to simulate an autonomous multi-agent system; this will require software engineering and design patterns understanding. In Section II-B, we'll understand which math and computational models, as well as algorithms, are necessary to apply in this system. In Section II-C, we define the problems we want to solve with this model, as well as the high-level solution that might fit in any kind of system.

A. Games and design patterns

We can use two object-oriented programming principles to say that an electronic game or software product is eligible to simulate autonomous agents and multi-agent systems: open/close and the dependency inversion principles [12].

Definition 1: **Eligibility of a software product to be used in modifications**. Let M be a software product, M can only be extensible to modifications when the following characteristics are present:

1) M's modules and libraries may be extended without being modified;
2) There is a set of public interfaces of M such that they may be implemented, incorporated and distributed with M.

By identifying an application that fits Definition 1, we will reduce and decouple the amount of work. The responsibility of simulating autonomous agents and multi-agent systems can be addressed to the researcher, while the dependencies of this work, like physical models, climate, objects and events can rely on the modified application only.

B. Routing & mechanics models

Graph theory is the study of the relation between elements of a set. $G = (V, E)$ is said to be a graph, where V is a set of vertexes and $E = (u, v)$ is a set of pairs, where $u, v \in V(G)$, meaning that a vertex u is related to v. A graph is directed when there is a relation over all edges $(u, v) \in E(G)$ such that u is said to be incident over v [13].

Graphs can be used to understand whether elements of a set are reachable from a source vertex. Also, we can apply the concept of paths or routes, where there is a sub-graph $P = (V, E)$ such that $P \subseteq G$, whose $V(P) = \{u_1, u_2, \ldots, u_k\}$, $k \leq |V(G)|$, where all edges are in the format $E(P) = \{u_1u_2, u_2u_3, \ldots, u_{k-1}u_k\}$. Basically, a route or a path is a sub-graph that establish a traversal inside the original graph [13][14].

Also, if there is a cost function $C : E(G) \rightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers, then we can calculate a route $P = (V, E)$ such that the cost to traverse $E(P)$ is said to be the distance of P. For instance, we say that the distance between two vertexes is the sum of all costs associated to the edges that connect them [14].

Lemma 1: **A path with minimal cost**. Let $G = (V, E)$ be a graph and $C : E \leftarrow \mathbb{R}$ a cost function, a path $P \subseteq G$ is said to be minimal if all sub-paths in P are also minimal.

Proof: Suppose that $P = \{P_1, P_2, \ldots, P_k, \ldots, P_n\}$, where P_1 is a sub-path of P that passes through P_k and reaches all the way to P_n. Now, P is minimal if all $P_i \subset P$ are also minimal.

Let's prove it by contradiction. Suppose that P_k is not minimal, then we know that $P = \{P_1, \ldots, P_{k-1}, P_k, P_{k+1}, \ldots, P_n\}$ have sub-paths from P_1 to P_{k-1} and P_{k+1} to P_n which are minimal. However, there is a minimal path P'_k whose distance is lesser than P_k. Therefore, the distance of $P' = \{P_1, \ldots, P_{k-1}, P'_k, P_{k+1}, \ldots, P_n\}$ is lesser than P, meaning that concatenating minimal sub-paths will result in a minimal path. ■

The process of identifying a path with minimal cost is based on Lemma 1. Therefore, Figure 1 lists an algorithm to calculate all minimal paths from a single-source vertex of a graph.

```
1: for all u ∈ V(G) do
2:     d(u) ← ∞ {Initial distance is unknown}
3:     p(u) ← ∅ {Identify which vertex is incident over u in the minimal path}
4: end for
5: d(s) ← 0
6: INSERT_HEAP(A, s) {Creates a priority queue A based on the distance to s}
7: while |A| ≥ 1 do
8:     u ← REMOVE_HEAP(A) {Removes the vertex with minimal distance from A}
9:     for all uv ∈ E(G), v ∈ V(G) do
10:        if d(v) > d(u)+ CALCULATE_COST(uv) then
11:            d(v) ← d(u)+ CALCULATE_COST(uv)
12:            p(v) ← {u}
13:            INSERT_HEAP(A, v)
14:        end if
15:    end for
16: end while
17: return (p, d) {Returns a pair with the minimal distances d to all vertexes and a set p that identifies incidences}
```

Figure 1. Algorithm to calculate the minimal cost of a path inside a graph $G = (V, E)$, such that the path starts from $s \in V(G)$ and terminates in all other reachable vertexes in V.

The algorithm in Figure 1 can be implemented in two well-known versions: Dijkstra's algorithm for shortest path and A* algorithm [15]–[17]. The difference between them is how the operation CALCULATE_COST is implemented. Please look at Figure 2 for each version.

```
1: return C(u, v)
```

(a) Standard cost function for Dijkstra's algorithm for shortest path.

```
1: return C(u, v) + h(v)
```

(b) Standard cost function for A*'s algorithm for shortest path.

Figure 2. Implementations for each cost function according to Dijkstra's and A* algorithms in a graph $G = (V, E)$, where $u, v \in V(G)$ are two vertexes, $C : E(G) \rightarrow \mathbb{R}$ is a cost function associated to the edge $uv \in E(G)$ and $h(v)$ is any consistent heuristic.

One of the best heuristics is the euclidean distance between two points in the space. Considering a map of three-dimensional coordinates, the distance between two points $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ is given by:

$$d(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}, \quad (1)$$

which can be applied as the cost function of a graph when its vertexes represent coordinates of a 3D map.

C. Problems and definitions

Since routing problems are modelled using graph theory, one way of modelling and solving traffic problems is combining graph theory with a multi-agent approach [13]. In this context, Problems 1 and 2 present the definition of an autonomous traffic model, whose agents are named as autonomous vehicles.

Problem 1: **Model an autonomous traffic system**. Given a set of vehicles A and a set of paths $P = (V, E)$, where V is a set of positions and $E = (u, v)$, $u, v \in V$, is a set of pairs that indicate that v is reachable through u, output a set of routes $R_i = (V_i, E_i)$, $\forall R_i \subseteq P$ that a vehicle a_i, $\forall a_i \in A$, can traverse autonomously.

Problem 2: **Model an autonomous vehicle**. Given a vehicle a and a set of paths $P = (V, E)$, where V is a set of positions and $E = (u, v)$, $u, v \in V$, is a set of pairs that indicate that v is reachable through u, output route $R = (V', E')$, $R \subseteq P$ such that a can traverse autonomously.

According to the definitions of Problem 1, considering G a set of graphs and V a set of vertexes, let $\Phi : G \times V \to G$ be a function that receives $P = (V, E)$ and an initial vertex $s \in V(P)$, such that it outputs a path $R \subseteq P$. In essence, Φ outputs a route starting from vertex s. Also, let $\Omega : V \to V$ be a function that receives a set of vertexes and outputs a random vertex of the set. Henceforth, consider the algorithms in Figure 3 as solutions for Problems 1 and 2.

```
1: R ← ∅
2: for all a ∈ A do
3:     R_a ← ∅
4:     s ← Ω(V(P))
5:     R_a ← R_a ∪ Φ(P, s)
6: end for
7: return R
```

(a) Solution proposed for Problem 1.

```
1: R ← ∅
2: s ← Ω(V(P))
3: R ← R ∪ Φ(P, s)
4: return R
```

(b) Solution proposed for Problem 2.

Figure 3. Solutions for the problems proposed in the paper. Consider R a set of routes, R_a a subset of routes for a vehicle a, $s \in V(G)$ a source vertex for a graph P, where $P = (V, E)$. $\Phi : P \times V \to P$ is an operator that returns a path given a graph and a source vertex.

According to the algorithms in Figure 3, Φ can be any single-source routing algorithm in a directed graph. Some examples are breadth-first and depth-first searches, which detect all reachable vertexes from a single source [13]. There are also shortest path algorithms, such as Dijkstra's and Bellman-Ford's, which calculate the minimal paths from a single source to every reachable vertex of the same graph [15][18]. Heuristics, such as A*, are welcomed as well [16][17]. Notice that the latter algorithms require a cost function $C : E(G) \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers.

Another important topic is reducing large amounts of work into sub-problems, which allows us to avoid repetition and implement reusable design patterns [14]. Lemma 2 applies this vision to Problems 1 and 2.

Lemma 2: The problem of modelling an autonomous vehicle is a sub-problem of modelling an autonomous traffic system.

Proof: Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of n vehicles and $P = (V, E)$ a graph, where V is a set of positions and $E = (u, v)$, $u, v \in V$ a set of pairs that indicate that v is reachable through u. Consider the following assumptions:

- There is a set of known, but randomly calculated source vertexes $S = \{s_1, s_2, \ldots, s_n\}$, where $S \subseteq V$;
- The solutions of Problems 1 and 2 will always calculate the route for a vehicle a_i starting from the corresponding vertex s_i, where $s_i \in S$;
- The algorithm used to calculate the route R_i for a_i will be the same in both solutions.

Given a set of routes $R = \{R_1, R_2, \ldots, R_n\}$ calculated by the solution of Problem 1, we define R' a set of routes that will be calculated individually by all vehicles $a \in A$, as said by the definition of Problem 2. According to Lemma 2, if $R' = R$ we can prove that Problem 2 is a sub-instance of Problem 1.

Let's prove it by induction. The steps are:

1) Initially, $R' = \emptyset$
2) For all vehicles $a_i \in A$, apply the solution of Problem 2, generating a route R'_i;
3) Insert the solution R'_i to R';
4) R'_i was generated with the same algorithm as R_i and starting from the position s_i, therefore $R'_i = R_i$.

Since every $R'_i \in R'$ is equals to the corresponding $R_i \in R$, then the sets R' and R are equal. Therefore, we proved that applying the solution of Problem 2 to all vehicles in A will produce a valid solution for Problem 1, hence Problem 2 is a sub-instance of Problem 1. ■

Lemma 2 is useful because it tells us that, to model an autonomous traffic system, we only need to model a set of autonomous vehicles. Autonomous traffic is hence a composition of autonomous vehicles.

III. Development

This section will present how we applied the methodology in our work. In Section III-A, we describe the development environment and how we used it to simulate a multi-agent system. In Section III-B, we do a more detailed analysis on how to use the environment's tools to model the agents. Then, in Section III-C we describe which algorithms and data structures we designed in order to make the environment behave like an autonomous vehicular traffic system.

A. GTA-SA/SA-MP: an online multiplayer game server

The GTA-SA/SA-MP is a multiplayer modification for the Rockstar's game GTA-SA [8][9]. It uses the dependency inversion principle to make an interface between an abstract machine interpreter and the game itself [12].

The abstract machine, AMX, is the compilation result of a well-known scripting language, named Pawn [19]. SA-MP is an AMX interpreter that executes the compiled AMXs with the developers' modifications. Therefore, GTA-SA/SA-MP is a client-server application, where GTA-SA is the client and SA-MP is the server [9].

 ISBN: 978-1-61208-795-5

Also, Pawn interpreters follow a design pattern that provides a common interface to be extended in separated plug-ins. For instance, developers can write dynamic libraries that include the AMXs' interpreters base functions, allowing them to be extended and reused in Pawn scripts. Since dynamic libraries are developed in middle-level languages, such as C or C++, we can improve our server for better memory management and computational cost [9][12][19].

When a player first enters a SA-MP server, it will establish a connection between the game and the respective server, whose events, objects and other dynamics will be controlled and modified according to the compiled AMXs [9][20].

In this context, we created a set of Pawn scripts and dynamic libraries that used the game's built-in objects, physics and events to modify its default behavior, allowing us to introduce our own functions to simulate an autonomous vehicle system.

SA-MP has a consolidated community that writes and shares plug-ins and scripts around the world [20]. We used some of these, like the *Fully Controllable Non-Playable Character* plug-in, FCNPC, which allowed us to insert and control non-playable characters, NPCs, in the game map [21]. This plug-in was important to implement the agent system's behaviors, which can be seen in more details in Section III-B.

Also, based on the premise that games render only a limited amount of objects in the screen, mostly to save memory and computational resources, it was important to use plug-ins that embodied this principle and avoided extra memory consumption. Therefore, we used *SA-MP Streamer Plugin* to draw objects and labels in the game map, allowing us to display some structural information about the system [22].

In addition, SA-MP exports a set of paths' data that we could decode and use to implement our system [23]. Please follow to Section III-C to have more details on how we modified the game's path structures to implement routing algorithms.

Finally, we used the described tools and data structures to solve Problem 2, which can be scaled to a solution of Problem 1.

B. Autonomous agents modelling

GTA-SA/SA-MP provides a whole set of objects and structures that can be modified by FCNPC plug-in [20][21]. Among them, there are non-playable characters and vehicles. We used these elements to solve Problems 1 and 2, proposed in this work's methodology at Section II-C.

In this case, vehicles can be controlled either by the player or by an NPC. Since NPCs can have autonomous behavior, then we could simulate autonomous driving by inserting a NPC into a vehicle [21]. Therefore, we created a dynamic library that extends SA-MP server to do basic operations over NPCs and vehicles, given by Definition 2.

Definition 2: **The Driver NPC plug-in**. It is a dynamic library that implements SA-MP's model to manage NPCs directly inside a vehicle. This plug-in exports three functions:

- *Create*: receives as input the vehicle's attributes, as a pair of primary and secondary colors C_1 and C_2, a vehicle model T, an initial position s and an angle θ in relation to the north, which indicates the direction the vehicle will be facing. It renders the model in the described position and will return a unique identifier a, representing the NPC;
- *Destroy*: receives as input an NPC identifier a. It will destroy both NPC and its corresponding vehicle from the screen;
- *Move*: receives as input an NPC identifier a and a pair of positions u and v. It will make the NPC travel from u to v simulating an autonomous driving.

In Definition 2, the operations *create* and *destroy* were developed using GTA-SA/SA-MP's own features [20][24]. However, operation *move* needed to be adapted in different situations, like curves and hills. In curves, we calculated the Bezier's curve between the initial and final positions [25]. Driving up and down hills was still an obstacle, and we couldn't find a method to adjust the vehicle's angle while traversing different reliefs.

C. Routing algorithms and data structures

GTA-SA/SA-MP provided a set of path structures that allowed us to model positions and hence a directed graph in the game map [23]. The structures are described in Definition 3.

Definition 3: **SA-MP's links and nodes**. There are two main types of nodes exported by GTA-SA/SA-MP:

- *Path-nodes*: a structure containing an identifier i, an area identifier p, a link identifier l and a position $P = (x, y, z)$ in the game's 3D space. Path-nodes are placed in streets and roads along the map. Also, they can be extended in different kind of nodes:
  - *Navi-nodes*: a structure containing detailed information about a path-node. It has the referred path-node's identifier i and area identifier p, and a position $P = (x, y, z)$ in the game's 3D space, usually between two adjacent path-nodes. It also has a set of flags, like the amount of left and right lanes in relation to the navi-node and a value to indicates if the traffic flow is allowed in the right or left lanes;
  - *Ped-nodes*: describe the game's default paths for pedestrians. Usually placed on streets intersections, houses and sidewalks;
  - *Vehicle-nodes*: describe the game's default paths for vehicles. Usually placed on streets, roads and parking lots;
  - *Boat-nodes*: describe the game's default paths for boats. Usually placed at the sea.
- *Links*: a structure containing an identifier l and pairs of nodes and area identifiers $(i_1, p_1), (i_2, p_2)$, indicating that the nodes i_1 and i_2 in areas p_1 and p_2 have a connection.

Therefore, we created a dynamic library that extends SA-MP server to do operations over map positions, which allowed us to implement the proposed routing model. Hence, path-nodes and all their derivations could be inputted as positions; the *move* function, exported by the Driver NPC plug-in, as described in Section III-B, received a pair of path-nodes which represent the initial and final positions of the autonomous vehicle movement.

Definition 4: **The Paths plug-in**. It is a dynamic library that implements SA-MP's model to manage paths and positions in GTA-SA. It exports the following functions:

- *GetRelativeStreetPosition*: receives as input the identifiers i_1 and i_2 of source and target path-nodes and the

vehicle's width w, returning a position $P = (x, y, z)$ in the correct lane for the vehicle to traverse;

- *Dijkstra*: receives a pair of random path-node identifiers s and t as input, returning a list of path-nodes identifiers, representing the route with minimal cost between path-nodes s and t calculated by Dijkstra's algorithm [15];
- *AStar*: receives a pair of random path-node identifiers s and t as input, returning a list of path-nodes identifiers, representing the route with minimal cost between path-nodes s and t calculated by A*'s algorithm [16].

The Definition 4 states that the Paths plug-in exports a set of functions related to routing calculation and path-nodes manipulation. The function *GetRelativeStreetPosition* needs a pair of source-target nodes because, according to Definition 3, path-nodes identify a street rather than a specific lane. Therefore, we used both nodes' angles to identify in which track of circulation the vehicle was and, as a consequence, in which lane it should be placed. For that purpose, we also used the vehicle's width to calculate the agent's correct position in the lane, $P : A \to \mathbb{R}$, given by

$$P(a) = c + \frac{w(a)}{2}, \tag{2}$$

where $a \in A$ is a vehicle, $w : A \to \mathbb{R}$ is a function that returns the vehicle's width and c is a constant factor.

Also, this function was developed in a way that allows implementing both traffic ways, meaning that the agents can traverse in the positive lane direction as well as in the negative lane direction.

Finally, we developed the functions *Dijkstra* and *AStar* to calculate routes using Dijkstra's and A*'s algorithms, respectively, both returning a list of path-nodes, where the cost of navigating between a pair of path-nodes is given by the distance between two points in a 3D space, as defined in the equation (1) [15][16].

IV. Results

This section will explore the results we obtained after applying the proposed methodology to GTA-SA/SA-MP. We indicate how many path structures were found in the game, as well as their category according to Section III-A. Then, we clarify how they were used to generate routes in order to simulate the autonomous traffic. Finally, we raised time execution metrics to understand if the generated routes match the expected cost for the applied algorithms.

In our development, we were able to identify a high number of path-nodes in the game, as displayed by Table I. According to it, the pedestrian nodes represent the majority of GTA-SA paths data, being followed by vehicles and boats.

TABLE I. Quantity of path-nodes discovered in the game, grouped by their types.

Type	Count
Pedestrian	37,650
Vehicles	30,587
Boats	1,596
Total	69,833

From Table I, we can also estimate that GTA-SA's default features allow an autonomous agent network to be composed by 54% of humans, 44% of vehicles and 2% of boats, considering the default path-nodes distribution.

In addition, the path-nodes and navi-nodes were processed into a graph $\Gamma = (I, L)$, where I is a set of path-nodes and L is a set of links, containing information regarding the connection between the nodes in the game map. Figures 4 and 5 show a visual representation of path and navi-nodes, respectively.

In Figure 4, we see the detailed information of path-nodes, like the identifier $i = 36$, the area identifier $p = 261$, the three-dimension position $P = (x, y, z)$ and a set of links $L = \{(36, 262), (36, 265), (36, 270)\}$, where each pair (i', p') represents a target path-node's identifier and area identifier, respectively.

Figure 4. Visual representation of a path-node and its attributes.

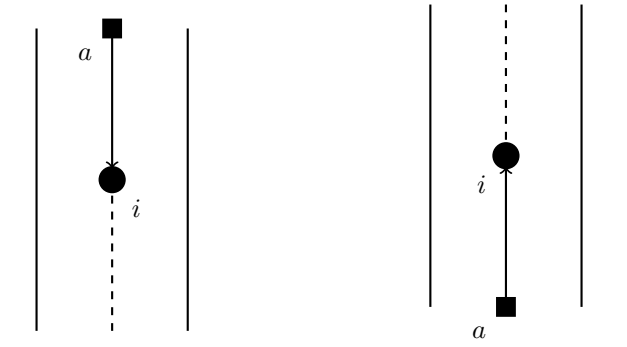
In Figure 5 we see detailed information about navi-nodes. They are mostly present on hills, curves and multi-lane streets or roads. We used their target path-node identifiers $i = 37$ and area identifier $p = 275$, as well as the amount of left and right lanes, which in the example counts as one for both.

Figure 5. Visual representation of a navi-node and its attributes.

Furthermore, we could categorize two different street models in the game: the single-lane and the multi-lane. The single-lane model has a single traffic way available or, more specifically, a way where navi-nodes expose zero left and right lanes. In the multi-lane model, there might be one or more traffic ways available for which the navi-nodes expose more than zero left and right lanes.

 ISBN: 978-1-61208-795-5

According to Figure 6, in the single-lane model the vehicle a can traverse in both traffic ways to reach a path-node i. This can be done by inverting all links' directions.

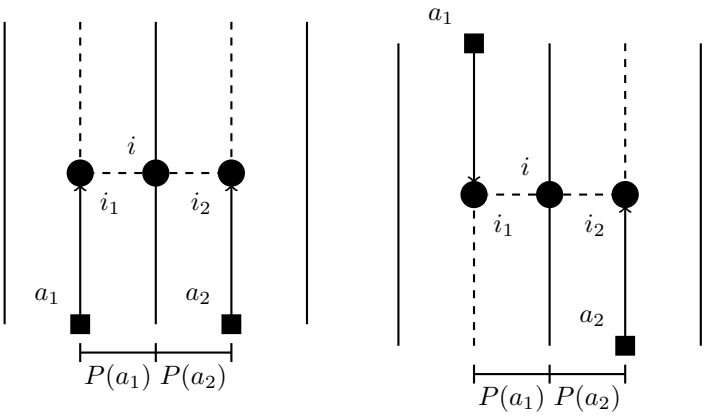


(a) North-south traversal in a single-lane street.

(b) South-north traversal in a single-lane street.

Figure 6. A single-lane street on which a vehicle a is reaching a path-node i.

According to Figure 7, in the multi-lane model vehicles a_1 and a_2 can traverse both in the same street to reach path-node i, but in different lanes. Therefore, we applied the equation in (2) to determine the correct vehicle's positions. Since we used the same vehicle models, w is a constant; hence, from empirical approximations, we determined $w = 0.75$ and $c = 1$.



(a) Multi-lane street model with a single direction.

(b) Multi-lane street model with different directions.

Figure 7. Multi-lane street models. Vehicles a_1 and a_2 need to reach path-node i, but it's placed in the middle of the street. Therefore, we calculate the displacements $P(a_1)$ and $P(a_2)$ to determine the correct next positions i_1 and i_2, respectively.

As for the routes calculations, we recorded the time spent, in milliseconds, for both Dijkstra's and A* algorithms according to the number of path-nodes that composed the routes. The results were compiled in the chart on Figure 8. According to the chart, we noticed that A* is faster when generating routes for the same set of path-nodes. In addition, both algorithms calculated the same routes with minimal costs, meaning that the vehicle's path did not take any influence in the route computation.

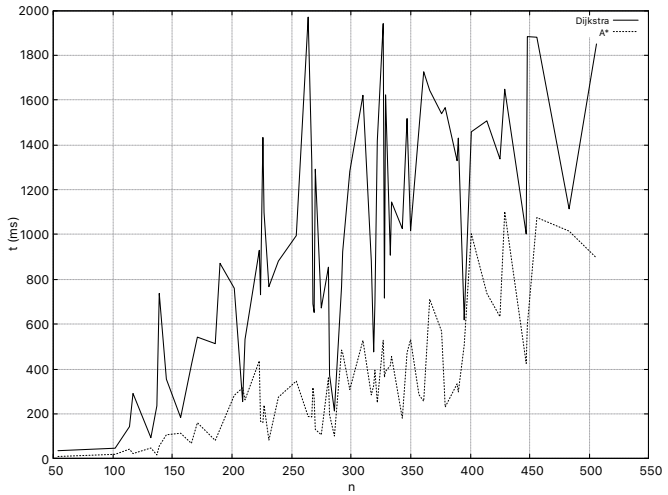


Figure 8. Time execution for the routing algorithms, in milliseconds (vertical axis), according to the number n of path-nodes (horizontal axis). The continuous line represents the time spent by Dijkstra's algorithm for shortest path, while the dotted one represents the time spent by A*'s algorithm.

The literature indicates that A* is faster than Dijkstra's algorithm whenever the heuristic is said to be consistent, which is also a good indicator that the adopted 3D space is consistent for routing models [16]. Also, we recorded a video of some simulations, which can be appreciated by the readers as a reference in this paper [26].

V. Conclusion

The results obtained in this work allowed a better simulation in the GTA-SA/SA-MP environment. When the autonomous vehicle network was extended to the whole map, it was possible to simulate different scenarios, taking advantage of many things that the environment offers, like different reliefs, street lengths and curve angles.

Furthermore, the results showed consistency with the reality, since A* and Dijkstra algorithms returned routes with minimal costs given the same set of path-nodes. Also, A* could calculate all routes with the proposed heuristic, which means that GTA-SA/SA-MP has a valid model of geometry and position.

This extension also allows the pedestrian and boat nodes to be implemented. Using the pedestrian nodes, it is possible to create an integration between autonomous vehicles and pedestrians in the roads, which is one of the main problems in autonomous vehicles systems. Using boat nodes, it is possible to simulate a maritime traffic, allowing to test the vehicles at sea as well.

Also, Autonomous Intersection Management protocols (AIM) can be implemented in a larger scale. Intersections are one of the main problems in autonomous vehicles systems too, since most of the traffic accidents happens in them. Therefore, implementing AIM at multiple intersections can be an efficient way to raise results about reductions in accidents and others factors, such as time spent, amount of emitted gases and spent fuel.

Another option would be increase the dynamics of the simulations, including weather manipulation, acceleration and deceleration in curves, uphills and downhills, reverse driving

 ISBN: 978-1-61208-795-5

and overtaking. This would allow us to explore more scenarios, as well as increase the difficulty of the problems we are solving.

Now, beyond the successful simulations that were done in the whole GTA-SA map, the progress made in this work opens doors to perform simulations of the main problems of autonomous vehicles systems in a larger scale, using the realistic environment GTA-SA/SA-MP.

ACKNOWLEDGMENT

A very special thanks to our institution, Centro Federal de Educação Tecnológica de Minas Gerais, for all support given during this work. Also, to Daniel de Sousa Santos, for bringing this wonderful environment to our team and allowing us to work and have fun.

REFERENCES

[1] D. Schrank, B. Eisele, and T. Lomax, "2019 Urban Mobility Report," Texas A&M Transportation Institute, 2019, URL: https://static.tti.tamu.edu/tti.tamu.edu/documents/mobility-report-2019.pdf [accessed: 2020-03-17].

[2] W. Bank, "The High Toll of Traffic Injuries: Unacceptable and Preventable," World Bank Group Transport, 2017, URL: https://openknowledge.worldbank.org/handle/10986/29129 [accessed: 2020-03-17].

[3] K. Zhanga and S. Batterman, "Air pollution and health risks due to vehicle traffic," Science of The Total Environment, vol. 450-451, 2013, pp. 307–316, ISSN: 0048-9697.

[4] R. E. Stern, S. Cui, M. L. D. Monache, R. Bhadani, M. Bunting, and M. Churchill et al., "Dissipation of stop-and-go waves via control of autonomous vehicles," Transportation Research Part C: Emerging Technologies, vol. 89, 2018, pp. 205–221, URL: https://www.sciencedirect.com/science/article/pii/S0968090X18301517.

[5] S. Park, H. Rakha, K. Ahn, and K. Moran, "Fuel economy impacts of manual, conventional cruise control, and predictive eco-cruise control driving," International Journal of Transportation Science and Technology, 2013, pp. p. 227–242.

[6] T.-C. Au, S. Zhang, and P. Stone, "AIM: Autonomous Intersection Management for Semi-Autonomous Vehicles," Handbook of Transportation, Routledge, Taylor & Francis Group, 2015.

[7] "AIM: Autonomous Intersection Management," 2006, URL: https://www.cs.utexas.edu/~aim/ [accessed: 2020-03-17].

[8] Rockstar, "Grand Theft Auto: San Andreas," 2004, URL: https://www.rockstargames.com/games/info/sanandreas [accessed: 2020-03-17].

[9] "San Andreas: Multiplayer," 2006, URL: https://www.sa-mp.com/ [accessed: 2020-03-17].

[10] R. R. Novaes Jr., "Um Novo Ambiente de Simulação para Sistemas de Gerenciamento de Tráfego para Veículos Autônomos," Centro Federal de Educação Tecnológica de Minas Gerais, Tech. Rep., 2017.

[11] R. R. Novaes Jr., D. S. Santos, G. M. F. Santiago, and S. R. Dias, "A New Solution to the Traffic Managing System for Autonomous Vehicles (Demonstration)," in 16th International Conference on Autonomous Agents and Multi Agent Systems, São Paulo/SP, Brasil, May 2017, pp. 1805–1807.

[12] R. C. Martin, "Design Principles and Design Patterns," 2000, URL: https://fi.ort.edu.uy/innovaportal/file/2032/1/design_principles.pdf [accessed: 2020-03-01].

[13] R. Diestel, Graph Theory. Springer-Verlag, 2000, URL: http://www.esi2.us.es/~mbilbao/pdffiles/DiestelGT.pdf [accessed: 2020-03-01].

[14] T. H. Cormen, C. E. Lelserson, and R. L. Rilvest, Introduction to Algorithms, 3rd ed. MIT Press, 2009.

[15] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, vol. 1, Jun. 1959, pp. 269–271.

[16] W. Zeng and R. L. Church, "Finding shortest paths on real road networks," 2009, URL: http://doi.org/10.1080/13658810801949850 [accessed in 2020-03-01].

[17] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding." J. Game Dev., vol. 1, no. 1, 2004, pp. 1–30.

[18] Walden and David, "The Bellman-Ford Algorithm and "Distributed Bellman-Ford"," Jan. 2008.

[19] CompuPhase, "Pawn Implementer's Guide," 2016, URL: https://github.com/compuphase/pawn/blob/master/doc/Pawn_Implementer_Guide.pdf [accessed: 2020-03-17].

[20] "SA-MP Wiki," 2017, URL: https://wiki.sa-mp.com/ [accessed: 2020-03-17].

[21] S. Marochkin, "SA-MP FCNPC Plugin," 2019, URL: https://github.com/ziggi/FCNPC [accessed: 2020-03-17].

[22] "SA-MP Streamer Plugin," 2014, URL: https://github.com/samp-incognito/samp-streamer-plugin [accessed: 2020-03-17].

[23] "Paths (GTA-SA)," SA-MP Wiki, URL: https://gta.fandom.com/wiki/Paths_(GTA_SA) [accessed: 2020-03-17].

[24] "Vehicle Models," SA-MP Wiki, 2019, URL: https://wiki.sa-mp.com/wiki/Vehicle_Models [accessed: 2020-03-17].

[25] J. London, New York, Ed., Numerical Control; Mathematics and Applications. J. Wiley, 1972, ISBN: 0471071951 9780471071952.

[26] R. R. Novaes Jr., P. C. Mesquita, and J. N. Cirqueira, "Autonomous traffic simulation with gta-sa/sa-mp," 2020, URL: https://bit.ly/3bfh0LN [accessed: 2020-03-20].

 ISBN: 978-1-61208-795-5