A Modular Architecture for Procedural Generation of Towns, Intersections and Scenarios for Testing Autonomous Vehicles

Conference Paper · October 2020

DOI: 10.1109/IV47402.2020.9304625

CITATIONS
16

READS
417

5 authors, including:

Ishaan Paranjape
University of California, Santa Cruz
4 PUBLICATIONS **22** CITATIONS

SEE PROFILE

Abdul Jawad
University of California, Santa Cruz
8 PUBLICATIONS **25** CITATIONS

SEE PROFILE

Jim Whitehead
University of California, Santa Cruz
83 PUBLICATIONS **1,650** CITATIONS

SEE PROFILE

2020 IEEE Intelligent Vehicles Symposium (IV)
October 20-23, 2020. Las Vegas, USA

A Modular Architecture for Procedural Generation of Towns, Intersections and Scenarios for Testing Autonomous Vehicles

Ishaan Paranjape*, Abdul Jawad†, Yanwen Xu‡, Asiiah Song§ and Jim Whitehead¶

Dept. of Computational Media
Univ. of California, Santa Cruz
Santa Cruz, CA USA

* iparanja@ucsc.edu, † abjawad@ucsc.edu, ‡ yxu83@ucsc.edu, § julinas@ucsc.edu, ¶ ejw@ucsc.edu

***Abstract*— Simulation-based testing is critical for ensuring safety of autonomous vehicles. Autonomous vehicles are enabled by deep learning techniques which require a large quantity of data. With simulation testing, we can create rare events for testing and training of autonomous vehicles. Procedural generation of roads and modeling of driving behaviors in an easily extendable architecture ensures that we are able to create rare scenarios at scale with minimal artistic burden. In this paper, we present CruzWay, a system that both supports and creates these scenarios. With CruzWay, we are able to procedurally generate town sized road networks or road intersections. CruzWay supports generation of road meshes as well as navigation meshes from SUMO road network files. CruzWay can generate cars as well as pedestrians run by behavior trees (BTs) in this environment. The self-contained, modular nature of BTs in combination with procedural roads allows us to create a large number of scenarios.**

I. Introduction

Autonomous, self-driving vehicles promise to reduce the tedium of driving, open up vehicle mobility for many people, reduce accidents, and automate many delivery tasks. Safety of autonomous vehicles is important to achieve these goals, since widespread adoption cannot occur if the technology is dangerous. Testing of autonomous vehicles must involve driving physical vehicles on real-world roads. But, such testing has many drawbacks. It is expensive. Many driving miles are commonplace and do not explore interesting aspects of the software. It is not possible to explore many dangerous situations.

Simulation-based testing of autonomous vehicles addresses these concerns. Simulated driving can focus on challenging situations, is less expensive per mile, and can result in orders of magnitude more miles of driving than physical tests. *Waymo* (a subsidiary of Alphabet) is an autonomous vehicle company which uses a virtual environment called Carcraft to supplement real-world vehicle testing. As of October 2018, Waymo had logged 10 million miles in the real world, as compared to almost 7 billion miles in simulation [1] [2]. Simulation-based testing is increasingly common in industry, with Amazon-backed autonomous vehicle company *Aurora* using an environment called Virtual Testing Suite, and GM's *Cruise* using one called the Matrix [3].

A defining characteristic of simulated testing scenarios is their variety. In order to fully test autonomous vehicles, scenarios need to explore a wide range of roads, traffic conditions, driving behaviors, vehicle types, pedestrians, bicyclists, road signs, lighting, weather, signals, lane markings, and more. Any environment designed to support simulation testing of autonomous vehicles should aim for covering a wide range of these axes of variability.

Supporting a wide variety of road and intersection types is especially important. In a recent fatal accident involving a Tesla Model X on a major US highway in 2018, one of several contributing factors was the inability of the Autopilot software to correctly detect lanes during a left exit scenario, or to detect the presence of a crash attenuator in front of a concrete barrier [4]. Procedurally generated roads, where an algorithm is responsible for automatically creating a synthetic road network or intersection, can create a near infinite number of different road configurations. This supports a wide variety of testing scenarios, and also permits modeling of a wide range of road situations in many countries. In a similar vein, support for a wide variety of driving behaviors for the "NPC cars" (non-player character cars, or cars that are not controlled by a full autonomous vehicle software stack) in a scenario also permits modeling a wide variety of driving situations in many different locations.

We present *CruzWay*, a system that both supports and creates autonomous vehicle testing scenarios. Notable features of CruzWay include its support for procedurally generated towns and intersections, and support for behavior tree (BT) driven NPC car and pedestrian behavior models. Roads are based on the road format used by the open source SUMO traffic simulator, and hence CruzWay offers the potential for integrating SUMO traffic simulations into autonomous vehicle test scenarios. As compared to existing environments that support autonomous vehicle simulation testing, CruzWay is distinguished by its emphasis on procedural road generation, the self-contained nature of the BTs which supports specialization of behavior, and its overall modularity.

In the remainder of the paper, we compare with related work, describe the system architecture in detail, and provide examples of supported road and NPC car behaviors.

978-1-7281-6673-5/20/$31.00 ©2020 IEEE

II. Related Work

There are many simulators for virtual testing of autonomous vehicles (AV). Simulation products such as PreScan [5], CarSim [6], Autoware [7], and rFpro [8] allow engineers to simulate sensors, conditions, mechanical builds, and mechanical decay. Game engine based simulators such as CARLA [9], Apollo [10], Nvidia DRIVE [11], LGSVL [12], and AirSim [13] are run with high-fidelity graphics and real-time physics engines which helps developers model collisions and agent dynamics.

Our work on CruzWay falls in this space of game engine based simulations. The main advantage of game engine based simulators is not only their support for graphics and collision detection but also the ability to integrate the AV stack with accurate sensor models. By using these tools and features, one can manipulate textures, character animations, and lighting to get a photorealistic simulation that makes the game engine based simulators an ideal platform for testing and validating perception, planning and other aspects of AV. We briefly review CARLA, Apollo, Deepdrive, NVIDIA DRIVE Constellation, AirSim, and LGSVL as they are closest to our work. They use hand authored maps and test scenarios, generally published at regular intervals. Our work is different from these existing simulators in their capability to generate realistic road structure and test scenarios procedurally. Although hand authored maps and test scenarios give us a massive number of possibilities to test and train the AV, it lacks the ability to generate emergent [14] behaviors from the vehicles. Moreover, due to the lack of widely adopted standards, most of these possible scenarios are not transferable between platforms. In the following paragraph, we will discuss these simulators' abilities and features to create/import maps and scenarios.

CARLA has a library of 3D models for vehicles, pedestrians, buildings and road signs, and supports ingesting maps using OpenDRIVE format files [15]. Scenarios in Carla can be defined by using a Python API or by the OpenSCENARIO standard and they provide some pre-authored scenarios to test the submissions for their CARLA challenge [16].

Apollo is a Unity-based AV simulation platform that also contains *Dreamland* [10] - a web-based simulation platform. Its main features are a list of two hundred scenarios, the ability to run simulations on the cloud, and an automatic grading system which consists of metrics to assess autonomous driving.

Deepdrive [17] provides three diverse maps and a traffic AI that can overtake other agents and negotiate intersections intelligently. Deepdrive offers five pre-authored scenarios and a Python API to access all game engine features. There is a version of Deepdrive available for the Unreal Engine.

AirSim provides 12 kilometers of virtual roads with 20 city blocks and APIs to retrieve data and control vehicles. The APIs are accessible via a variety of programming languages. AirSim does not provide any pre-authored scenarios but supports hardware-in-the-loop with driving wheels for physically and visually realistic simulations.

NVIDIA DRIVE Constellation [11] is a cloud-based virtual reality simulation platform that enables hardware-in-the-loop testing. It comes with a 3D map and detailed assets along with the capability to author traffic behavior at a micro and macro [18] level. It is built using the Unreal Engine. However, it is not open-sourced.

LGSVL [12] provides seven different maps to test the AV stack. It uses predefined seed for deterministically controlling and spawning NPC vehicles and pedestrians. It is built with the Unity game engine.

In CruzWay, we are able to generate complex roads with stop signs, detailed textures, splines and navigation meshes procedurally. This implies that we can uniquely create high variation in road structures with minimal artistic burden. Other simulators lack these procedural generation capabilities, with the CARLA simulator rapidly building some of these up in the form of the *OpenDrive standalone mode* in version 0.9.8 [15]. However, creating customized road structures and navigation meshes still needs other applications and may not be procedural. Another feature in CruzWay is BT controlled vehicles and pedestrians. Uniquely, each agent here is controlled by its own BT. This provides the agents and consequently the scenario emergent properties as the agent behavior descriptions and assignments become more complex. Other simulators use authored event or maneuver based scenarios where agents aren't able to dynamically adapt to changing environments. However, BTs are supported in CARLA's scenario definition and execution engine, Scenario Runner [16] which uses a single BT to define non ego-vehicle behavior.

Table 1 summarizes a comparison of popular game engine based simulation platforms across various metrics such as their ability to create diverse simulation scenarios, whether they are open-sourced and the ease with which they can be integrated with AV software such as Autoware or Apollo.

III. System Architecture

The main requirement for the CruzWay system is the capability of generating roads and intersections procedurally. That is, we wish to have computer algorithms generate novel, synthetic roads and intersections. This accommodates a large range of road geometries and consequentially a wide variety of lane markings and textures. Another requirement is the ability to support agents in the simulation which can be controlled using BTs. BTs allow us to design NPC (non-player character) agents, such as vehicles and pedestrians, with a wide variety of complex behaviors. There is also a need to support an ego vehicle agent which is controlled using a complete AV software stack in this simulation environment. This would be our unit under test (UUT).

To minimize redundant work with respect to other, relevant open source projects and to enable support for other platforms such as SUMO, Apollo and CARLA, CruzWay is designed to be highly modular. We generate road data separately in a file and our road generation and NPC simulations are designed as plugins for the Unreal Engine.

Simulation environment metrics	Carla	Apollo	Deepdrive	AirSim	LGSVL	NVIDIA	CruzWay
Diverse, emergent scenarios	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Possible
Access	Open	Open	Open	Open	Open	Commercial	Open
Platform	Unreal	Web viz.	Unreal	Unreal	Unity	Unity	Unreal
Procedurally generated roads/intersections	No	No	No	No	No	No	Yes

TABLE I: A comparison of AV testing simulators.

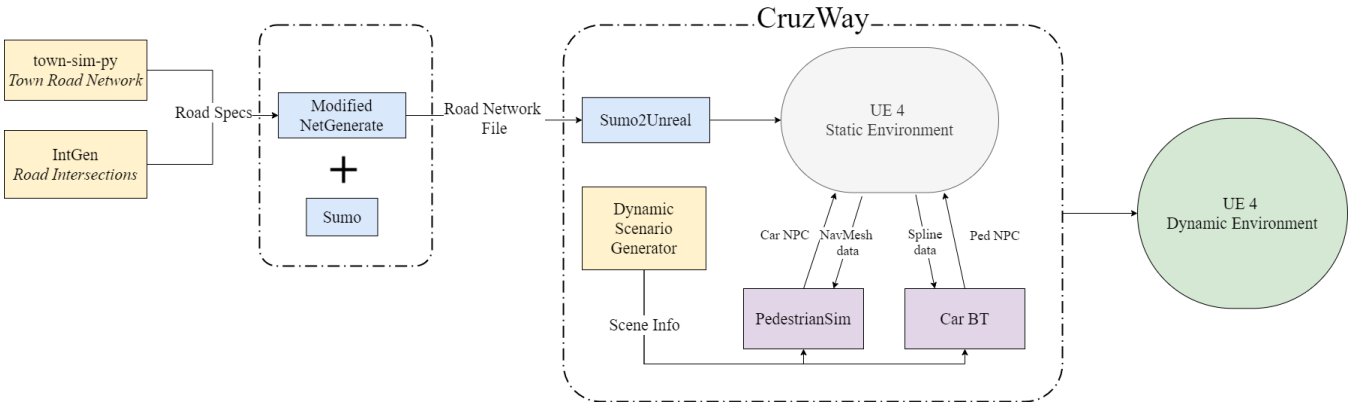

Fig. 1: The system architecture of CruzWay. Standalone applications are shown in yellow-colored boxes. Applications that are a part of a larger environment for road network generation are shown in blue, and dynamic behavior generation applications are shown in purple.

The modules of CruzWay which are dynamic (where their positions change with respect to time) are controlled with the simulation scenario description. CruzWay supports a simple scenario description in a vector form specifying the initial configurations of agents in the simulation. These scenarios are for testing components of the AV stack such as path planning, behavior prediction and perception. The final result of the system is a complex road network/intersection and agents with complex behaviors. Since the road geometries and NPC behaviors are procedurally generated, CruzWay can create and run a large variety of simulation scenarios for testing AV. A diagram of the complete system architecture is shown in Figure 1.

The system works as follows: A hierarchical road network specification file is generated using either *TownSimPy* or *IntGen* applications for the purpose of generating town scale road networks or individual road intersections respectively. The road network file has a high-level description of roads, including road length and connectivity information but lacking exact shape descriptions of road and intersection polygons. This specification is then read by a customized *Netgenerate* application (which belongs to the SUMO traffic simulator's suite of applications) which generates a much more detailed road network file in the standard SUMO road network format. This file describes coordinates of the road, intersection polygons, crosswalks, sidewalks, etc. This file is then imported and converted into a geometry of road meshes into the Unreal Engine using the *Sumo2Unreal* application. A second stage adds procedural navigation meshes (navmeshes). Using this road and navmesh information, other agents (such as NPC cars and pedestrians) are procedurally placed on top of these meshes according to a scenario description. The medium of communication of road data to vehicles and pedestrians in the simulation is through splines and navmeshes respectively. With CruzWay, we can run a wide variety of procedural simulation scenarios to capture many of the situations which an AV would encounter in the real world.

Below we describe the components of CruzWay in detail.

A. Road Generators: TownSim and IntGen

TownSim [19] is an agent based city evolution algorithm that generates road networks. It procedurally generates town-sized road networks with different road sizes that vary from grid-like sections to organic, dendritic sections. Figure 2 shows an example of a road network generated using TownSim and imported via Sumo2Unreal.

IntGen [20] is a road intersection generator with the capability to create a detailed specification file of a road intersection. The inputs to this application are a number of command-line parameters detailing the intersection information as well as details of individual roads. The output is a road specification file represented in JSON format which can be read by *Netgenerate*. It details the high-level intersection information in the form of the Cartesian coordinates of the intersection, the number of incoming roads for it and the presence of pedestrian crosswalks. This data also specifies details of each incoming road, including the length (in meters), start and end points, the number of lanes (and turn lanes) and the presence of sidewalks. Having these many parameters allows this application to express a large range

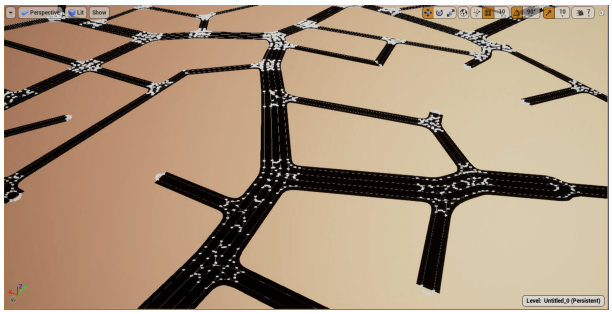

Fig. 2: A town sized road network generated using TownSim, imported using Sumo2Unreal, and rendered on Unreal Engine.

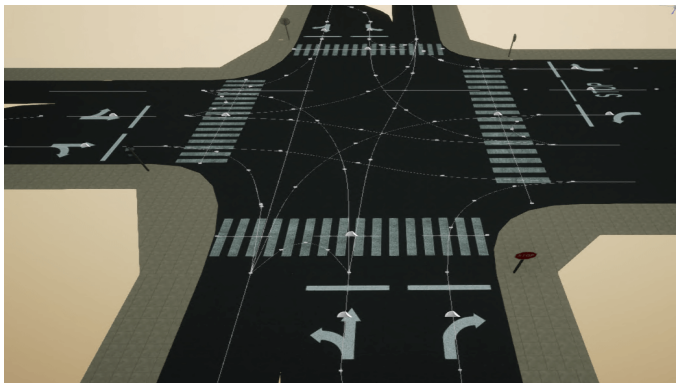

Fig. 3: A four-way intersection generated using IntGen. All of the incoming roads have one turn lane and one of the incoming roads has four lanes, while the remaining default to two lanes each. Note thin lines, which are splines used by NPC cars.

of road intersection types. Consequently, this would allow an even larger space of scenarios. Examples of four-way and five-way intersections generated by IntGen are shown in Figures 3 and 4.

B. Enhanced Netgenerate

Netgenerate is a road network generation tool within the SUMO traffic simulator suite of applications. Netgenerate typically has the capability to generate synthetic grid networks, spider networks and random networks. For CruzWay, we extended Netgenerate [21] to accept the JSON formatted road specification files created by *IntGen* or *TownSimPy*. The output of Netgenerate is a road network XML file in the SUMO road network format. This XML file contains much more detail regarding individual roads and sidewalks. By extending Netgenerate, we can generate a wider variation of SUMO road networks than the three mentioned above. We can then use other SUMO applications such as *Netconvert* which can convert SUMO road networks to other standard road network formats such as OpenDrive and MATsim. This pipelining allows us to generate in multiple road network formats thereby allowing us to generate roads for multiple platforms. It also permits running traffic simulations using SUMO. We can then import this road network into the

Fig. 4: A five-way intersection generated using IntGen.

Unreal Engine as procedurally generated road meshes by using *Sumo2Unreal*.

C. Sumo2Unreal

Sumo2Unreal [22] is an Unreal Engine plugin for importing a SUMO road network file into the Unreal Engine as procedural road meshes. Unreal Engine is a widely used game engine that supports the creation of real-time software with high-fidelity graphics. Procedural road meshes generated include driving lanes, sidewalks and pedestrian crossings. Using Unreal Engine provides us with many useful capabilities, such as built-in 3D models such as cars, an API for creating BTs, along with photorealistic graphics such as a variety road textures. Sumo2Unreal generates stop signs for road networks describing road intersections only (such as the ones from IntGen). Sumo2Unreal also generates splines from a collection of SUMO road way points. These splines can be used by BT controlled cars to to follow logical lanes and make turns correctly. Splines also contain other information, such as other splines connected to them (in the driving direction) and whether there is a stop sign at the end of the spline segment. Together, the splines and the navigation meshes are able to communicate the map data to cars and pedestrians in the simulation.

The movements of pedestrians are more fine-grained than vehicles. The action space includes walking, running, jumping, to any combination of the above. A *Navigation Mesh* (NavMesh) is a path-finding data structure that guides the agent to navigate through complicated spaces. Compared to splines, NavMeshes can model more complex motion behaviors and supports crowd simulation. Specifically, CruzWay uses NavMeshes to define the potential walkable areas for pedestrians in the scene. Walkable areas include sidewalks, roads, and all areas a pedestrian can step onto. This enables the scenario generator to produce both standard test cases and edge cases where the pedestrians are jaywalking.

Areas in the NavMesh can have attributes such as cost and constraints using this cost attribute. In CruzWay, undesirable locations are marked with higher costs. When determining the shortest path from one waypoint to the next, an NPC

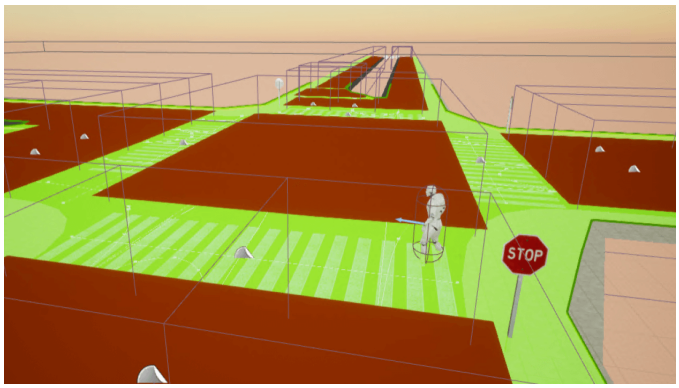

Fig. 5: Procedurally generated navigation bounds volume (green, normal cost) and navigation modifiers (red, high cost) on the four-way intersection. Pedestrians tend to navigate across green areas.

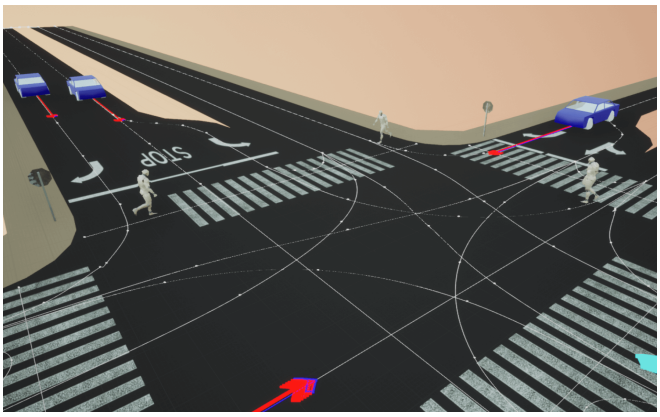

Fig. 6: Multiple BT-car and multiple BT-pedestrian agents at an intersection.

pedestrian tends to avoid higher-cost areas, such as the road. However, pedestrians can still cross higher-cost areas if the path is the shortest. Unreal achieves this with the NavMesh Modifier Volume component, which is an explicitly annotated volume with properties such as positions and dimensions to mark different areas. We procedurally generate Modifier Volumes along with NavMesh Bounds Volumes on the scene, as shown in Figure 5.

Hence, with CruzWay, we are able to generate complex roads inside a game engine with stop signs, detailed textures, splines and navigation meshes procedurally. This implies that we can generate high variation in road structure with minimal artistic burden.

D. Behavioral Agents

To create diverse simulation scenarios, CruzWay employs Behavior Trees (BTs) [23], a popular technique used in video games to author NPC behaviors. BTs support an execution approach that involves sensing the world, determining one or more actions based on the state of the world, then enacting those actions. Basic actions are arranged in a tree like structure, with basic actions as leaves and containers as internal nodes. Containers have varying execution semantics, including: executing a series of actions (sequence), trying actions until one succeeds (fallback), or trying actions in parallel (parallel). Figure 7 shows an example BT.

BTs have many beneficial qualities that led to our adoption in CruzWay, including modularity, reusability, controllability, and the ability to iteratively add complexity to an NPC car or pedestrian. Unreal also has a designer-friendly user interface for creating BTs. Unreal's implementation of BTs uses an event-driven approach (as compared to the traditional tick-based implementation), which leads to lower processor use, and permits better scalability [24]. CruzWay has successfully run over 75 simultaneous BT NPC vehicles in real-time.

Cruzway includes two types of BT controlled agents: vehicles (BT-vehicle) and pedestrians (BT-pedestrian), which is shown in Figure 6.

1) BT-vehicle: All of the BT controlled NPC vehicles receive information about the road through the spline data structure. Splines contain information about other splines connected to the current spline, stop sign locations, and speed limit signs. NPC vehicles navigate through the environment using this information. We define a *driver* for an NPC vehicle as a set of threshold values, for example, *ThresholdDistanceStopSign* determines when a driver will start the maneuver *StopAtStopSign*. These threshold values are stored in the blackboard along with other information necessary for the BT to run. By setting different threshold values, driver behaviors can be altered. For example, more aggressive behavior can be formed by decreasing the *ThresholdDistanceStopSign* and increasing the *ThresholdBrakeValue*

The behavior of our NPC cars is shown in Figure 7. In general, the car accelerates up to a preset velocity value and then maintains that velocity. The selector node looks for conditions that might change this default behavior. The first child node of each Sequence node is a precondition check. In this tree, the cars check to see if it is nearing a stop sign or is reaching the end of a spline. If so, the node to its right will execute, causing either a stop at the stop sign or changing a spline. According to the current implementation, NPC cars select a random spline from the connected splines when a *spline change behavior* happens. Since BTs are modular, future BTs (such as for a bicycle) can also make use of the same spline changing behavior. By procedurally generating BT and changing the driver behavior thresholds, a range of behaviors and scenarios can be created.

2) BT-pedestrian: All BT controlled NPC pedestrians perceive road and sidewalk information through the generated navigation meshes. Unreal pre-computes NavMesh data structures offline and can update them dynamically at run-time. This enables pedestrian movement components to bypass obstacles or other pedestrian agents and re-adjust routes in real-time. The simplest NPC pedestrian model selects an arbitrary random location on the sidewalk area and navigates to the destination using the single source shortest path algorithm. All pedestrian agents share the same NavMesh, supporting multi-agent path-finding (Fig. 8).

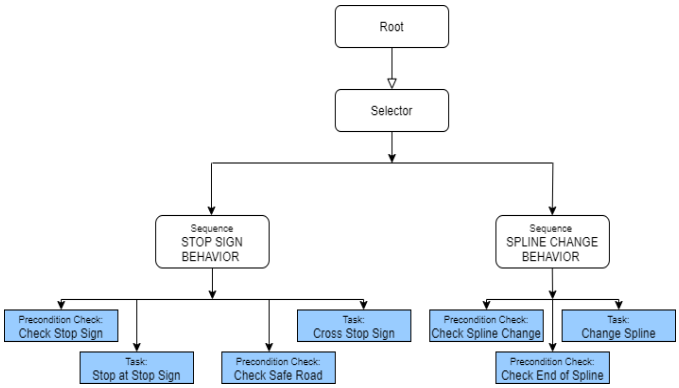

Fig. 7: A vehicle BT containing precondition nodes and task nodes. The car will stop at the stop sign if it detects a stop sign. Otherwise, it will continue driving and change the lane if it has reached the end of the spline.

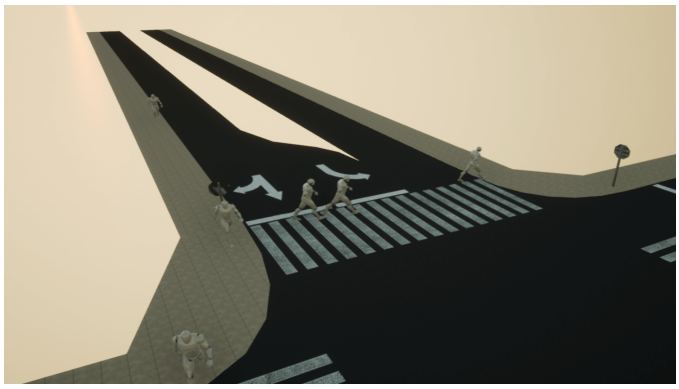

Fig. 8: Multiple BT-driven pedestrian agents walking across a crosswalk.

E. Scenario Generation

CruzWay has the capability to create scenarios consisting of BT vehicles and pedestrians. Vehicles and pedestrians can be spawned at different times during the scenario. At spawn time, vehicles are initialized with an initial spline position. Multiple cars can be spawned on a single spline at different non-colliding spline distance. Vehicles are not limited to being spawned at specific spawn points defined on the map (as is the case in other environments, such as CARLA). Pedestrians are spawned at a specific location within a low-cost navmesh region, with a randomly chosen destination. In the current implementation of CruzWay, we can create scenarios with multiple cars that follow stop sign rules, follow lane maintenance rules and safely accelerate when the road ahead is safe. Scenarios can support multiple pedestrians with varied destination points. Simulation parameters are represented in a vector format, which is well suited for reasoning over scenario execution runs.

Though not currently implemented, our simulation architecture is well suited to support scenarios containing vehicles with varying behaviors. Due to the modularity of BTs, different vehicle behaviors can easily be assembled, and multiple cars with varying behaviors can be executing simultaneously. Similarly, BTs can support multiple simultaneous pedestrian models. Scenarios contain a wide variety of behaviors, leading to greater diversity in the simulation, thereby modeling a wide range of real-world situations.

IV. Conclusion

CruzWay is a simulation environment for creating scenarios to test autonomous vehicles. As compared to other similar environments, CruzWay supports procedurally generated roads and intersections, thereby supporting a much wider range of road geometry. CruzWay supports procedural road mesh and navmesh generation based on SUMO road network files. BT driven vehicles and pedestrians allow for scenarios to be populated with intelligent agents comprised of modular behaviors. In combination, these capabilities support the generation of a broad range of scenarios involving vehicles and pedestrians navigating through procedurally generated worlds. CruzWay's modular architecture allows for subcomponents to be reused in other contexts and environments.

Acknowledgment

This work was supported by a URP award from Ford Motor Corporation. Jelica Hipolito created the models and textures for the road surface, lane markings, sidewalks, and road signs.

References

[1] M. DeBord, "Waymo just crossed 10 million self-driving miles – but the company has a secret weapon that gives it even more of an edge," https://www.businessinsider.com/waymo-self-driving-cars-secret-weapon-is-simulation-testing-2018-10, 2018, web page, accessed 17 April 2019.

[2] A. C. Madrigal, "Inside Waymo's secret world for training self-driving cars," https://www.theatlantic.com/technology/archive/2017/08/inside-waymos-secret-testing-and-simulation-facilities/537648/, 2018, web page, accessed 17 April 2019.

[3] K. Wiggers, "Aurora urges autonomous vehicle industry to adopt better safety metrics," https://venturebeat.com/2020/01/24/aurora-urges-autonomous-vehicle-industry-to-adopt-better-safety-metrics/, 2020, web page, accessed 9 March 2020.

[4] National Transportation Safety Board, "Collision between a sport utility vehicle operating with partial driving automation and a crash attenuator," https://www.ntsb.gov/news/events/Documents/2020-HWY18FH011-BMG-abstract.pdf, Meeting minutes, Feb. 25, 2020.

[5] TASS International. (2020) Prescan: Simulation of adas & active safety. https://tass.plm.automation.siemens.com/prescan. Web page, accessed 12 March 2020.

[6] Mechanical Simulation Corporation. (2020) CarSIM. https://www.carsim.com/. Web page, accessed 12 March 2020.

[7] Autoware. (2020) Autoware.AI. https://www.autoware.ai/. Web page, accessed 12 March 2020.

[8] rFpro. (2020) rfpro: Simulation software. http://www.rfpro.com/about/. Web page, accessed 12 March 2020.

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[10] ApolloAuto. (2019) Apollo Dreamland. https://github.com/ApolloAuto/apollo/blob/master/docs/specs/Dreamland_introduction.md. Web page, accessed 12 March 2020.

[11] NVIDIA, "NVIDIA Drive - autonomous vehicle development platforms," https://developer.nvidia.com/drive, 2020, web page, accessed 10 March 2020.

[12] LGSVL Simulator, "LGSVL simulator overview," https://www.lgsvlsimulator.com/downloads/ LGSVL-Simulator-Overview.pdf, 2020, web page, accessed 10 March 2020.

[13] Microsoft. (2020) Airsim. https://microsoft.github.io/AirSim/. Web page, accessed 12 March 2020.
[14] Wikipedia. (2020) Emergence. https://en.wikipedia.org/wiki/Emergence. Web page, accessed 12 March 2020.
[15] CARLA. Carla 0.9.8 release. http://carla.org/2020/03/09/release-0.9.8/. Web page, accessed 14 May 2020.
[16] CARLA. ScenarioRunner for CARLA. https://github.com/carla-simulator/scenario_runner. Web page, accessed 12 March 2020.
[17] Deepdrive Voyage. Build self-driving ai. https://deepdrive.voyage.auto/. Web page, accessed 12 March 2020.
[18] H. Abouaissa, D. Jolly, A. Benasser *et al.*, "Macro-micro simulation of traffic flow," *IFAC Proceedings Volumes*, vol. 39, no. 3, pp. 351–356, 2006.
[19] A. Song and J. Whitehead, "Townsim: agent-based city evolution for naturalistic road network generation," in *Proc. 10th Workshop on Procedural Content Generation (PCG 2019)*, 2019, pp. 1–9.
[20] I. Paranjape. (2019) IntGen. https://github.com/AugmentedDesignLab/intgen. Web page, accessed 12 March 2020.
[21] ——. (2019) Modified Netgenerate. https://github.com/AugmentedDesignLab/sumo-mirror/tree/sumoIntgen. Web page, accessed 14 May 2020.
[22] ——. (2019) Sumo2Unreal. https://github.com/AugmentedDesignLab/Sumo2Unreal. Web page, accessed 12 March 2020.
[23] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2019.
[24] Unreal Engine. (2020) Behavior tree overview. https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html. Web page, accessed 12 March 2020.