CS 492 Senior Design Project
Final Report

DriveSim

Group Members
Bora Ecer
Oğuz Kaan Ağaç
Alp Ege Baştürk
Alptuğ Albayrak
Alperen Erkek

Supervisor:
Prof. Uğur Güdükbay

Jury:
Prof. İbrahim Körpeoğlu
Prof. Özgür Ulusoy

Innovation Expert:
Cem Çimenbiçer (TaleWorlds Entertainment)

May 9, 2019

1 Introduction

DriveSim is a simulation which aims to train and improve skills of drivers in a safe and controlled environment. It features rules for traffic lights, pedestrians, other cars and the lanes. In addition to training mode, which allows a user to train themselves using keyboard or an external wheel and pedals, there is a machine learning part which learns how to drive. This works by ray casting from specific sensors around the car to provide environment information to the vehicle. DriveSim uses Unreal Engine 4, which provides basic engine functionality and assets, and it was implemented using C++.

The training mode starts with a car placed in a randomly created city environment which is built from simpler blocks. The driver controls a car either with keyboards keys or wheel and pedals. User is allowed to drive on the current lane, opposing lane or on sidewalks. Driving on the road is the expected use so it does not cause any direct penalties however driving on the sidewalks results in point loss. Crashing into other vehicles, pedestrians or not obeying the lights cause great deduction in points. User is shown an indicator showing the distance to the target point which will be the positive ending condition, also a minimap is presented for a better experience, however user should see the environment in order to take actions against dynamic changes like pedestrians or lights.

The purpose of this machine learning mode is to visualize an driverless car to the trainee so that she/he can observe how the computer drives the car and learn.

2 Final Architecture and Design

2.1 System Overview

We have tried to separate rendering on the engine part, logic of the objects in the software, UI and machine learning. For example, base classes were implemented as C++ code, then engine specific Blueprints were created based on these. This was mostly necessary due to planned use of custom ECS. However it became unimportant since we have not used our custom ECS because of practical reasons. However UE4 provides a component system to a degree which was used internally. Blueprints which are used to render the objects were separated from the ones which provide logic as much as possible to make the project more maintainable. These separation allowed UI, ML, game logic and objects to be developed separately and merged at a later stage.

ML part uses the NEAT algorithm. It tries to find best candidates by evolving trained models in generations [1]. Training part was separated into a simpler map because of performance considerations, however it can be adapted to different maps.

2.2 System Architecture

Class Name: VehicleAI
VehicleAI is the class whose instances are the dynamic car objects that roam through the roads in the simulation mode.
Properties
-AI Target: Character -OnGoingTarget: Character -VehicleHitBox: Box

-VehicleReceiveBox: Box
Methods
- void stopMoving() //Stops the movement of the VehicleAI, used in cases of red lights and collusion with other VehicleAI instances. - void setCurrentTarget(Character target) //Sets the target of the VehicleAI the given parameter so that the instance moves towards that target.

Class Name: TargetBP
TargetBP is the class whose instances represent the point that which the VehicleAI instances move towards.
Properties
-TargetBox: Box -NextTarget: Character
Methods
-BeginCollusion() // when an instance of VehicleAI begins colluding with TargetBox, the SetNextTarget() function VehicleAI is called, and next target is changed.

Class Name: VehicleAIController
Performs UE4 AI Controller functions, and initiates the movement of the controlled VehicleAI pawns to the given TargetBP instance.
Properties
Methods
-MoveToTarget(Character Target) // gets the controlled pawn and calls setNextTarget() function and passes the target.

Class Name UIScreen<T>

Draws visuals loaded into its buffer. These visuals can be:
-Map: Holds visuals belonging to each map
-Options Screen: Holds visuals related to options.
-Main Menu: Holds visuals related to main menu. In this UIScreen, there are 3 methods:
*void exitGame() // Triggers default exit pathway
*void playGame() // Loads level and initialise belonging game logic. Loads visuals as well
*void operation() // Loads Neural-Network simulation
-Gameplay Menu Screen: Holds visuals related to game level selection. It has 2 integer variables called gameMode and gameLevel which holds information about current game state. It has getter and setter methods for these variables
-Design Selection: Holds visuals related to vehicle design screen

Properties
vector<BufferedImage> screen_visuals // holds current screen's visuals
Methods
-vector<BufferedImage> getVisuals() // returns current screen's visuals

Class Name BotCharacter
This is the basic humanoid character used as the base of pedestrians. It can be controlled by AI controllers to give different controls.
Properties
-expiryTimer // Timer to remove pedestrian after its functionality ends
Methods

-virtual void BeginPlay();
-virtual void Tick(float DeltaTime);
-void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent);
-void OnOverlapBegin(UPrimitiveComponent* OverlappedComp,
AActor* OtherActor, UPrimitiveComponent* OtherComp,
int32 OtherBodyIndex, bool bFromSweep,
const FHitResult& SweepResult); // Function called when overlap event begins
-void OnOverlapEnd(class UPrimitiveComponent* OverlappedComp,
class AActor* OtherActor, class UPrimitiveComponent* OtherComp,
int32 OtherBodyIndex); // Function called when overlap event ends

-void CollidedWithVehicle(); // Function called when collision is detected

Class Name BotCharacterAIController
Basic AI controller which controls the pedestrian according to specified logic.
Properties
-APawn* myPawn; // Reference to controlled bot
Methods
-virtual void BeginPlay(); -virtual void Possess(APawn* pawn); -virtual void Tick(float DeltaSeconds); -virtual FRotator GetControlRotation() const;

Class Name PedestrianSpawnPoint
Placeable object derived from Unreal Target asset, which adds new pedestrians from its location. Also provides rotation information for the pedestrian direction.
Properties

-ClassToSpawn Pawn; // Type of the class to be spawned according to implemented logic.
-SpawnedUnit Pawn; // Stores a reference to spawned unit
-RespawnTimer float; // Min value for respawn timer
-SpriteComponent* spriteComponent; // Component inherits from Bilboard Component which provides rotation information.

Methods
-virtual void BeginPlay(); -virtual void Tick(float DeltaSeconds);

Class Name MLNode
A single node only holds a single weight.

Properties
float input float output int layer
Methods
-void feed_forward(TArray<MLNode> &genome_nodes, TArray<MLConnection> &connections) -void add_connection(int connection_index)

Class Name MLConnection
Class that represents the connection links between layers.
Properties
-int from_node -int to_node -double weight -bool enabled
Methods
-void mutate()

Class Name MLGenome
Class that represents a genome of training.
Properties
-int input_count -int output_count -int node_count -int layer_count -TArray<MLNode> nodes -TArray<MLConnection> connections
Methods

```
-void new_connection(int from_node, int to_node, float weight)
-bool is_fully_connected()
-bool can_make_connection(MLNode& node1, MLNode& node2)
-void add_node_between(int f_node, int t_node)
-void remove_all_connections()
-void reset_genome()
-void mutate()
```

3 Tools and Technologies

Unreal Engine 4(UE4): DriveSim was developed around UE4 for engine features like rendering, user IO and physics.

Unreal Engine Assets: Assets provided by the engine like blueprints, widgets and simple meshes were used.

Unreal Engine Libraries: Library functions provided for the engine were used during the implementation of the project.

Microsoft Visual Studio IDE: Visual studio is the default IDE of the Unreal Engine for code development. It supports various languages but it was mainly used for C++ codes in this project.

4 Innovation and Creativity

Even though there are countless innovative ideas that are being used in games and simulations, there are not many that uses machine learning as a informative mechanic for driver candidates. Our aim is by introducing this new concept in our simulation to create a unique visualization of driverless car so that the trainee can observe and learn how the computer does the driving, and retrieve meaningful information that can help him/her.

5 Impact of Engineering Solutions

Use of games in education and training is important reduce costs and provide a safe environment. In addition to this, game engines are being used to provide visualization in different domains to provide insight, such as visualizing floods and fires [2]. DriveSim looks like a classic driving game however it focuses on making user to obey the rules and train using the system. It can be used as a support system which will provide the basic training prior or during a driver licensing process. Furthermore, machine learning part provides a training step which would give insight to use of a vehicle in different scenarios but more importantly we expect it to provide a basic insight into driverless cars in a simulation environment.

6 Contemporary Issues

The project is not expected to cause professional or ethical issues. All of the assets that will be used in the game will checked for copyright issues. Therefore, it will not contain anything that will violate copyright issues. This will be free to play game and the source code of the game will be provided publicly for educational purposes.

7 Lifelong Learning

DriveSim is meant to be a informative, entertaining for the users who are trying to improve their skills as drivers. Also, DriveSim provides simulation of driverless cars as a new research part of the project. The core feature of this is making a neural network to learn how to drive while not hitting static and dynamic objects such as walls, cars, pedestrians.

8 Conclusion

In this report, we have presented the finalized system architecture and overview in detail and concluded the structural information about DriveSim. We have aimed to explain in detail how we have accomplished what we have proposed in the previous reports, what were the necessary changes that we have done and the specifications of the project itself. These specifications gives detailed answer how and why DriveSim will be different compared to other similar simulations. DriveSim is aimed to be implemented in correlation with these factors.

Also, we have explained the impact of the engineering solutions upon our project and the contemporary issues and lifelong learning regarding DriveSim.

DriveSim will provide unique way to inform the trainee and teach them how to drive cars in a simulative manner. Also, it provides a unique way for teaching them how neural network thinks based on their choices. We believe that such an application and usage of neural networks will improve driverless car technology. Experiences learned through the simulation's life can be transferred real-life situation, if needed.

Appendix: User's Manual

Game is distributed as a single executable file. Main menu is displayed after executing the file.

General Usage

Main Menu: User can select Play Game from the menu to move to mode selection screen

Mode Selection: In this screen, user is shown two options in boxes. Simulation and AI mode.

- In the simulation mode user controls the vehicle inside the generated city.
- In the AI mode vehicles are trained using NEAT algorithm. Player can see the training process of the self driving cars.

Controls & Keybindings

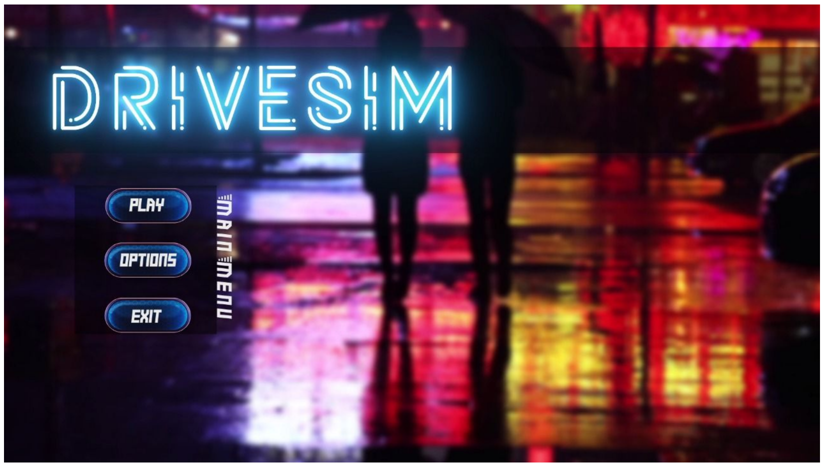
Game supports both keyboard and external wheel controller. These can be switched from the pause menu, which can be opened with Escape key.

- Keyboard Controls: W, A, S, D keys control the vehicle direction and speed.

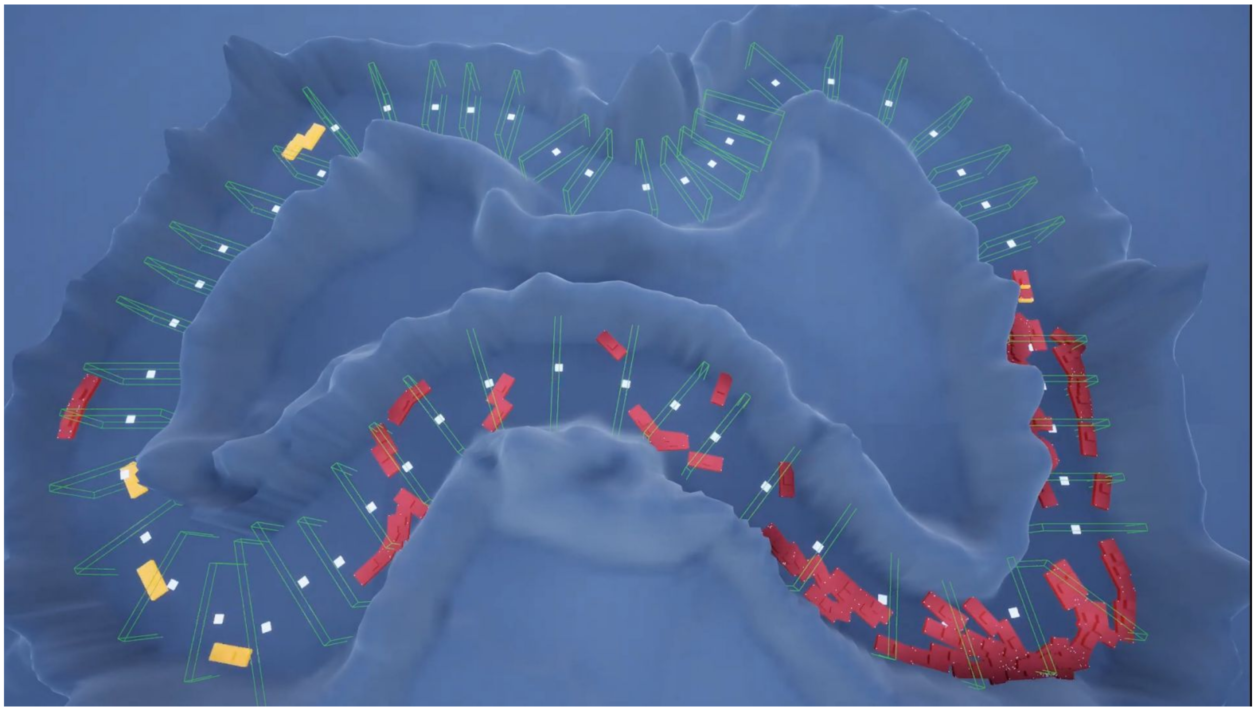
Gameplay

Simulation mode is based on point deduction. Colliding with other vehicles and pedestrians cause significant point loss, while moving on the sidewalks cause minor losses. There is an indicator shown on the map which looks like a target point. This shows the finish point to the player. There is also a mini map on the top right corner which might help with navigation in the surrounding area.

Appendix: Screenshots



Main Menu



AI mode training

References

[1] Adam Gaier, Alexander Asteroth and Jean-Baptiste Muret, “Data-efficient Neuroevolution with Kernel-Based Surrogate Models.” arXiv. [17 Apr 2018]. Available: https://arxiv.org/pdf/1804.05364.pdf

[2] Blondin Andy. “Floods and fires: how The Weather Channel uses Unreal Engine to keep you safe”. *UnrealEngine.* [Accessed 6 May 2019]. Available: https://www.unrealengine.com/en-US/spotlights/floods-and-fires-how-the-weather-channel-uses-unreal-engine-to-keep-you-safe