Linköpings University

Master Thesis

Passenger flow analysis using behavior trees in Unreal Engine 4

Ramin Assadi

supervised by
Dr. Pierangelo Dell'Acqua

September 13, 2017

Abstract

This thesis report will show how to create an simulation in Unreal Engine 4 with behavior trees. The thesis will explain the purpose and usage of the simulation. It will cover how the creation of an AI is made and how to make the AI behave in different ways. It will cover what behavior trees are and how behavior trees are implemented in Unreal Engine 4. The simulation will cover how the AI enters, finds a seat, gets seated and exits a vehicle. The connection between all methods will be made and explained.

Contents

List of Figures

Chapter 1

Introduction

In this chapter I will present an introduction of my thesis. First I will talk about the company XperDi and their purpose for the thesis. Afterwards I will talk about the methods used and then mention the delimitations.

1.1 XperDi

XperDi is a company located in Linköping, Sweden. They specializes in configuring 3D-models in CAD and game engines. With their product, customers can reduce the amount of time consuming CAD-modelling processes and stay focused on building better products. At this moment they are working with companies that are associated with buses and trains.

1.2 Purpose

The purpose of the thesis work is to see how much time it takes for the passengers to enter, find a seat and exit the vehicle. This can help the team at XperDi to see if the 3D-models for the vehicles needs to be adjusted to make the vehicles more time-efficient. In other words, the passenger flow analysis will show if the vehicles can be improved to save time for passengers to enter, find a seat and exit the vehicle. This can help companies to find issues in the design before they actually make the vehicle. This will save time and money.

1.3 Methods

The methods used in this thesis work is Unreal Engine 4 and Behavior Trees. In-depth explanation of the methods can be read at chapter 2. The main reason for using Unreal Engine 4 is because XperDi's product is made in Unreal Engine 4. The thesis work will therefore be able to be used with their product.
When it comes to the AI, the use of Behavior Tree was familiar to me. In a previous project I've made, I came to learn that Unreal Engine 4 has an really good tool for Behavior Trees. The combination of using Unreal Engine 4 with Behavior Trees was an obvious choice because of the previous work I've done and the tools made by Unreal Engine 4.

1.4 Goal

Because XperDi demands that the passenger flow analysis needs to be created in Unreal Engine 4 the goal is to see if it is actually possible to do this. If it is possible then the passenger flow analysis shall be implemented as much as possible in the specified time frame. The idea is to create a backbone for XperDi to then further develop.

1.5 Delimitations

The purpose with the thesis work is to see how long it takes for passengers to enter, sit down and exit the vehicle. Therefore this thesis work will not take into account the different situations that will occur in traffic or other phenomenons. The soul purpose is to see if the design of the vehicles can be improved to shorten the time for passengers to get in and out of the vehicles.

Chapter 2

Literature Study

This chapter will go through similar projects to compare and analyze what the differences are, to determine what methods should and should not be used in this thesis work.

2.1 Passengers flow analysis and security issues in airport terminals using modeling & simulation

The paper[10], talks about dealing with passenger flow and security issues of an Italian airport terminal in Calabria. They create an simulation model in AnyLogic to calculate the system performance. The system performance is the average wait time for each passenger at the gate area. The purpose of this paper is to see if the wait time of a passenger in an airport terminal increases or decreases depending on what happens. For example they change the amount of check-in places from 10 to 15 and see that the wait time goes down by 10 minutes. With this information they can optimize the airport to reduce the wait time for each passenger.
When looking at the methods used we can see that there is no visualization of the system. In our thesis work there is a demand from XperDi to use Unreal Engine 4. While the result of the paper[10] is interesting and good, we can not be using this because of the limitations in Unreal Engine 4. First of AnyLogic is based on the language Java and Unreal Engine 4 is based on C++. Secondly modeling of an simulation does not create a visualization of the system. To make this possible in Unreal Engine 4 there needs to be an AI character with some kind of logic connected to it.

2.2 Modelling Urban Bus Service and Passenger Reliability

The paper [11], talks about measuring reliability of public transport systems. The study is using a dynamic microsimulation model framework applied to a case study based on a bus route in the city of York. In the paper they model

a simulation to figure out what makes public transports unreliable. They take into consideration the travel time, headway and passenger wait time to make the models of the simulation.
As before this paper models the simulation and does not have any visualization of the system. Also because of the limitations of using Unreal Engine 4 the methods used does not have anything to do with AI.

In all the papers regarding simulation of passengers, they all have modeling of simulations in common. Because XperDi is demanding the passenger flow analysis to be created in Unreal Engine 4, the use of an AI is necessary. As a result this thesis work will be to figure out the best method for creating an AI in Unreal Engine 4 and afterwards simulating an passenger flow analysis.

Chapter 3

Background

3.1 Theory

When constructing a game there is three Hierarchical logics that can be used that is suited for games. These are Scripting, HFSM (Hierarchical Finite State Machines) and Planners HTN(Hierarchical Task Network). Each of these hierarchical logics has its own advantages and disadvantages. When it comes to scripting the advantages is that any computation is possible. But the downside of scripting is when looking inside of the script and figuring out what the script is doing and what it will do. For example planning ahead is not an easy thing to do with scripting. And for designers the scripts are not accessible. A solution to scripting would be to use HFSM. These are great for designers, the only thing they have to do is to take a bunch of states and edit transitions, and plug them together. That gives them all the control they need at the lowest possible level. The downside of HFSM is that it takes a long time to implement all the transitions and states. One possible solution to HFSM is planning. Planning essentially uses a form of search to plug together these modular behaviors to reach any kind of objective within a game. And that provides a certain level of automation. The downside is that planners are more difficult to implement than scripting. The problem here is that an elegant solution in theory can result in a brute force solution in practice. So the problem is that all these hierarchical logics have there advantages and disadvantages. What would be the best suitable solution for a game would be a combination of these logics to fix each others disadvantages. This is were Behavior Trees comes in. This does not mean that behavior trees do it better then the other logics. This is just a solution that makes it easier for game developers. Behavior Trees makes it very easy to make decisions over time, follow out those decisions and making sure the plans executes correctly. This is why Behavior Trees are good for games.

3.2 Unreal Engine 4

The first method used in this thesis work is Unreal Engine 4. Unreal Engine 4 is a complete suite of game development tools made by game developers, for game developers. [1]
Unreal Engine is a game engine developed by Epic Games. It was showed for the

first time in 1998 in the game Unreal. At first it was made for developing first-person shooters. As time went by the engine was used in many other games and then later on, Epic Games decided to put there focus on their engine instead. In November 2009 Epic Games made the first free version available to the general public, it was called Unreal Development Kit and was using Unreal Engine 3. And then at March 2014 they released Unreal Engine 4 with all of its tools and features.
Now Unreal Engine 4 is being used by several companies not only for game development but for various different projects. XperDi is now using Unreal Engine 4 to make it easier to do CAD-modeling and in this thesis work I'm using It to do an passenger flow analysis. The possibilities are endless and this is why Unreal Engine 4 is one of the biggest engines in the world.

3.3 Behavior Tree

The second method used in this thesis work is Behavior Trees. There are three different nodes in a Behavior Tree. Composite, Decorator and Leaf. The composite can consist of different children. There are two different composite nodes. Selector and Sequence. The sequence node tells each child to run one at the time until all children succeeds. The selector node runs each child until one returns success then the selector stops. A decorator node, like a composite node, can have a child node. Unlike a composite node, they can specifically only have a single child. Their function is either to transform the result they receive from their child node's status, to terminate the child, or repeat processing of the child, depending on the type of decorator node. The leaf node is the lowest node and therefore has no children. This node will carry out the task that the AI is going to do.
When it comes to Behavior Trees in Unreal Engine 4 there are couple of components that is necessary to have for the AI to be complete. In Unreal Engine 4 there are four different components. Conditions[1], Loops[2], Tasks and Blackboard. The AI needs to store some values that the user sets. In other words it needs a memory. If the AI should be smart and make decisions depending on what is going on in the game, there needs to be a memory for the AI to store information in. The blackboard is the memory. In the blackboard whatever value can be stored. But the important part is to not store to much information in the blackboard since it increases the time to save and retrieve the data. The tasks in Unreal Engine 4 is the leaf node. The conditions and loops are connected. The conditions are different queries that can be put on the composite and leaf nodes. It simply asks the blackboard for information to see if the value is true or not. It is in other words a if statement. The loops look at the world and change the blackboard depending on what happens. For example a loop can look if the AI is sitting down or not and save the result in the blackboard. Then the condition will ask the blackboard if the AI is sitting down and if it returns true or not depending on what the condition is, it will proceed or stop. The one thing that makes Behavior Trees extremely suitable with Unreal Engine 4 is the debugging system. With the debugging system in Unreal Engine 4 it is possible to pause the behavior tree at any point. When the behavior tree

[1]In Unreal Engine 4 they are called Decorators
[2]In Unreal Engine 4 they are called Services

is paused, the possibility to jump back and forward in time is possible. While doing the debugging it is also possible to see what the value of the blackboard is at that specific time. This makes it easy to see when and why things happen or not. In other words it is possible to see how the behavior tree behaves at a specific time to pinpoint problems and bugs in the AI.

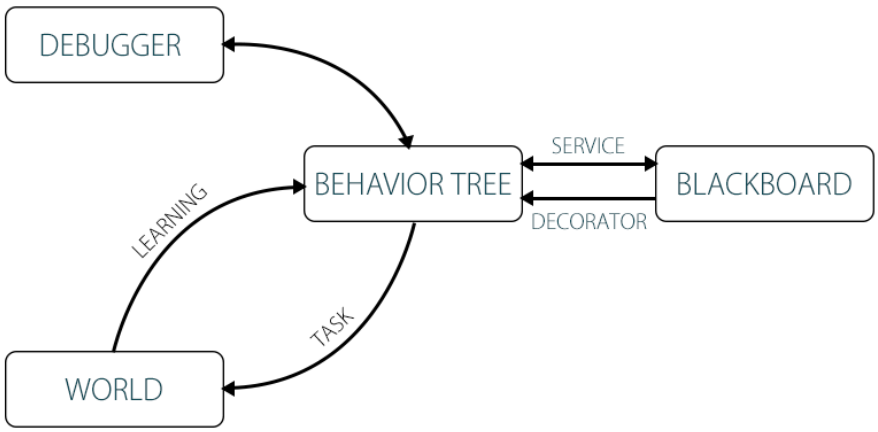

Figure 3.1: System architecture of the Behavior Tree in Unreal Engine 4

In figure 3.1 we can see the system architecture for behavior trees in Unreal Engine 4. What the world represents in the image is all of the information from the game. The behavior tree can learn anything from the world and use it to modify the behavior. We can see that the service can change and receive information from the blackboard while the decorator only can receive information. When it comes to the debugger it can get and send information to the behavior tree. And when all the interactive parts are done it will end with the behavior tree sending a task back to the world. This task is what makes the AI interact with the world. This process continuous until the AI is removed.

Chapter 4

Implementation

In this chapter the implementation will be explained. The first section will cover the game creation and its features. The second section will be about the AI itself with all of its components explained in detail. To easier understand the variables and functions it can be useful to look at the result in chapter 5 while reading this section.

4.1 The Game

To be able to create an passenger flow analysis in Unreal Engine 4 a game needs to be constructed with rules how the player and passengers can interact with the world. In other words the passenger flow analysis is a game were AI's will enter a vehicle, sit down and exit while the player is observing.

4.1.1 Player

The first thing created is the player. In order to get the player working there are three important parts that needs to be created. The character blueprint, controller and HUD.

Character Blueprint

The character blueprint is the core of the player and where the model and settings are set. In this case the model only consists of an camera because the player is only made to observe the game. The speed of the movement is also set in this class. After the character blueprint is created there needs to be a controller connected to the character.

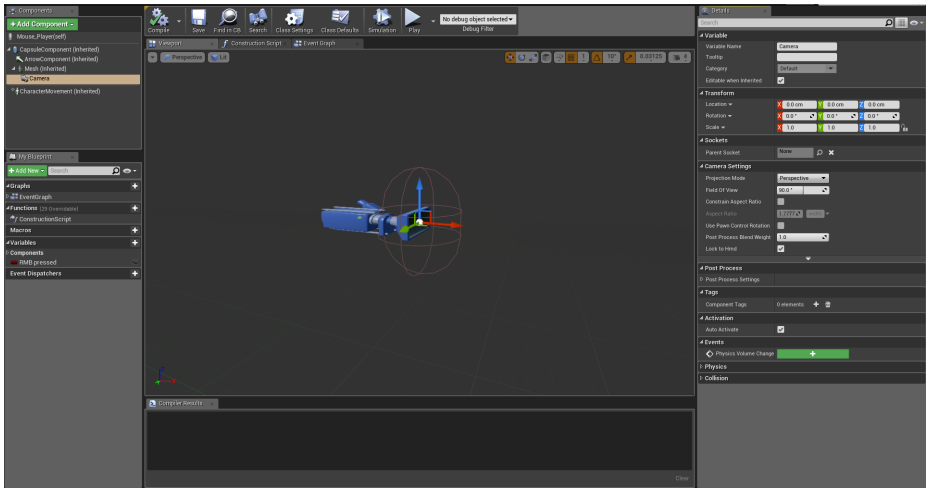

Figure 4.1: A preview of the character blueprint in Unreal Engine 4

Player Controller

The controller will store the information about how the character can interact with mouse and keyboard. In this project the keyboard will control the translation of the camera while the mouse will control the pitch. The mouse should also be able to interact with the HUD.

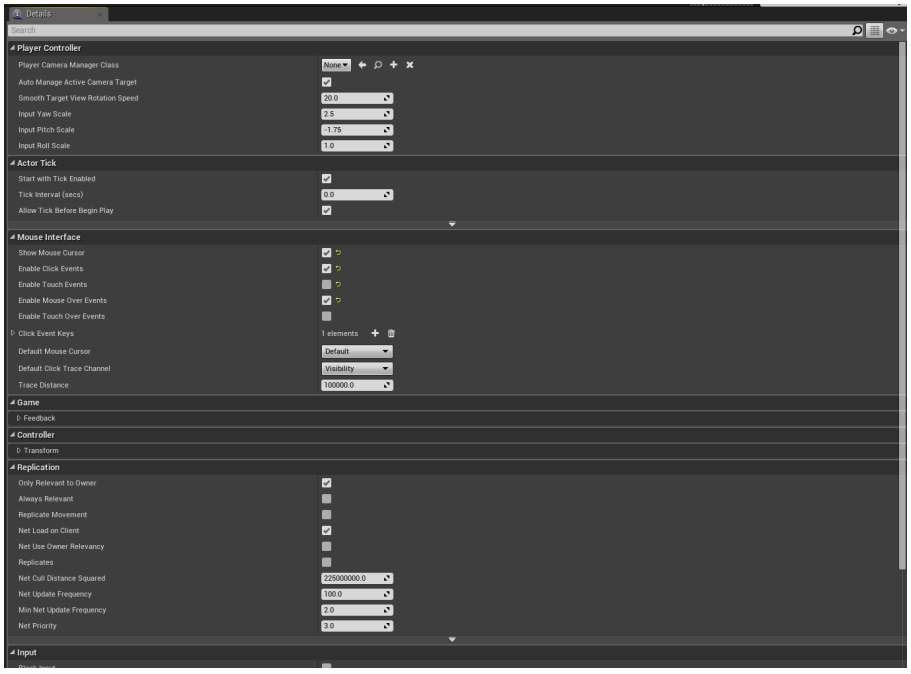

Figure 4.2: Settings in the player controller

HUD

The last part that needs to be created is the HUD. The HUD is the visual representation of information on top of the player screen. The HUD consists of buttons and information. The HUD has four buttons. Start/Stop Simulation, Add Passenger and two buttons to increase and decrease the number of passengers to add. The Start/Stop Simulation is connected to the behavior tree for each AI. When the simulation is started the behavior tree gets activated and when stopped the behavior tree gets deactivated. The increase and decrease button will change the amount of passengers to be added when the Add Passenger button is pressed. There is an number displayed on the HUD to see the amount of passengers that will be added when pressing the Add Passenger button. The HUD also shows the elapsed time for the simulation. This function will be activated when Start Simulation button is pressed and stopped when Stop Simulation is pressed.

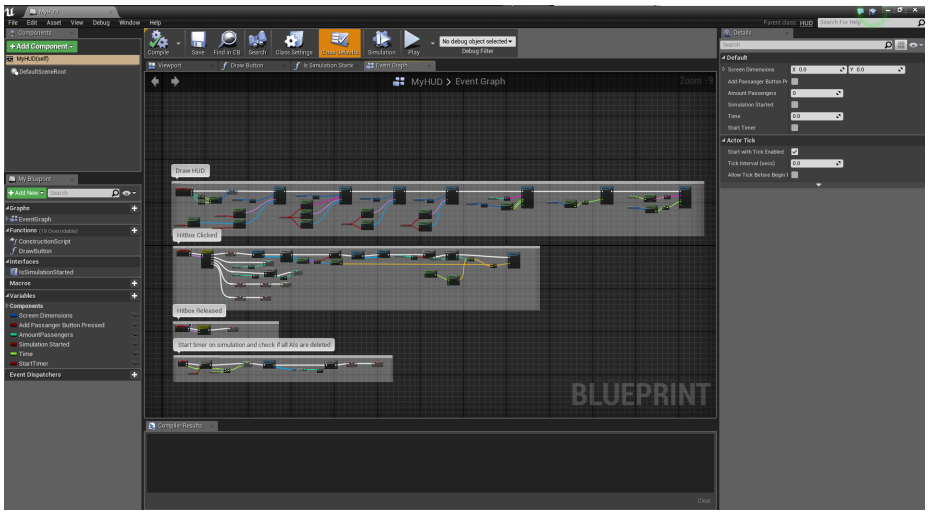

Figure 4.3: The HUD in Unreal Engine 4

4.1.2 Model

The whole project evolves around the vehicle of the game. XperDi works to reduce the amount of time CAD-modeling. In other words they try to make tools to easy change the model in buses and trains. Therefore the model used in this thesis is provided by XperDi.

Figure 4.4: The bus model provided by XperDi

4.2 AI/Passenger

This is the section with the main focus of the thesis work. This section will contain explanation of each part of the AI and exactly how they are implemented in the thesis work. The section is divided in the different components containing in the AI and lastly how the different components are connected in the behavior tree.

4.2.1 Services

This section will cover all of the services used in the AI. For information about services and in-depth explanation read section 3.3.

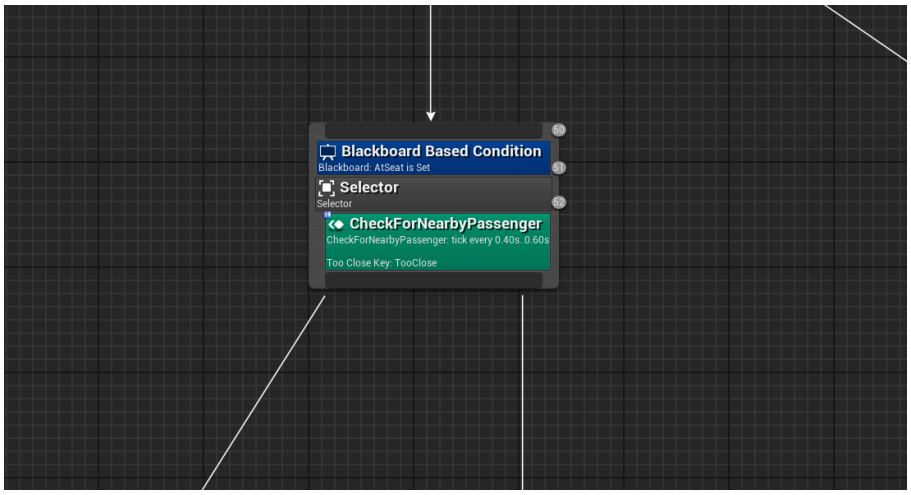

Figure 4.5: The green node represents the service node in Unreal Engine 4

In figure 4.5 we can see the green service node, CheckForNearbyPassenger. We

can see that this service changes the TooClose blackboard key whenever a passenger is too close. The different service nodes used in this thesis work will be explained in detail in this subsection.

IsSimulationStarted

This service checks if the simulation is started or not. It fetches information from the HUD class and looks if the Start Simulation button is pressed and active. If it is it returns true and changes one of the blackboard values so the behavior tree can start executing.

CheckForNearbyPassenger

This service checks if there are any nearby passengers for the current AI. It uses the AI's front vector and calculates a cone in that direction. The radius of the cone determines how wide the AI will look for nearby passengers. If a passenger is in the cone radius and is close enough to the AI it will return a bool with the value true.

IsEveryoneSeated

This service checks if all the passengers in the game is seated or not. It is quite simple, it checks if every passenger has the seated blackboard value set to true. If every passengers value is set to true the service returns a bool that will cause the simulation to go to the next stage, the transfer and exiting stage.

IsPassengerAtEntrance

This service checks if the current AI is at the entrance of the vehicle or not. It checks for a target point called Entrance Point and takes the points position. A circle around the position is created with a specific radius. If the passenger is in the circle the service returns a bool with the value true. This service will be changed in a later development to get the position of the model with the name door instead of the target point. The model does not support this right now and that is why the target point solution is created.

IsPassengerAtBuy

This service works exactly the same as IsPassengerAtEntrance. The only difference is that it checks for the Buy Point instead of the Entrance Point.

4.2.2 Decorators

The Decorators are if statements for values in the blackboard. The specific values in the blackboard will be in subsection 4.2.4. For information about decorators and in-depth explanation read section 3.3.

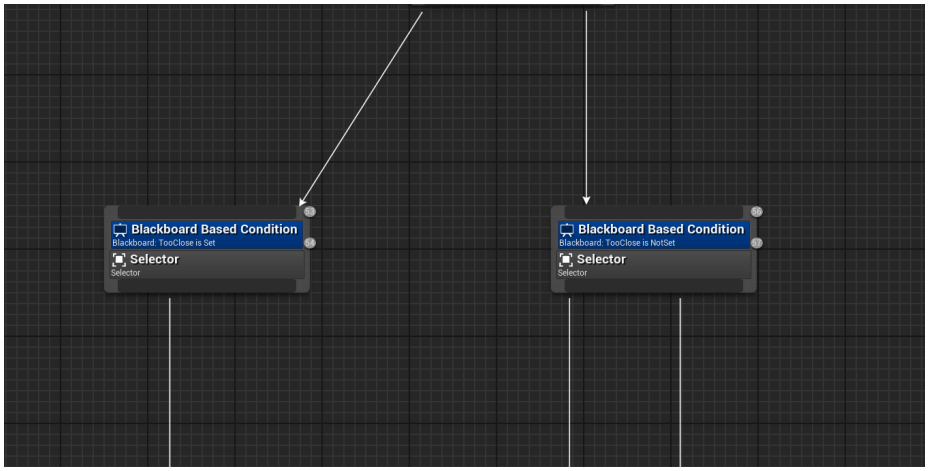

Figure 4.6: The blue nodes represents decorators in Unreal Engine 4

In figure 4.6 we can see that the left node is a decorator when the blackboard value TooClose is set to true and the right node is when the value is set to false.

4.2.3 Tasks

This section will cover all of the tasks used in the AI. For information about tasks and in-depth explanation read section 3.3.

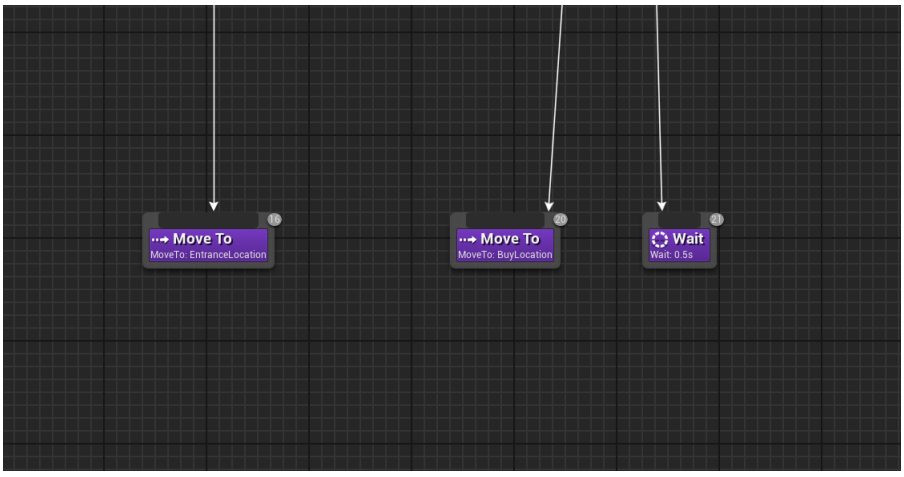

Figure 4.7: The three purple nodes represents task nodes in Unreal Engine 4

In figure 4.7 we can see three different task nodes. The different task nodes used in this thesis work will be explained in detail in this subsection.

FindSeat

This task calculates the desired seat and saves it to the blackboard. It checks for the closest seat. The seats are in groups of two. If one of the seats are taken then the task will look in the next closest group. If there are no free groups it will take the closest group with only one seat taken. The seats are searched by name and saved in the blackboard. By saving the seat as a object in the blackboard we can easily get the position of the seat in a later time. This makes it also possible to have different vehicles and still have the task working without changing anything. The only requirement is that the seats should have the word seat in it. The task also returns a bool to the blackboard to confirm that a seat is found and set.

MoveToSeat

This task moves the AI to the selected seat. It will take the seat object from the blackboard and get the position of it. Because there is blocking object in the pathway the position needs to be changed to the walkway in the middle of the vehicle. As soon as the position is set, the AI will be instructed to go to the position. The task will check if the AI is close enough to the position and return a bool with the value true if it is.

TeleportToSeat

This task will teleport the AI to the seat. Because of the blocking pathway to the seats the AI needs to be teleported to the seat. In the thesis work the AI will walk next to the seat and then be teleported to the seat. What should be implemented in a later stage is to trigger an animation of the AI going to the seat and sitting down. After the animation is done the AI should be teleported to the seat. As soon as the AI is teleported a bool is returned to the blackboard to confirm that the AI is now seated.

CloseDoor

This task closes the doors to the vehicle. If all passengers are seated the doors to the vehicle will be closed and a timer for the duration of the ride will be started. This task should be extracted from the passengers and added to the driver of the vehicle. Because there are no drivers in this thesis work the passengers handle this task.

OpenDoor

This task will open the door to the vehicle. When the timer for the duration of the ride is finished this task will open the door to the vehicle and then return that the doors are open and it is time for the passengers to exit the vehicle. This task should also be extracted to the driver of the vehicle.

IsPassengerClosestToBuy

This task will check if the current passenger is closest to the buy point. If the condition is true, it is allowed to stand up och start walking to the exit. The task also takes into consideration if any passengers are in the way. If passengers

are in the way the passenger waits and stands up after the passenger has moved out of the way. This task will also change the seated bool value to false to make the decorators able to know that passenger is not seated now.

ResetTheValues

This task will reset a couple of the values in the blackboard to make it possible to reuse decorators and services when passenger is trying to exit the vehicle.

TeleportOutOfSeat

This task will teleport the passenger out of the seat and on to the pathway next to the seat. When the passenger is able to stand up and exit the vehicle, this task is needed to teleport the passenger out from the seat and on to the pathway. It will teleport the passenger to the position before TeleportToSeat task was used. This task also returns a bool to confirm that the passenger is not seated.

RemovePassenger

This task is used to remove the passenger when it exits the vehicle. When the passenger is outside of the vehicle the passenger will be removed. When all passengers are removed the simulation will be stopped. The elapsed time will now show how long time it took to run the simulation. At this point the simulation can be started again with different amount of passengers.

4.2.4 Blackboard

This section will cover the blackboard values used for the AI. For information about blackboard and in-depth explanation read section 3.3.

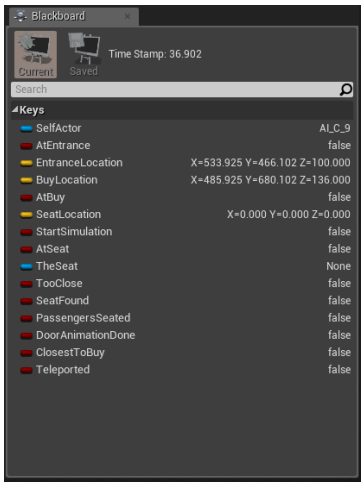

Figure 4.8: The values stored in the blackboard

In figure 4.8 we can see a preview of all the values stored in the blackboard. Each value will be explained in detail in this subsection.

SelfActor

This blackboard value represents the current AI actor. It's saved as an object and is used to get different values from the actor for services and tasks.

TheSeat

This blackboard value represents the seat that is selected by the AI. It's saved as an object and is used to see what seat is taken or not by other passengers.

EntranceLocation

This blackboard value represents the location of the entrance. It's saved as an vector and is used when entering and exiting the vehicle.

BuyLocation

This blackboard value represents the location of the buy terminal in the vehicle. It's saved as an vector and is used when entering and exiting the vehicle. When exiting it is used only for an extra point to show the AI the correct way to move out of the vehicle.

SeatLocation

This blackboard value represents the location of the seat for the specific AI. It's saved as an vector and is used when the AI is going to the seat.

StartSimulation

This blackboard value shows if the simulation is started or not. It's saved as an bool. This value is changed when the Start/Stop Simulation button is pressed.

AtEntrance

This blackboard value shows if the AI is at the entrance or not. It's saved as an bool. This is used when entering and exiting the vehicle.

AtBuy

This blackboard value shows if the AI is at the buy location or not. It's saved as an bool. This is used when entering and exiting the vehicle.

AtSeat

This blackboard value shows if the AI is at the seat location or not. It's saved as an bool. This is only used when entering the vehicle.

TooClose

This blackboard value shows if the AI is too close to another AI. It's saved as an bool and is used all the time to keep the AI from walking in to each other.

SeatFound

This blackboard value shows if a seat is found. It's saved as an bool and set to true when the FindSeat task finds a seat.

PassengerSeated

This blackboard value shows if the AI is seated or not. It's saved as an bool and is used to check if all passengers are seated. And if the case is so it continuous the simulation to the next stage.

Teleported

This blackboard value shows if the AI is teleported to the seat or not. It's saved as an bool and is used to see if the teleportation from pathway to seat is completed.

DoorAnimationDone

This blackboard value shows if the animation for the door is complete. It's saved as an bool and is used when the vehicle is closing and opening the door.

ClosestToBuy

This blackboard value shows if the current AI is the closest passenger to the buy location. It's saved as an bool and is used when the passengers are about to exit the vehicle. This determines which passenger is allowed to get up from the seat first and then move to the exit.

4.2.5 Behavior Tree

This section will cover how the different parts are connect in the behavior tree. In other words this section will take all the services, decorators and tasks and connect them together with selector and sequence nodes. For information about behavior tree and in-depth explanation read section 3.3.

The behavior tree start with a selector that contains a service called IsSimulationStarted. This service keeps checking if the Start Simulation button is pressed or not. As soon as the simulation is started the service sets the StartSimulation bool value to true.

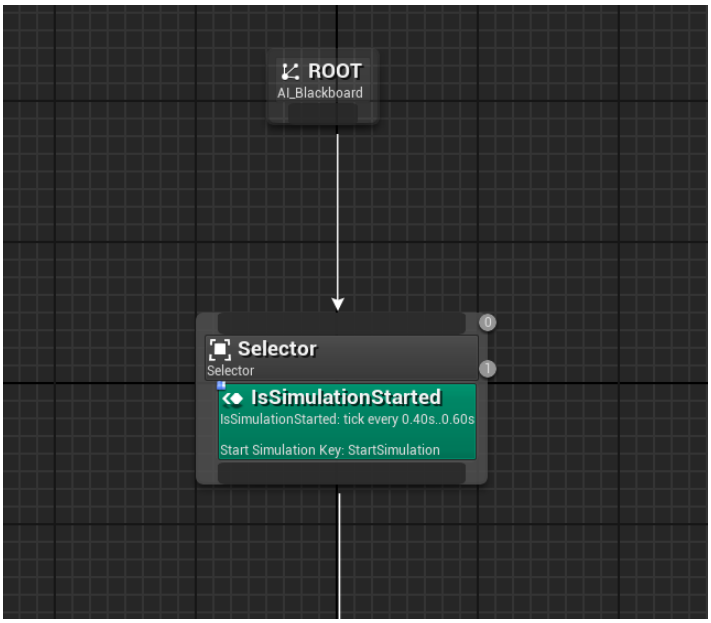

Figure 4.9: First selector containing the IsSimulationStarted

The next selector has a decorator with the StartSimulation value. This selector has three children. These children are the three different stages of the simulation. The first one is the enter vehicle and finding a seat stage. Second one is the travel stage and the last one is the exiting vehicle stage. The PassengerSeated bool is set to false at the start and therefore will stage one will be activated. As soon as the PassengerSeated bool gets the value true, stage two will be started.

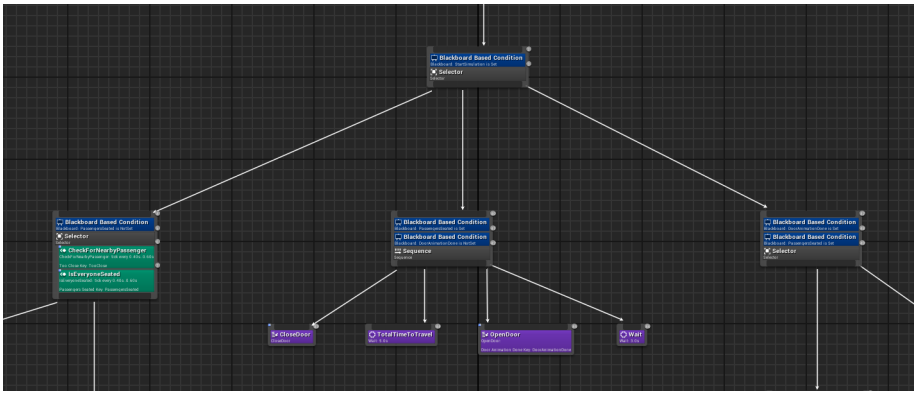

Figure 4.10: The selector dividing the behavior tree into three different stages

When the first stage gets started the first selector has two services connected to it. CheckForNearbyPassenger and IsEveryoneSeated. IsEveryoneSeated is checking if every passenger is seated so it can change the PassengerSeated bool and start the second stage. The CheckForNearbyPassenger service will make

two selector children in the tree. If there are nearby passengers it will change the TooClose bool to true. In this case if the TooClose is set to true, nothing happens, in other words the passenger stands still. If there is no passenger close enough the tree continuous down to the next selector.

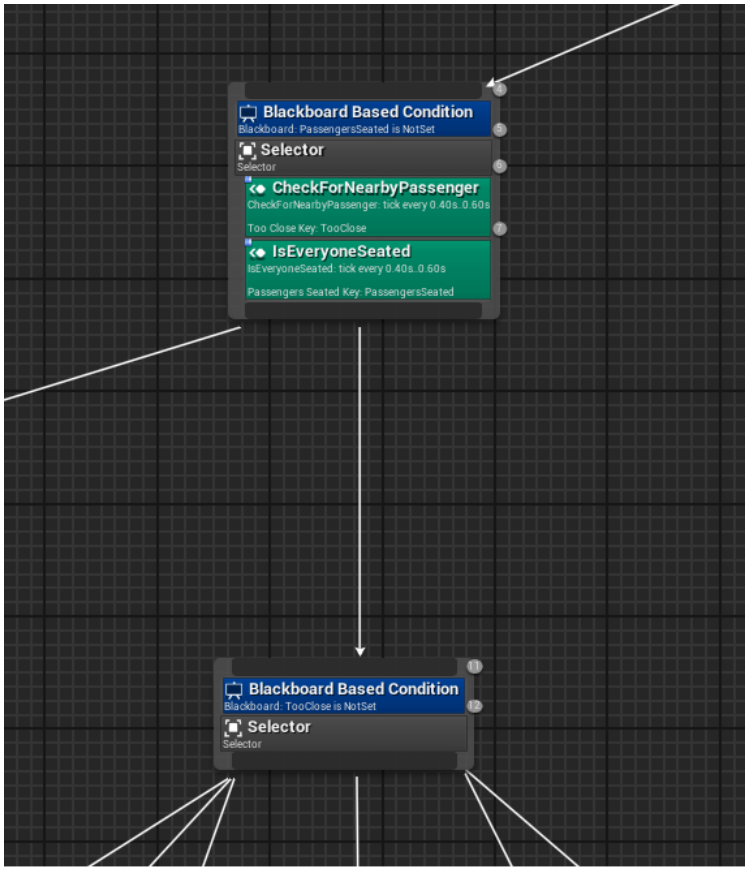

Figure 4.11: TooClose selector at stage one

The next selector will branch into six different composite nodes. They are all executed like an sequence node but they are in a selector node. The reason for this is because, if one child execution returns fail the sequence is stopped and started over again. In this case we want to calculate the six different nodes in a sequence but we do not want the composite node to start over if one of the children fails. To make this happen a lot of decorators are needed in a selector node.

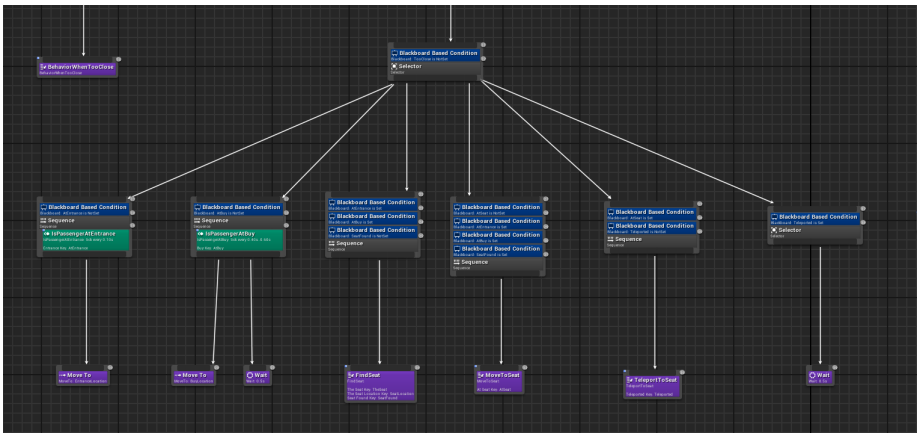

Figure 4.12: The six different composite nodes

The first composite node of the six is a sequence node that has a decorator and service connected to it. The AtEntrance decorator will indicate when the Move To task is completed and make it possible to go to the next composite node. The IsPassengerAtEntrance service is the loop that checks if the AI is close enough to the entrance point and then returns the value of AtEntrance bool. When passenger is at entrance the decorator will be set to true and the second composite node will be active.
The second composite node is exactly like the previous one. The only difference is the location the passenger needs to go to. It's the buy location in this case. There also is a wait task to simulate the time it takes for the passenger to pay for their ticket. When the passenger is close enough to the buy location and the wait task is completed, it will set the AtBuy bool to true and continue to the third composite node.

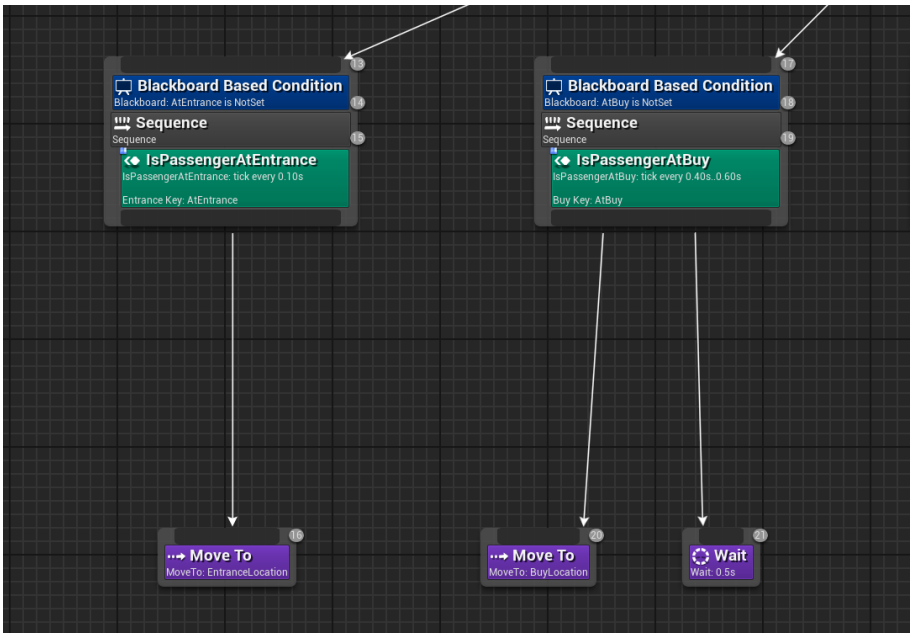

Figure 4.13: The first and second composite node

The third composite node is a sequence node with three decorators connected to it. Because the parent composite node is a selector we now need decorators to tell us the first two composite nodes are completed and therefore the AtEntrance and AtBuy values are needed. The last decorator is the SeatFound bool value. This is so when the task FindSeat has found a seat and it returns true on SeatFound bool it will continue to the forth composite node. The SeatFound will find the specific seat for the passenger and save the seat object in TheSeat and the location of the seat in SeatLocation.
The forth composite node is a sequence node and like the previous node it needs to know that the three previous composite nodes are executed and completed. Therefore it has all of the previous decorators and a new one called AtSeat. The task MoveToSeat will move the passenger to the seat and when it is close enough it will change the AtSeat value to true. Now the fifth composite node will be executed.

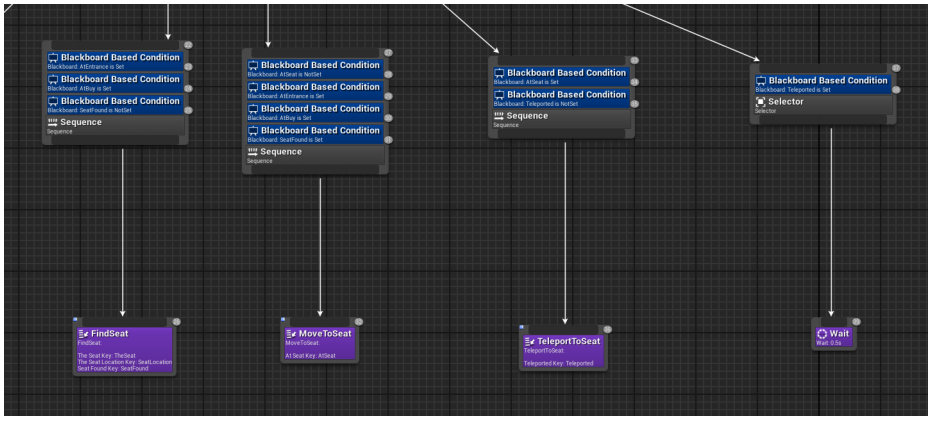

Figure 4.14: The rest of the composite nodes

Now the same logic will be applied to the two last composite nodes. First the TeleportToSeat will teleport the passenger to the seat and then the last composite node is there to keep the passenger waiting for every passenger to be seated.
In other words the passenger will go to the entrance then go to the buy location. After the passenger has paid the ticket, the passenger will find a seat, move to the seat and teleport to that seat. Then the passenger will wait for all other passengers to be seated. When all passengers are seated stage two will begin.
The second stage is the travel stage. This is where the animations of the doors and the travel time is set and controlled. It is all controlled with an sequence node. To the sequence node there are two different decorators connected. PassengersSeated is for knowing that stage one is completed and DoorAnimationDone decorator is to make the behavior tree to continue to stage three.
The sequence node will run four different tasks. It will first do the CloseDoor task. The task runs the animation for closing the doors. The next task is a wait task and is used to simulate the time it takes to travel to the next stop. In this thesis the value is set to five seconds. The third task OpenDoor, runs the animation for opening the doors. It will also return true for the DoorAnimationDone value. The last task is another wait task that is used to even out the animation time for opening the doors. When the last task is done stage three will be started. Stage two should be exported and imported to a driver behavior tree instead. For all the passengers to check if they all are seated is not necessary. Because we are using behavior trees this can easily be transferred to another behavior tree. In this thesis there is no driver AI and therefore is stage two implemented in the passenger behavior tree.

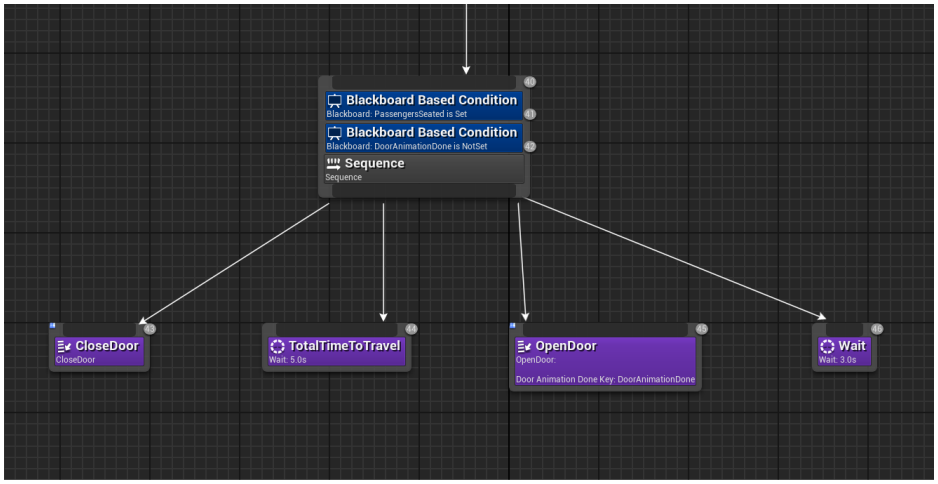

Figure 4.15: Stage two of the behavior tree

The third and last stage is divided into two branches. The first one is for checking if the passenger is allowed to stand up from their seat. It will start with checking for nearby passengers and if they are too close to the passenger. If so it is not allowed to move forward in the branches. If there is no passenger in the way it will proceed by first doing the IsPassengerClosestToBuy task. This task looks if the passenger is the closest passenger to the buy location. If the case is true the passenger is allowed to proceed in the tree. When ClosestToBuy value is set to true the next task is ResetTheValues. This task will set the AtEntrance and AtBuy value to false to make it able to reuse the values for exiting the vehicle. Afterwards the TeleportOutOfSeat task will be executed. This task will teleport the passenger out of the seat and onto the pathway. The AtSeat value will be returned as false making it possible to continue to the second part of stage three.

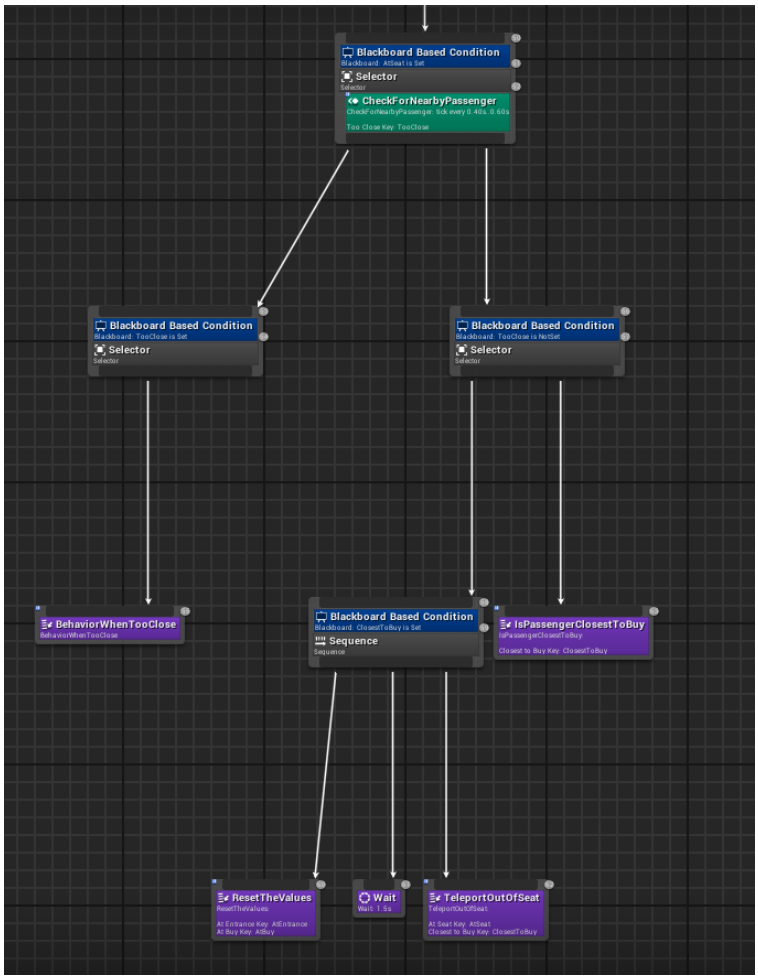

Figure 4.16: First part of stage three

At this stage the passenger has moved from seated to standing. This last part will reuse the same methods for entering the vehicle to exit the vehicle, the only difference is that i will execute it in revers. As soon as the passenger is outside of the vehicle the passenger will be removed. When all passengers are removed the simulation will be stopped. At this point the elapsed time can be saved and a new simulation can be initialized.

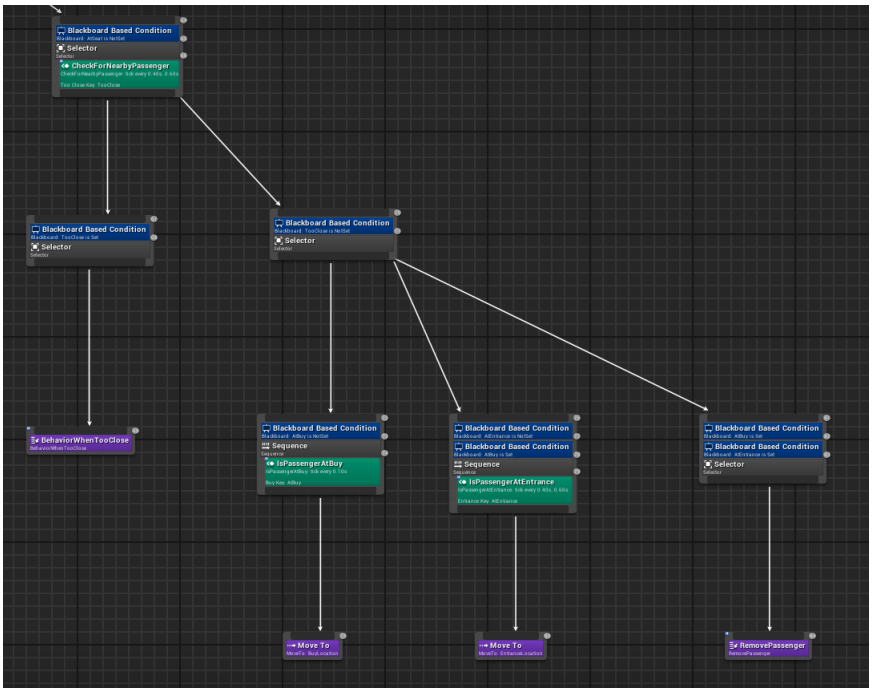

Figure 4.17: Second part of stage three

Chapter 5

Result

This chapter will show the result of the thesis. It will start with showing values of the blackboard in different stages at the simulation. Afterward the behavior tree will be shown and the chapter will end with three runtime videos.

5.1 Blackboard

There are three different figures in this section. They all explain one of the stages in the simulation.

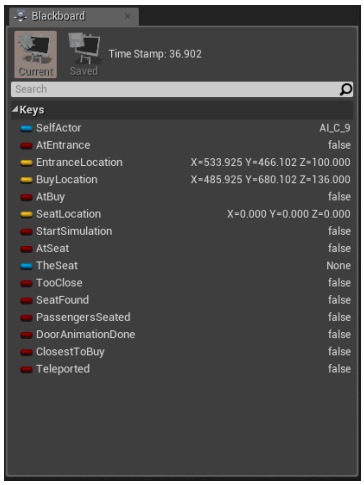

Figure 5.1: Blackboard values at stage one in the simulation

In figure 5.1 we can see the values when the passenger is at stage one. Every bool is set to false and none of the values are set except the two different locations. By looking at the values we can see that this passenger is at the start of the simulation and nothing has yet been done.

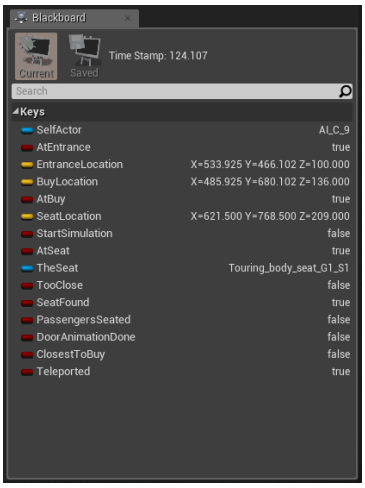

Figure 5.2: Blackboard values at stage two in the simulation

In figure 5.2 we can see the values when the passenger is at stage two. In this blackboard, stage one has been completed and the passenger is seated. The passenger is waiting for the other passengers to find a seat and sit down.

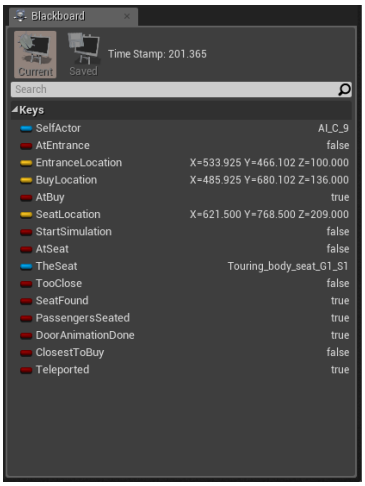

Figure 5.3: Blackboard values at stage three in the simulation

In figure 5.3 we can see the values when the passenger is at stage three. In this blackboard we can see that stage two is completed and the passenger is no longer seated. The passenger has walked to the buy location and is moving towards the entrance location that is the exit of the vehicle.

5.2 Behavior Tree

This section will show each of the stages of the behavior tree. The green nodes are the services, the blue nodes are the decorators and the purple nodes are the tasks. The gray nodes are the selector and sequence nodes.

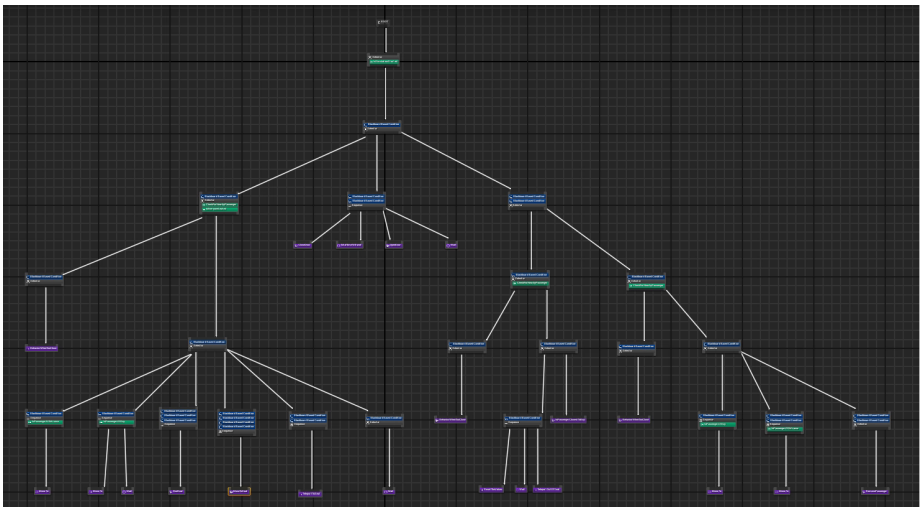

Figure 5.4: The whole behavior tree

Figure 5.4 shows the whole behavior tree. This is the behavior tree with all of the three stages. In the next figures each of the stages will be shown in more detail.

5.2.1 Stage one

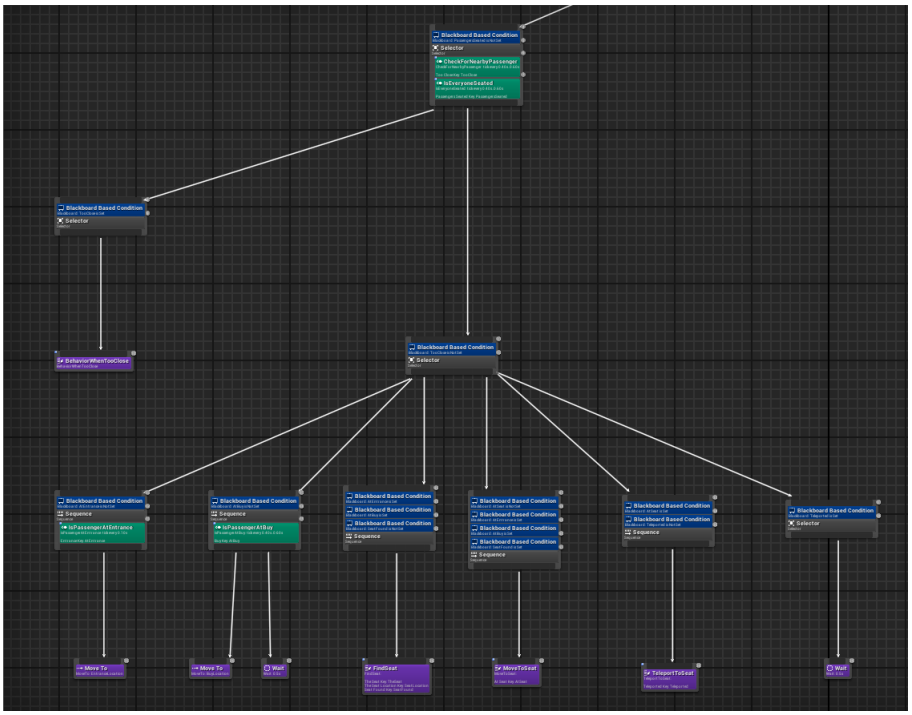

Figure 5.5: Stage one of the behavior tree

Figure 5.5 shows the first stage of the behavior tree. At the beginning of the simulation when the Start Simulation button is pressed this is the stage that will be activated. The first two children nodes it jumps between is to see if a target is too close or not. The branch to the left does nothing when something is too close and the right branch continues the behavior so the passenger can walk in to the vehicle, find a seat and sit down. As soon as this stage is completed the second stage will be activated.

5.2.2 Stage two

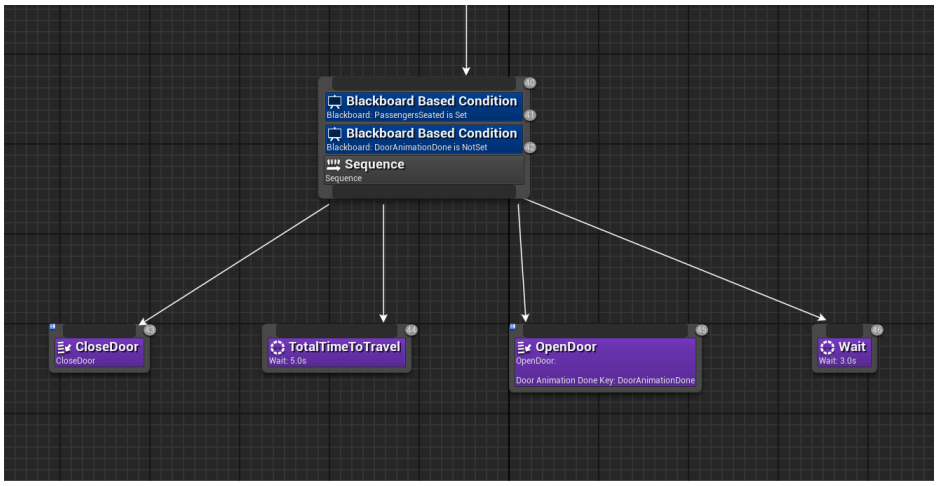

Figure 5.6: Stage two of the behavior tree

Figure 5.6 shows the second stage of the behavior tree. This stage will be activated as soon as all of the passengers have completed the first stage. As soon as stage one is completed this stage will close the doors and start a timer on how long the ride will last. Afterwards it will open the doors and start stage three.

5.2.3 Stage three

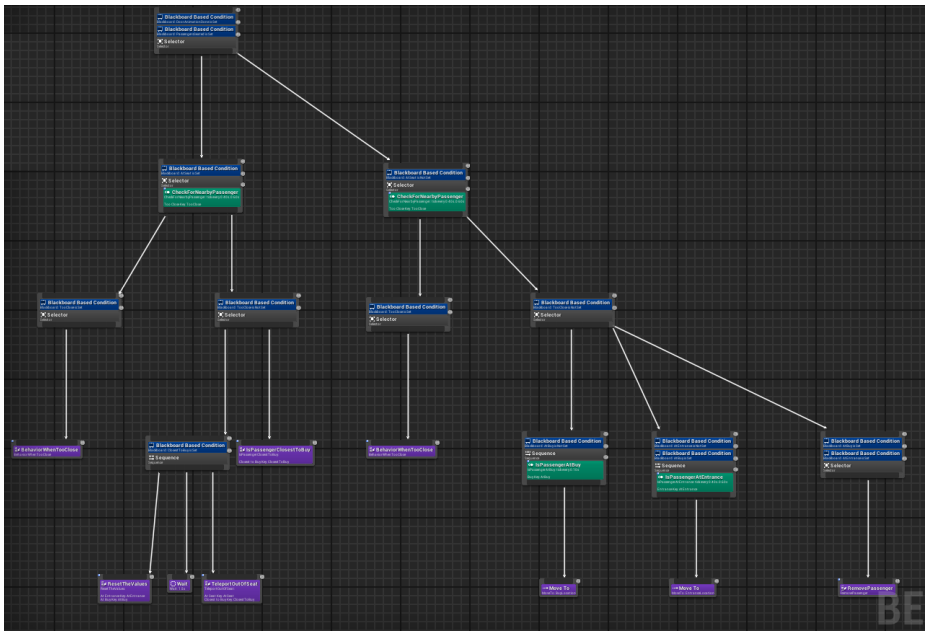

Figure 5.7: Stage three of the behavior tree

Figure 5.7 shows the third and last stage of the behavior tree. This stage is divided into two parts. The first part of the stage is to see if the passenger is allowed to stand up from the seat. The second part is to exit the vehicle. In the figure we can see the first branch to the left represents the first part and the second branch to the right represents the second part.

5.3 Runtime

This section will show three videos. It will start with an preview of the behavior tree. The second video is to show the behavior tree when it is running. The last video will show the simulation.

5.3.1 Preview on behavior tree

This video will go through the entire behavior tree to show how the behavior tree actually looks.

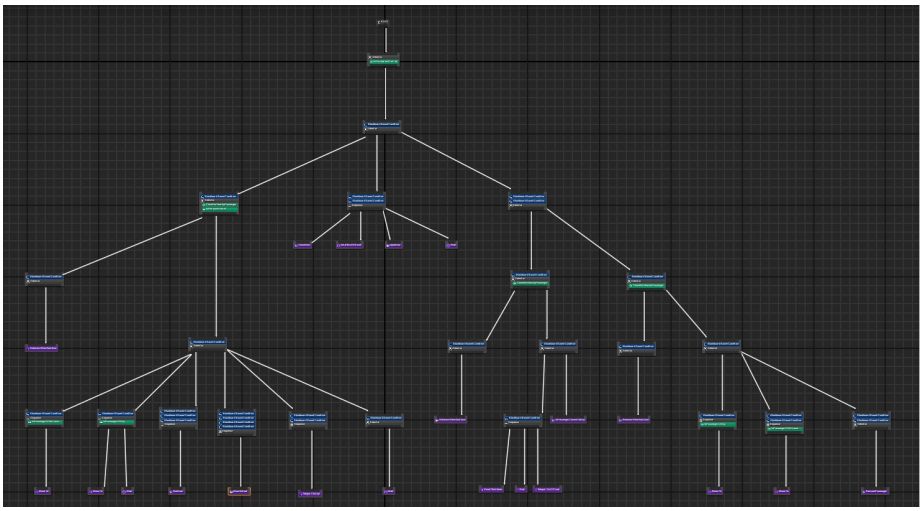

Figure 5.8: Preview video of the behavior tree (press image to open video). [7]

5.3.2 Runtime of behavior tree

This video will show the flow of the behavior tree when it is running on one of the passengers. It is also possible to see parts of the blackboard in the lower right hand corner, to see when the values changes depending on where the flow is.

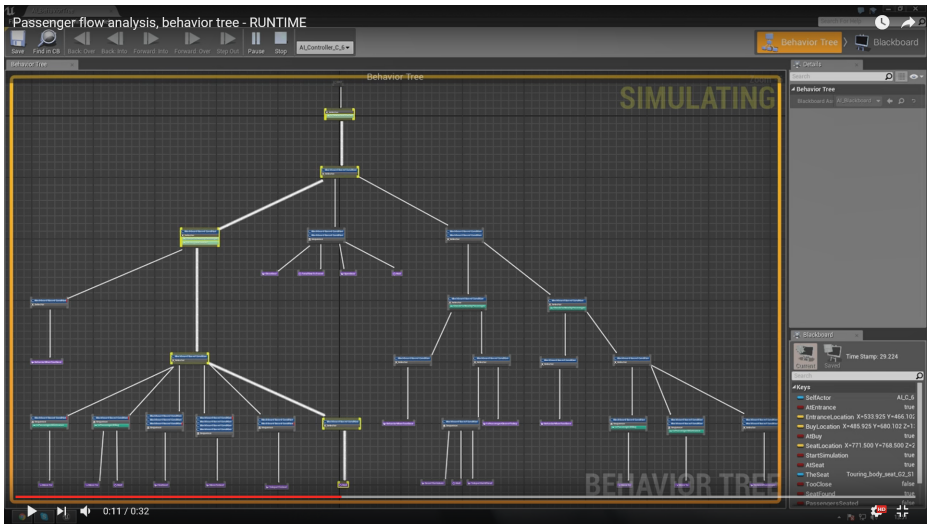

Figure 5.9: Runtime video of the behavior tree (press image to open video). [8]

5.3.3 The simulation

This video will show the passenger flow analysis. It will show how the controls work and how the simulation looks like.

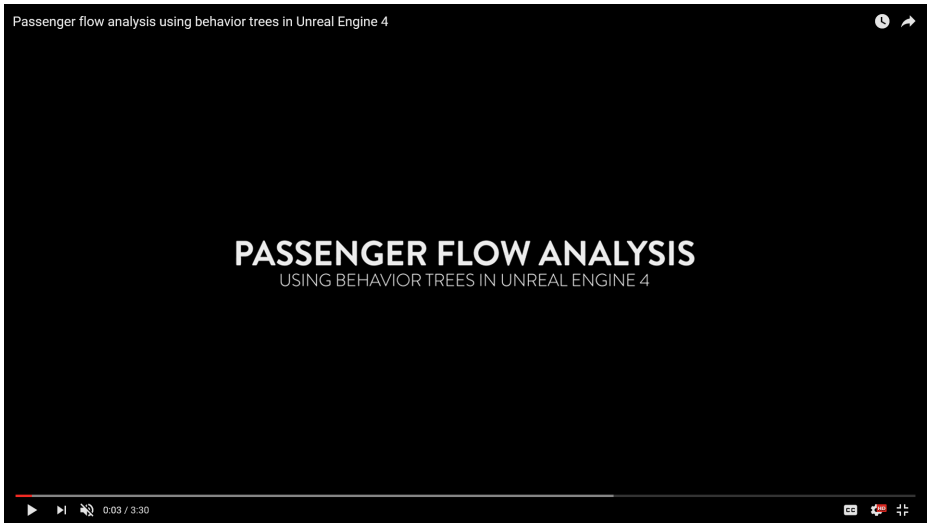

Figure 5.10: The simulation (press image to open video). [9]

Chapter 6

Discussion

This chapter will start discussing the methods used in the thesis. The chapter will end with discussing the results from chapter 5.

6.1 Unreal Engine 4

The usage of Unreal Engine 4 is for game development. In this case we use it to create a simulation. Even the company XperDi is using Unreal Engine 4 for doing CAD-modeling easier. If Unreal Engine 4 is made for creating games, then why are we using it. It all depends on what is considered as a game. According to Kramer, W [6], a game needs to have different criteria to be called a game. And if one of these criteria is not fulfilled, it is not a game. There is some truth behind this statement but the simulation created in this thesis will not fulfill all of the criteria. Does that mean that this thesis work is not a game, and is Unreal Engine 4 the wrong tool to use. Kramer, W has a point but a game is a bigger concept then he claims. I talk in chapter 4 that in order to make a simulation possible we need to create a game and rules for that game. And in reality that is exactly what is done. XperDi is also creating a game to make it easier for CAD-modelers. The point is that Unreal Engine 4 is more then a game developing tool. Unreal Engine 4 states [1] that the engine is a game developing tool. I would say that the engine is more then that. In other words, Unreal Engine 4 is a tool for producing interactive environments. This would explain the engine much better and also show that Unreal Engine 4 is not only for games but for different kinds of applications.

6.2 Behavior Trees

The method Behavior Trees brings a lot to the table. What makes the method exceptionally good is the ability to easily work with designers that does not have any knowledge about programming. For example the tasks and behaviors are programmed by the programmer and saved in nodes. When a designer wants to change the behavior tree they can use the tasks and behaviors how many times and how they want. This makes it easy for a programmer to work with a designer when implementing an AI. If the designer wants the AI to be able to jump, the programmer will simply create a task. This task can now be

reused whenever and wherever the designer wants to use it. This is what makes Behavior Trees such an easy method to use. An example in this thesis work is when entering and exiting the vehicle. We reuse the same tasks for moving in and out of the vehicle. The only difference is when exiting the vehicle, in this case we do the process in reverse. This makes it easier even for programmers, not only for designers.
Another aspect of why Behavior Trees are good is the debugging of the tree. For example if there is a problem in a big system it is nearly impossible to pinpoint the problem. By using something called breakpoints it can stop the AI at the specified node and then it is possible to check all the values in the blackboard. It is also possible to jump back and forth in time to see where the problem occurs. This makes it easy to find where problems occur in the Behavior Tree.
There are also some disadvantages with using Behavior Tree. When it comes to bigger systems it is really hard to navigate the Behavior Tree if it contains hundreds or even thousands of nodes. Then tool support is needed to be able to navigate. For example navigating programming code is quite easy because you read the code from the top, then going downwards. Even though the code can jump back and forth you know how it will jump back and forth. In a behavior tree the flow of an system can go back and forth and is not possible to read from the top and going downwards. This makes it difficult to navigate the behavior tree, especially when the behavior tree contains hundreds of nodes. This problem is resolved with Unreal Engine 4 because of its built in debugging system for the behavior tree.

6.3 Blackboard

When it comes to the blackboard we can see the information of each value at a specific time. It is possible to pause the behavior tree and look at the values and then jump back and forth in the tree to see where the flow is and what values are set in the blackboard. It is also possible to put breakpoints on each node to see when the flow will come to the specific node and then look at the blackboard values. In other words there is a complete debugging system for the behavior tree. This shows again why Unreal Engine 4 is a good tool to use in combination with behavior trees.

6.4 Runtime

This section will discuss the result from the videos. The two videos about the behavior tree will be discussed first and then the last video covering the simulation will be discussed.

6.4.1 Behavior Tree

We can see in the first two videos how the behavior tree looks and works. As mentioned before we can see how Unreal Engine 4 makes it easy for designers to work with behavior trees. The easy layout with the tree makes it easy to create new branches in the behavior tree. It is possible to drag and drop the already created services, decorators and tasks to easily create new behaviors.

This makes it easy for designers and programmers to work together, were the programmer makes the nodes and the designer creates the tree.
When it comes to the runtime of the behavior tree we can see the yellow flashing lines in the behavior tree. This line represents the flow of the behavior tree. This can be used to see if the behavior tree is behaving in the right way. This function including with the debugging system of the behavior tree is what makes Unreal Engine 4 a great tool to use. This will save time finding issues and problems in the the behavior tree. As mentioned before if the behavior tree has hundreds or even thousands of nodes it can be really hard to know how the flow of the system is. But using these functions that are built in Unreal Engine 4, will make it much faster and easier to pinpoint a problem in the tree.
We can also see how the blackboard values changes depending on what the passenger is doing. This can be used to see if the values are changed correctly depending on what nodes are called and where the flow is. This is also another way to pinpoint where a problem is occurring.

6.4.2 The simulation

The last video shows the simulation or in other words the passenger flow analysis. It is interesting to see in real-time how the behavior of the passengers are. Immediately after the simulation is started we can see how the passengers try to move to the entrance from different directions and how they behave towards each other. We can see that the behavior of all the passengers are the same. This is because they all have different instances of the same behavior tree. This means that the passengers think exactly alike. For realistic purposes this is not good. But from a backbone point of view this is a start. The goal was to create as much as possible for the company XperDi and make it easier for them to further develop the passenger flow analysis. How to fix this is to create more behavior trees with different flow and behaviors to make each passenger unique. This thesis work shows how it is possible to make an passenger flow analysis using behavior trees in Unreal Engine 4. The making of different behavior trees is for the designers who further develop the analysis.

Chapter 7

Future work

The thing with behavior trees are that they can always be improved and expanded. Future work would be to make the behavior tree more advance, for example adding functionality when another AI is too close. In this thesis the AI will stop and do nothing until it is possible to walk again. Some kind of algorithm can be implemented to make the AI walk around or find another path if it is possible. There are always room for more functionality to be added in behavior trees. This is one out of many future works that can be done.
Another aspect is to have different behavior trees for different passengers. In other words making passengers with different brains. In this thesis they all are thinking and reasoning the same because they have the same behavior tree. Adding different types of thinking for different passengers will create an more realistic feel to the simulation. In this case there are endless amounts of behavior trees that can be created.
What we can see is that no matter how much time is spent developing, there is always room for improvement and different kinds of behaviors. This is what makes behavior trees and AI in general a broad subject.
The idea for this thesis work was to implement and make as much as possible for the amount of time available. The company XperDi will further develop and improve this simulation if needed.

Chapter 8

Conclusion

The purpose of the thesis work was to create a simulation in Unreal Engine 4 and see how long it takes for passengers to enter and exit a vehicle. The company XperDi uses Unreal Engine 4 for their current product and it was demanded that the passenger flow analysis be created in Unreal Engine 4. The problem is that the engine is made for game development and not creating simulations. By creating a game and connecting the behavior trees in the engine it is actually possible to create anything in this engine. It all depends on what a game is. In other words the passenger flow analysis is a game were the player watches how AI pawns interact in a world. The tools included in the engine makes it easy to work with behavior trees. By looking at the results we can see that it is possible to create an simulation using the available tools. We can also determine that the more time spent developing, the more complex system can be created. And a more complex system results in a more realistic simulation. The thesis work was made for XperDi to make it easier for them to learn about behavior trees in Unreal Engine 4 and to make it easier to start implementing this further.

Bibliography

[1] What is Unreal Engine 4. (2016, July 14). Retrieved from https://www.unrealengine.com/what-is-unreal-engine-4

[2] Unreal Engine 4 API. (2016, July 20). Retrieved from https://docs.unrealengine.com/latest/INT/API/

[3] AIGameDev. (2016, July 14). Retrieved from http://aigamedev.com/

[4] G. Flrez-Puga; M. Gomez-Martin; B. Diaz-Agudo and P. Gonzalez-Calero (2008). Dynamic expansion of behaviour trees.

[5] J. Laird and M. VanLent (2001). Human-level AI's killer application: Interactive computer games.

[6] L. Kramer (2000) What is a Game? Retrieved from http://www.thegamesjournal.com/articles/WhatIsaGame.shtml

[7] Passenger flow analysis, behavior tree - PREVIEW. (2017, Mars 8) Retrieved from https://www.youtube.com/watch?v=QDni0MX5LDk

[8] Passenger flow analysis, behavior tree - RUNTIME. (2017, Mars 8) Retrieved from https://www.youtube.com/watch?v=S6ov33RvNQ4

[9] Passenger flow analysis using behavior trees in Unreal Engine 4. (2017, Mars 8) Retrieved from https://www.youtube.com/watch?v=vsziCigHLPI

[10] D. Curcio; F. Longo; G. Mirabelli and E. Pappoff (2007, June). Passengers flow analysis and security issues in airport terminals using modeling & simulation.

[11] R. Liu and S. Sinha (2007). Modelling Urban Bus Service and Passenger Reliability.