NTNU
Norwegian University of
Science and Technology

Simulating Cyclists in a Simulator with the use of Behaviour Trees

Rune Nilsen Abrahamsen

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Jo Skjermo, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This thesis will present the work done by Rune Abrahamsen for his Master's Thesis in Game Technology. The project was done in the first half of 2016 at the Norwegian University of Science and Technology. The goal of the project was to make an autonomous cyclist agent. This agent should simulate human behaviours for a cyclist, by using behaviour trees. This meant that the autonomous cyclists had to behave realistically in different traffic scenarios.

The project used an existing simulator that was made in Unreal Engine 4. It was made by the author in a project at NTNU in the last half of 2015. The simulator included a cyclist character, which could be used for this project. Using the simulator also meant that one could use Unreal Engine's implementation of behaviours trees. The focus of the project was therefore on making an AI agent that could control the cyclist character and then testing the AI in different traffic scenarios.

The thesis starts by explaining the theory behind the AI agent. This includes a study of cyclist laws, rules and behaviours in Norway, and a state-of-the-art explanation of behaviour trees in games. After this, the thesis explains how this theory is used to implement an AI agent in Unreal Engine 4. This includes an explanation of how the AI agent controls the cyclist model and how behaviour trees are used to simulate different behaviours.

The analysis of the results show that the project has successfully made an autonomous cyclist agent. The AI agent can control the cyclist model and simulate behaviours for a child, transport or leisure cyclist. This is categories of cyclists defined in the project. Each cyclist type was simulated by the AI agent. Then the cyclist types participated in several tests, in different traffic scenarios. The tests are defined and implemented as part of this project, and is explained in detail in the thesis. The test results have been recorded and analysed. The videos, together with the implementation, was then used to conclude the project.

Sammendrag

Denne oppgaven vil presentere Rune Abrahamsens masteroppgave i spillteknologi. Prosjektet ble gjennomført i første halvdel av 2016 ved Norges teknisk-naturvitenskapelige universitet (NTNU). Målet for prosjektet var å lage en autonom syklist agent. Denne agenten skulle simulere menneskelig atferd for en syklist, ved hjelp av atferdstrær. Dette betydde at de autonome syklistene måtte oppføre seg realistisk i ulike trafikkscenarier.

Prosjektet brukte en eksisterende simulator som er laget i Unreal Engine 4. Denne ble laget av forfatteren i et prosjekt ved NTNU i siste halvdel av 2015. Simulatoren inkluderte en syklistmodell, som kunne brukes i dette prosjektet. Dette betydde også at man kunne bruke Unreal Engines implementering av atferdstrær. Prosjektet kunne derfor fokusere på å lage en AI-agent som kunne kontrollere en syklistmodell, og deretter teste denne AI-en i ulike trafikkscenarier.

Oppgaven starter med å forklare teorien bak AI-agenten. Dette omfatter et bakgrunnsstudie av sykkel-lover, -regler og -atferd i Norge, samt en «state-of-the-art» studie av atferdstrær i videospill. Deretter forklarer oppgaven hvordan denne teorien kan brukes til å implementere en AI-agent i Unreal Engine 4. Dette inkluderer en forklaring på hvordan AI-agenten styrer syklistmodellen og hvordan atferdstrær brukes til å simulere forskjellige oppførsler.

Analysen av resultatene viser at prosjektet har resultert i en autonom syklistagent. AI-agenten kan kontrollere syklistmodellen og simulere atferd for disse syklisttypene: barne-, transport- og fritids-syklist. Dette er syklisttyper som er definert i prosjektet. Disse syklisttypene har deltatt i flere tester, satt i ulike trafikkscenarier. Testene er definert og implementert som en del av prosjektet, og blir forklart i detalj i oppgaven. Det er tatt opptak av testene. Videoene, sammen med implementeringen, ble så brukt til å konkludere prosjektet.

Keywords

3D environment	A reference to a virtual 3D world in which the simulator is made.
Actor	An object in a 3D environment which can be controlled by an agent or a human.
AI Agent	An autonomous entity responsible for some task. For example, controlling a bicycle.
AI	Abbreviation for Artificial Intelligence.
BT	Abbreviation for Behaviour Tree. See Chapter 2 for an explanation of behaviour trees.
CBR	Abbreviation for Case-Based Reasoning
HFSM	Abbreviation for Hierarchical Finale State Machine
NPRA	Abbreviation for the Norwegian Public Roads Administration.
NTNU	Abbreviation for the Norwegian University of Science and Technology.
Personality	Refers to how an autonomous cyclist behaves. The characteristics of the behaviours defines that cyclist's personality.
Simulator Framework	A Simulator Framework refers to a simulator and the related tools needed to used or modify the simulator.
UE4	Abbreviation for Unreal Engine 4. See Chapter 2 for a very basic introduction to Unreal Engine 4.

Table of Content

List of Figures

List of Tables

1 Introduction

This thesis presents the work done by Rune Abrahamsen as part of his Master's Thesis in Game Technology. The work was done during the first half of 2016 at the Norwegian University of Science and Technology. The project's goal was to simulate human behaviours for a cyclist in a simulator. This resulted in an AI agent, which can be used to control a cyclist autonomously in a simulator. The thesis gives a thorough explanation of the project. It contains a theoretical explanation of the technologies used, an explanation of the research method, an explanation of the AI implementation, the results of the research, and a discussion about the findings. The rest of this chapter will give some motivations for the project and give a more detailed project description.

1.1 Motivation

In the Autumn of 2015, the author worked on a prototype bicycle simulator[1]. That work was part of his fifth year specialization project at NTNU. That project was an introduction to this thesis. After finishing the prototype, there were several things missing to get a useful simulator framework[2]. One of these was the lack of autonomous actors which could populate the 3D environment. This project will therefore try to solve this by simulating believable cyclists.

The existing simulator can be used to set up different traffic scenarios. Right now, a human player must control all of the actors in these scenarios. It would be beneficial to have an AI that also can control these actors. This could mean anything from simple path-following to more complex behaviours. An example could be a cyclist interacting with traffic and following the traffic rules. The simulator does not have this AI functionality yet, but it got the essential building blocks[3]. It is therefore interesting to learn how to use these built in features, and try to extend them to this use-case.

The growth in Norway's cyclist traffic should increase by 3%. Today all commutes consist of 5% cyclists, by 2023 this should increase to 8%. This would mean around an 100% increase in cyclists, due to an overall increase in commutes. In other words, cyclists in the biggest cites in Norway must increase by more than 100%. For example, this means that 80% of children and young adults must walk or cycle to school. There are several ways to achieve this, but a prediction is that the cyclists' safety and navigability must be increased along their travel-routes. This information is found in the Norwegian National Transport Plan (The Norwegian National Transport Plan, 2016). Transportation planners and researchers must plan for this increase when they make the streets of tomorrow. To have a tool to help them visualize and test different scenarios can help them make better and safer solutions.

It is going to be more and more cyclists, due to the above mentioned traffic-increase. The simulator can therefore be used to train children or less experienced cyclists in more dangerous situations. The cyclists can learn what they should do in different scenarios. This

[1] This is a 3D game where one can use a real bicycle to control a virtual cyclist. See Chapter 2 for a better explanation.
[2] A Simulator Framework refers to a simulator and the related tools needed to used said simulator. In this case, it means the simulator presented in Chapter 2 and the Unreal Engine 4 Editor.
[3] The building-blocks are a part of Unreal Engine 4's implementation. These must be adapted and extended for this use-case.

will give them valuable experience in a safe environment. For example, a cyclist that has not cycled on the mixed roads before might view it as dangerous. They might therefore not use their bicycle to and from work if using the mixed road is their only option. They might start to cycle to and from work if they gained some more experience. The simulator can help a cyclist to train for this, or similar, scenarios. But for this to be possible, one needs autonomous actors which can simulate the traffic in these scenarios.

1.2 Project Description

The project's goal is to simulate human behaviours for cyclists in a simulator setting. This means that an autonomous agent will control a cyclist actor in a 3D environment. The cyclist should behave realistically in different traffic scenarios, based on their personality. The project will use a bicycle simulator made in Unreal Engine 4. The simulator provides a basic cyclist model, but it lacks a proper 3D environment and the autonomous agents. There are three major tasks in the project:

1. Improving an existing cyclist model, so it can be controlled by an AI.
2. Implementing an autonomous cyclist agent in the simulator.
3. Making the 3D environment within the simulator.

The project will consist of these research topics:

- Understanding cyclist behaviours, rules and laws.
- Modelling and using 3D models.
- Working with and scripting in Unreal Engine 4.
- Using behaviour trees as an AI technique in games.

1.2.1 Research Goals

The overall research goal is to make an autonomous agent that simulates human behaviours for cyclists in a 3D simulator setting. This entails thee following research questions:

Question 1: How can realistic cyclist behaviours be simulated?

- What separates different cyclist's behaviours from each other?
- Can one categorize different cyclists into groups based on behaviours?

Question 2: What is a good simulation of cyclist behaviours?

- When is a simulation of cyclist behaviours good enough?
- What defines a good or bad simulation?
- How can one test the quality of a cyclist's behaviour?

Question 3: Which technologies must work together to make an autonomous cyclist agent?

- Which technologies are necessary to control the cyclist?
- Which technologies can be used to simulate different behaviours?
- What degree of connection is needed between the environment and the agent?

1.3 Thesis Outline

Chapter 1 has given an introduction to the thesis and its goals. Chapter 2 will give a literature study of the different topics and technologies used in this project. Chapter 3 will present the research method. This includes how the research will be conducted and evaluated. Chapter 4 will look at the implementation of the whole project. Chapter 5 presents the results from the project’s tests. Chapter 6 will discuss the result of the project and then concludes the project. Finally, chapter 7 will present some future works based on this project.

2 Literature Study

It is important to understand the different topics and technologies used in the project. This chapter will therefore present a literature study on each topic or technology. Section 2.1 give an introduction to the bicycle simulator used in the project. Section 2.2 will present the basics of behaviour trees. Section 2.3 will look at how artificial intelligence (AI) can be implemented in Unreal Engine 4. Finally, section 2.4 will look at cyclist behaviours and cyclist laws in Norway.

2.1 Bicycle Simulator

This section will give an overview of the bicycle simulator that will be used in the project. The simulator contains a virtual cyclist model. This is the cyclist that the AI agent will try to control. The simulator is made in Unreal Engine 4, which includes some features that will be useful in this project. The simulator was made as part of the author's fifth year specialization project[1] at NTNU. The simulator uses a real bicycle as a controller to control a cyclist in a virtual world. The simulator can be split up in two parts: The virtual cyclist model and the controller. Each part communicates together with simple text messages over a USB-cable.

2.1.1 The Virtual Cyclist Model

The cyclist model is made in Unreal Engine 4. Unreal Engine 4 is a modern 3D game engine. It allows a user to make real-time 3D environments, with very little effort. Setting up a basic 3D environment is as simple as clicking a button. The real work starts when one wants to make actors interact with the environment, often is a very specific way. The 3D environment's goal is to give the user a visual representation of a cyclist. This cyclist should interact with the 3D environment as realistically as possible. This means that the visual cyclist should reflect a cyclist from the real world.

Unreal Engine 4 comes with a physics system that allows basic simulation of real world physics. This system allows simulations such as: torque, friction and gravity, which is needed for simulating a cyclist. Unreal Engine also comes with an advanced vehicle system. This allows advanced simulation of vehicles. It includes options such as: wheel physics, advanced engine physics, and transmissions. The simulator used these systems to make the cyclist model.

The cyclist model in the simulator can be described as a semi-realistic model of a real cyclist. The cyclist model behaves much like a real cyclist on flat terrain, but it needs some improvements when there are hills or very uneven terrain. The model also has some simplifications when it comes to balancing and handling. This should not be an issue in this project. Figure 2-1 shows a picture of the virtual cyclist model. Everything is animated, so the wheels, handlebar, pedals and the rider moves when the bicycle moves.

[1] The specialization project is part of an obligatory course at NTNU for students taking their ninth semester. The student works with a task related to their Master's specialization-field. This project often serves as an introduction to a possible Master's thesis on the subject the next semester.

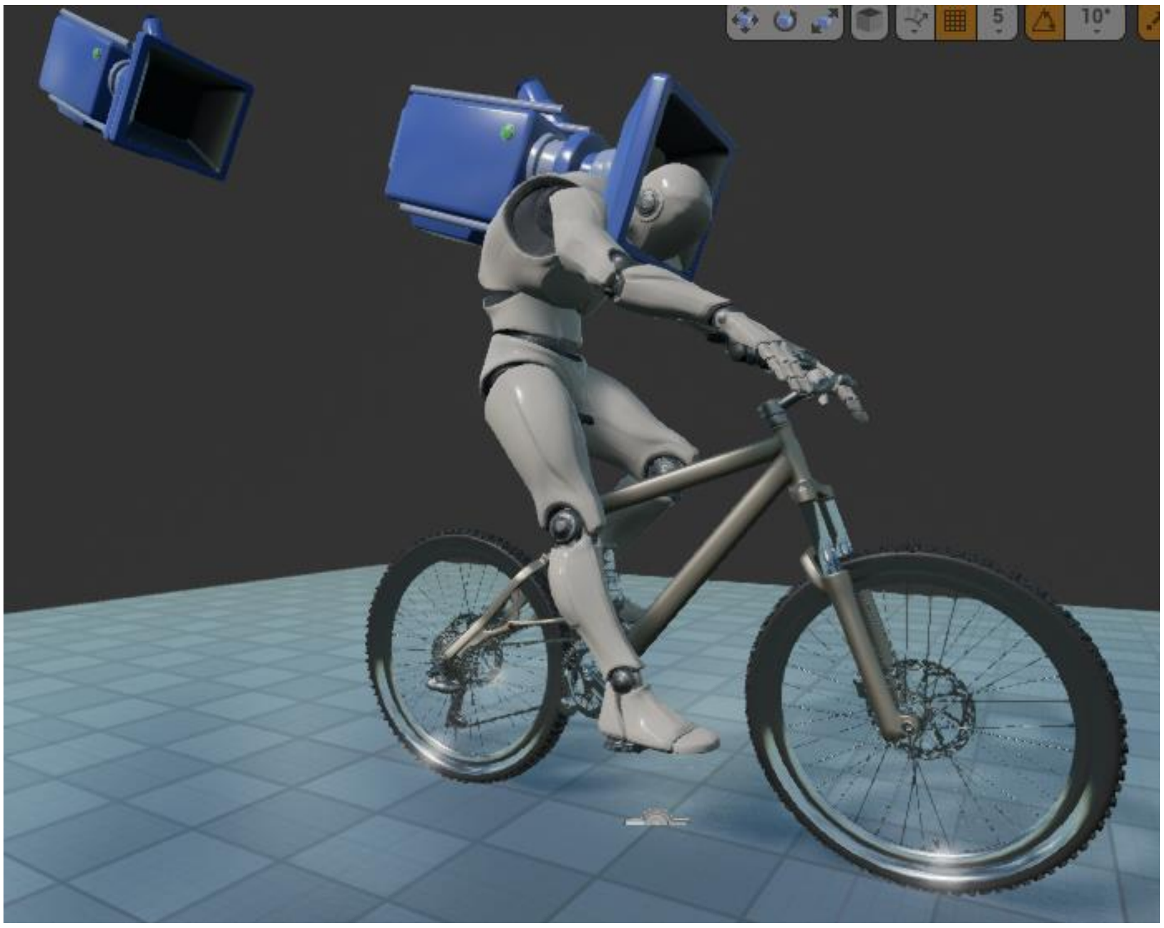


Figure 2-1 – The virtual cyclist model. The blue cameras are used to get a first- or third-person view when cycling.

2.1.2 The Controller

The controller is a real bicycle that can control the cyclist model in the simulator. See figure 2-2. The controller is one of the things that an AI agent must automate in this project. This will be done by removing the hardware all together, but the controller can still be used by human players. For example, if a human player cycles together with AI cyclists.

The controller is made up of a spin-cycle (indoor exercise bicycle) and an Arduino[1]. It allows the rider to use the pedals and several buttons to control the virtual cyclist model. The buttons allow the rider to change gears, turn the handlebar and apply the brakes. The virtual cyclist reacts to these inputs.

[1] A small computer which can easily be programmed and modified for different purposes. See https://www.arduino.cc/ if you are interested.

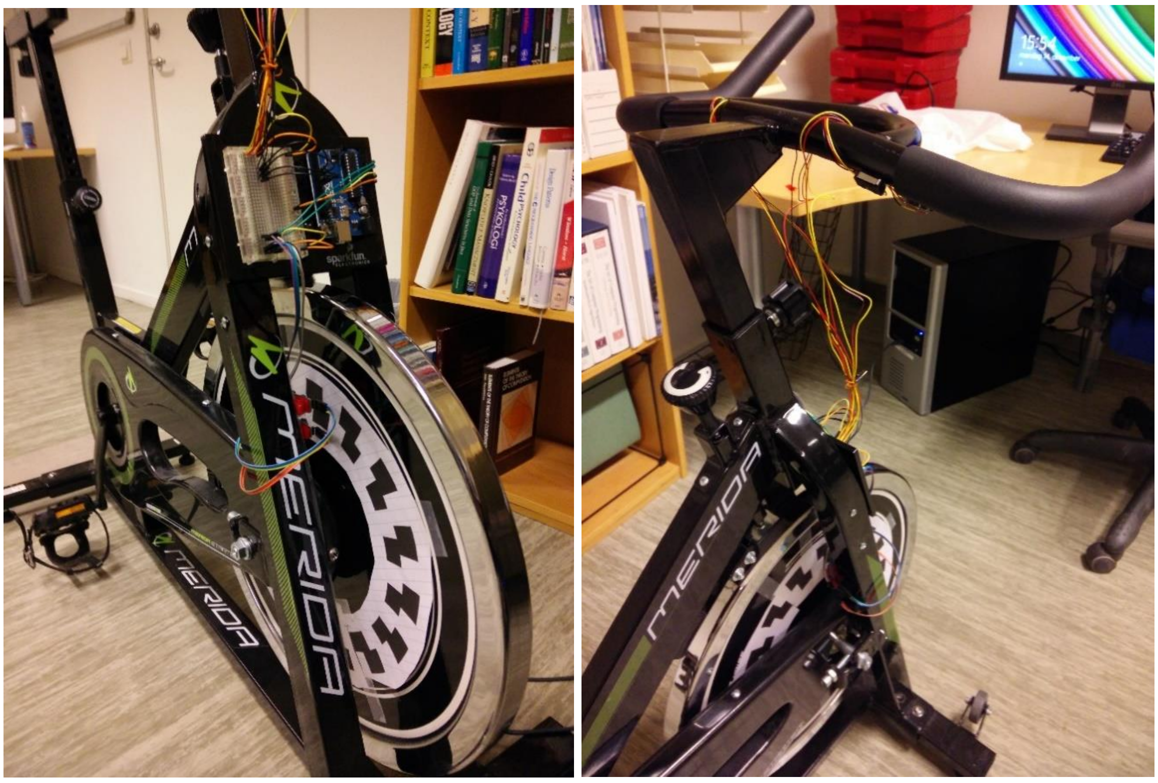


*Figure 2-2 – The controller. The **left** picture shows the Arduino and the breadboard that connects all the wires. One can also see the sensors that measures the speed of the front wheel. The **right** picture shows the handlebar. You can see one of the buttons on the right side. This picture got two buttons on each side, but one can add as many as one needs.*

2.2 Behaviour Trees

Behaviour trees is a core technology used in the project, so a good understanding of it will be useful. This section explains the core concepts of behaviour trees, how behaviour trees have been used in games, and the state-of-the-art of behaviour trees in games. The information in this section is based on work done by Champandard (2008), Ji and Ma (2014), Florez-Puga et al. (2008), and Epic Games (2016), Champandard (2007c), Champandard (2007b), Champandard (2007a).

2.2.1 Core Concepts

For the last few years, behaviour trees have been the major formalism used in game industry to build complex AI behaviours. This success comes from the simplicity to understand, use and develop behaviour trees by non-programmers. Behaviour trees have been proposed as an improvement over Hierarchical Finite-State Machines (HFSM) for designing game AI. Their advantages over traditional AI approaches are being simple to design and implement, scalability when games get larger and more complex, and modularity to aid reusability and portability.

Finite-State Machines

Behaviours trees are said to replace finite-state machines, but what is a finite-state machine? Champandard (2007b) explains it like this:

"*A finite state machine is based on the concept of a state, which typically consists of two things:*

1. *A set of actions running at the same time (e.g. playing an animation, a sound, or waiting for a certain amount of time).*
2. *A set of transitions with a conditional check to determine when to engage the next state.*

States can be made quite generic and robust by adding many transitions to support all the desired cases. For simple problems, this is fine, but for large problems you need a more scalable approach."

Finite-state machine is typically represented by a state diagram like in figure 2-3.

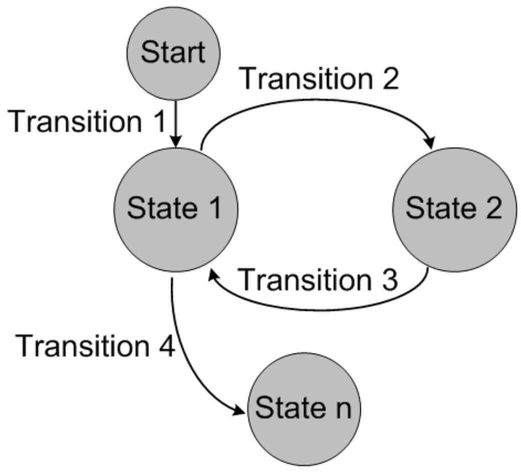


Figure 2-3 – A theoretical example of a finite-state machine.

As Champandard stated, FSM does not scale very well. This issue was first solved by what is called a hierarchical finite-state machine.

Hierarchical Finite-State Machines

Hierarchical finite-state machines are a more advanced version of the FSM. This is what Champandard (2007a) says about HFSMs:

"Hierarchical finite state machines offer some help for reusing logic. The design process is very similar to non-hierarchical finite state machines:

1. *You build the logic state by state, connecting them up with transitions bottom up.*
2. *While creating new states, you may group them together to share transitions.*

This gives you a simple way to avoid duplicating transition logic.

One can use the term super-states to indicate groups of states. These super-states too can have transitions. This theoretically allows you to prevent redundant transitions by applying them only once to super-states, rather than each state individually…

…HFSM certainly provide a way to reuse transitions, but it's still not an ideal solution. The problem is that:

- *Reusing transitions isn't trivial to achieve, and requires a lot of thought when you have to create logic for many different contexts (e.g. dynamic goals, actor status).*

- *Editing transitions manually is rather tedious in the first place."*

As Champandard said, this is still not a very good solution. Another solution is to focus on making individual states modular. This way they can be easily reused in different parts of the AI logic. This is what behaviour trees try to do.

Behaviour Trees

Behaviour Trees increase the modularity by encapsulating logic transparently within nodes. These nodes should have no external transitions, so they become self-contained. A node is now just a behaviour and needs no transition to another state.

Champandard (2007c) tries to explain this with programming terms like this:

"You may ask, since the transitions are no longer encoded explicitly in each state, how is it possible to sequence behaviours using a behaviour tree?

This is done using the same method as most programming languages — which is demonstrably more scalable. In C++, Python or any other modern language, each operation does not contain a pointer to its successor, as it would be difficult to reuse these operations in different contexts (like calling them from another function).

Instead, operations are inserted into a parent scope (e.g. in a function, or for a conditional check), and they are executed according to the semantics of the parent scope. So in essence, the transitions between the operations are automatically defined based on which type of parent scope is chosen."

In a behaviour tree, the nodes are nested within each other and thus forms a tree-like structure. This also restricts transitions to only these nested nodes. The root node branches down to more nodes until the leaf nodes are reached. These leaf nodes are the base actions that define the behaviour of the AI. Essentially, a node can be seen as a high level AI behaviour. The links to the nested child nodes define sub-tasks, which make up the main behaviour. The leaf nodes then become a group of basic actions that define a behaviour.

The nodes of a behaviour tree are formally constructed out of two classes of constructs: leaf nodes and composite nodes. The next subsections will describe these in detail. See figure 2-4 for a visualization of a basic behaviour tree.

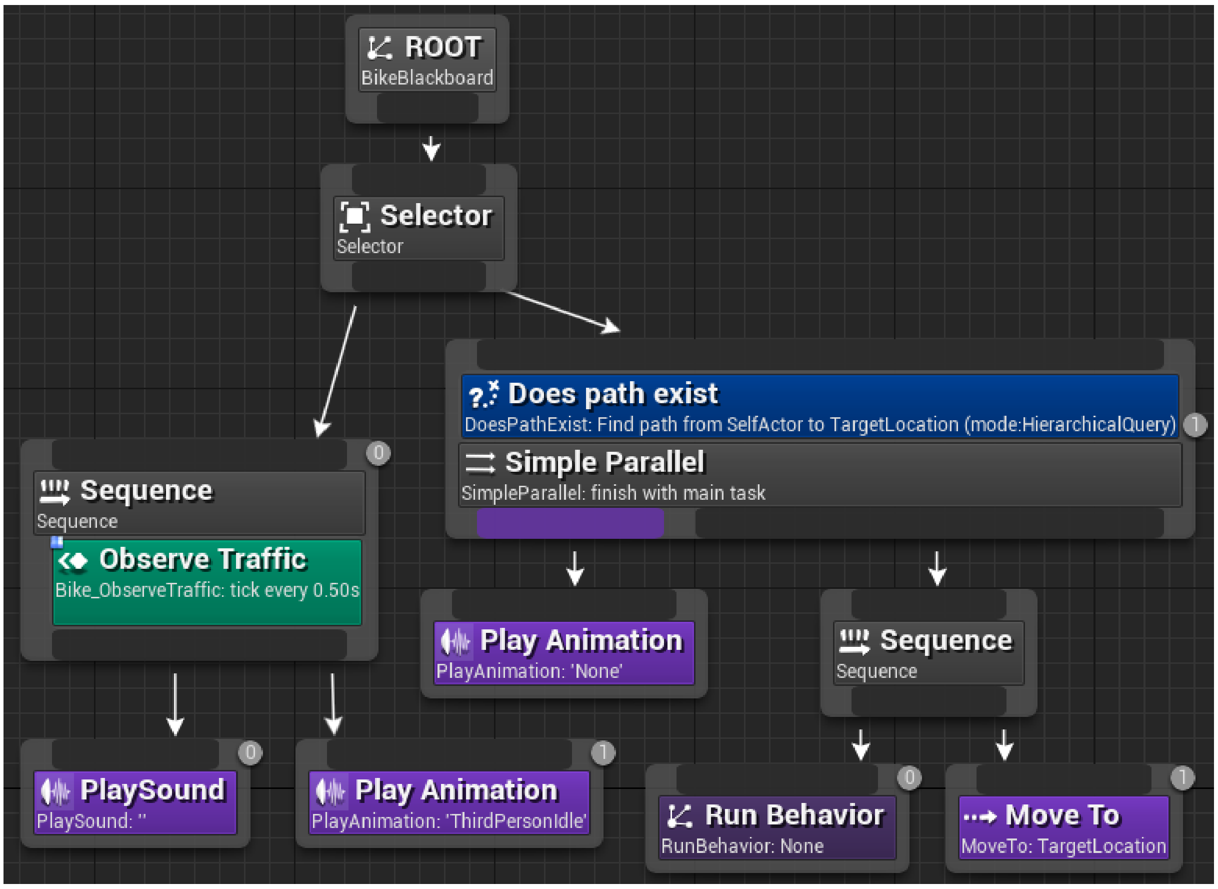


Figure 2-4 – A Behaviour Tree implemented in Unreal Engine 4. This serves as an illustration of one way to implement BTs. The differences between the theory in this section and Unreal Engine 4's implementation of BTs are discussed in section 2.3.

2.2.2 Leaf Nodes

Leaf nodes are the terminal nodes of the tree and define low level actions which describe the overall behaviour. They are typically implemented by user code, for example a script. They can be something as simple as looking up the value of a variable in the game state, executing an animation, or playing a sound effect. See the purple nodes in Figure 2-4.

Leaf nodes consist of two types:

- **Actions** – Actions often cause the execution of methods or functions on the game world. E.g. move a character or decrease health.
- **Conditions** – Conditions usually query the state of objects in the game world. E.g. location of character or the amount of health.

Leaf nodes usually have two end conditions. These are used to determine if the task was completed successfully or if it failed. The result is sent back to the parent node when a node reaches a condition. The two end conditions are:

- **Failure** – This means the task have failed.
- **Success** – The means the task has succeeded

Some implementations of BTs also include other end conditions.

2.2.3 Composite Nodes

Composite nodes provide a standard way to describe relationships between child nodes. For example, how and when a child node should be executed. Contrary to leaf nodes, which are defined by the user, composite nodes are predefined and provided by the behaviour tree formalism. They allow you to build branches of the tree in order to organize their sub-nodes. Basically, branches keep track of a collection of child nodes.

There are normally only a handful of composite nodes. Because with only a few different grouping behaviours, one can build complex behaviours. Normally the composite nodes consist of **selectors**, **sequences**, **parallels** and **decorators.**

Selectors

A selector is a branch node that runs each of its child behaviours in turn. See the “Selector” node in Figure 2-4. It will return immediately with a success status when one of its children runs successfully. As long as its children are failing, it will keep on trying. If it runs out of children completely, it will return a failure status.

Selectors are used to find the first successful child of a set of choices. For instance, a selector might represent a character wanting to reach safety. There may be multiple ways to do that: take cover, leave a dangerous area, and find backup. Such a selector will first try to take cover; if that fails, it will leave the area. If that succeeds, it will stop since there is no point in also finding backup. If all options are exhausted without success, then the selector itself has failed.

Sequences

A sequence is a branch node that runs each of its child behaviours in turn. See the “Sequence” nodes in Figure 2-4. It will return immediately with a failure status when one of its children fails. As long as its children are succeeding, it will keep going. If it runs out of children, it will return in success.

Sequences represent a series of tasks that need to be undertaken. For example, there are several steps that needs to be taken to reach safety behind a cover. To find cover you will need to choose a cover point, move to it, and play an animation to hide behind it. If any of the steps in the sequence fails, then the whole sequence has failed.

Decorator

The name “decorator” is taken from object-oriented software engineering. The decorator pattern refers to a class that wraps another class, modifying its behaviour. If the decorator has the same interface as the class it wraps, then the rest of the software doesn’t need to know if it is dealing with the original class or the decorator.

In the context of a behaviour tree, a decorator is a node that has one single child node and modifies its behaviour in some way. You could think of it like a composite node with a single child. See the “Does path exist” decorator node, with the child “Simple Parallel”, in Figure 2-4. There are many different types of useful decorators. Some of them are:

- **Always Fail** – This will always fail no matter the if wrapped task fails or succeeds.
- **Always Succeed** – This will always succeed no matter if the wrapped task succeeds or fails.
- **Include** – This copies an external subtree. This decorator enhances behaviour trees with modularity and reusability.

- **Limit** - This controls the maximum number of times a task can be run.
- **Repeat** – This will repeat the wrapped task a certain number of times, possibly infinite.
- **Until Fail** – This will repeat the wrapped task until that task fails, which makes this decorator succeed.
- **Until Success** – This will repeat the wrapped task until that task succeeds, which makes this decorator succeed.

These are some of the possible decorators. Decorators can be defined by the user, so it is only limited to what the user needs.

Parallel

A parallel composite node handles concurrent behaviours. It is a special branch node that starts or resumes all children every single time. See "Simple Parallel" node in Figure 2-4. The actual behaviour of this node depends on its policy:

- **Sequence policy** – The parallel node fails as soon as one child fails; if all its children succeed, then the parallel node succeeds. This is the most common policy.
- **Selector policy** – The parallel node succeeds as soon as one child succeeds; if all its children fail, then the parallel node fails.

Here are some typical uses of the parallel nodes:

- **Non conflicting actions** – The parallel node is most obviously used for sets of actions that can occur at the same time. One might, for example, use parallel nodes to have a character roll into cover while shouting an insult and changing primary weapon.
- **Condition checking** – One very common use of the parallel node is to continually check whether certain conditions are met while carrying out an action. For instance, one can make a character react on events while sleeping or wandering. Using parallel nodes to make sure that conditions hold is an important use-case in behaviour trees. With it one can get much of the power of a state machine. In particular, the state machine's ability to switch tasks when important events occur and new opportunities arise.
- **Group behaviour** – One can use parallel nodes to control the behaviour of a group of characters, such as a fire team in a military shooter. While each member of the group gets its own behaviour tree for its individual actions (shooting, taking cover, reloading, animating, and playing audio, for example), these group actions are contained in parallel nodes within a higher level selector that chooses the group's behaviour. If one of the team members can't possibly carry out their role in the strategy, then the parallel node will return in failure and the selector will have to choose another option.

2.2.4 Evolution of Behaviour Trees in Games

According to AiGameDev.com (Champandard, 2012) and Epic Games (2016) there have been two generations of behaviour trees in games. The first generation implements the basic behaviour tree that has been presented so far in this chapter. It got the root node, the different kind of composite nodes and the two leaf nodes. The problem with this implementation is that it struggles to keep up with the size and complexity of modern demands. The first generation of BTs tries to improve upon this by introducing shared tree nodes. In this way several characters can use the same behaviour tree, but store their data separately. This saves some memory, but does not reduce the runtime for the BTs.

The second generation of BTs does not introduce more composite or leaf nodes. As mentioned earlier the existing nodes have several ways to express complex behaviours. The second generation are interested in solving the computational demands of the BTs. The less resources they use the better they are for a game. Both memory and processing time is of interest. The improvements have so far gone in two directions, but are predicted to be merged together for the best possible representation. The second generation is the current "state-of-the-art" and will be presented and discussed in the next subsection.

2.2.5 State-of-the-Art

This subsection will take a close look at the state-of-the-art of behaviour trees. The current (2nd) generation of BTs are split into two different approaches. Both brings a unique improvement to the previous generation, but due to the differences in the implementations they have yet to be combined. The two approaches are can be called **event-driven** BTs and **data-driven** BTs. They both evolve around the same idea; They want to improve the way the code interprets the BT. This means how to BT is read or traversed by the underlying compiler or game engine.

Data-Driven Behaviour Trees

This approach is a "brute force" solution to the problem. It is designed to reduce the number of function calls, account for every possible bit of the code, and control all memory accesses. It still functions in the same way as the first generation, by starting at the top and traversing the tree.

This technique is done by using a stack allocator for the whole BT. This works by continuously adding the child nodes of a node to the stack, often in a depth-first order. This means that child nodes comes after their parent. By using this you get a smart way to access a child node, even though one must start at the top of the BT each run.

Instead of using pointers to the next node in the BT, you can now use relative offsets in the stack to access children. This can save a lot of memory. E.g. by going from a 64- or 32-bit pointer to a 16- or 8-bit index value.

Other optimizations can also be done. They are all dependant on how the BT is implemented in a given game. The advantages of these improvements are that they are easy to do. One just applies common optimization wisdom. The disadvantages are that the code becomes harder to debug, and it becomes harder to make the code look normal. This solution is overall very good, but by using domain knowledge of BTs one can further improve the code. This is done in the event-driven BT implementation.

Event-Driven Behaviour Trees

This approach uses domain knowledge of how BT works. The idea is to optimize the architecture or algorithm of the BT; To get the same results with less work. The optimizations are gained by the following ideas.

On the **first** traversal of the BT each node is expanded depth-first; Everything works like the previous implementations. One the **subsequent** traversals one does not start from the root node anymore. This works by allowing child nodes to broadcast their status. For each run of the BT you jump to the active node. For example, an action node. When it finishes it broadcasts its status to the parent node, then the parent can either also finish or expand to its next child.

To do this one can still use a queue of active nodes. The parent must be able to insert a child in the front of the queue. The parent must also be able to terminate its children. This allows for some unique features that will be explained in next section of this chapter. On each run of the BT you just jump to the first task in the queue. If the task is finished it returns its status to the parent and is removed from the queue; If the task is still running, you do nothing and wait for the next call to the BT. The benefits of this solution is that one gets a guaranteed minimal computation.

2.2.6 Hybrid Solutions

Another approach to improving behaviour trees could be to use a hybrid solution. This involves using either case-based reasoning or HFSM to help select the best BT. A short explanation of these solutions will be given, but they are not relevant for the rest of this thesis.

Case-Based Reasoning and Behaviour Trees

This solution is based on the work done by Florez-Puga et al. (2008). By combining case-based reasoning (CBR) and BT one can make what is called a Dynamic Behaviour Tree (DBT). CBR is an expert system. It uses a database of previous cases to determine the best solution to the current task. By combining this with BTs, one can store sub-trees in a database. One can then use the information available to query the CBR for the best sub-tree to solve a problem. In this way one can have a large set of sub-trees stored in the database and reduce the BTs size in a game. Since the CBR can evolve by learning, this means that a games AI also could improve over time.

To make this possible one can use the **decorator** composite node. By using the **include** decorator, one can add a sub-tree into the currently running BT. All one would need in addition to this is a way for the AI to query the CBR. The problem with this solution is that it makes the AI less predictable. Game designers often want the AI to behave in a certain way. This makes the outcome of a scenario known in all cases. By using DBT one can get variations in the ways a non-player character behaves, depending on what the CBR proposes as a solution. In this paper this would not matter. Here I am more interested in an AI agent that simulates as good a cyclist as possible. This DBT technique could therefore be an interesting way to achieve that.

Hierarchical Finale-State Machines and Behaviour Trees

A hybrid system with HFSM and BT might be a good solution for several situations. A BT can replace a HFSM, but in some cases one could use both. The two most reasonable solutions would be:

- Character have multiple BTs and use a state machine to determine which tree they are currently running.
- Certain tasks in the behaviour tree can act like a state machine, detecting events and terminating the current sub-tree to begin another.

Both case-based reasoning and HFSMs could be used to select between different behaviours in this project. This would depend on how different each personality is. It could be a good idea to use one of these techniques, if the different personalities require their own sub-BT.

2.3 Artificial Intelligence in Unreal Engine 4

Unreal Engine 4 is a game engine written in C++ by Epic Games. It is an open-source engine and the creators relies on a royalty system to earn money. As with most game engines, it gives developers a framework and a set of tools to design their games in an easier manner. UE4 provides a graphical user-interface for creating a game. This can be used to make and manipulate a virtual world and objects in it. The information in this section is based on the documentation of Unreal Engine 4 by Epic Games (2016).

What makes Unreal Engine 4 unique is it's visual scripting system (or blueprint system). This system replaces simple code scripts in a game with a tree-like, visual script. This makes it possible to visually debug the script in real-time, when the game is running. It also makes code more understandable for non-programmers. In addition to this you can use Visual Studio 2015 from Microsoft to write any custom code or modify the game engine itself. It is possible to make a game by just using the blueprint system, just using C++ code, or one can use a combination of the two.

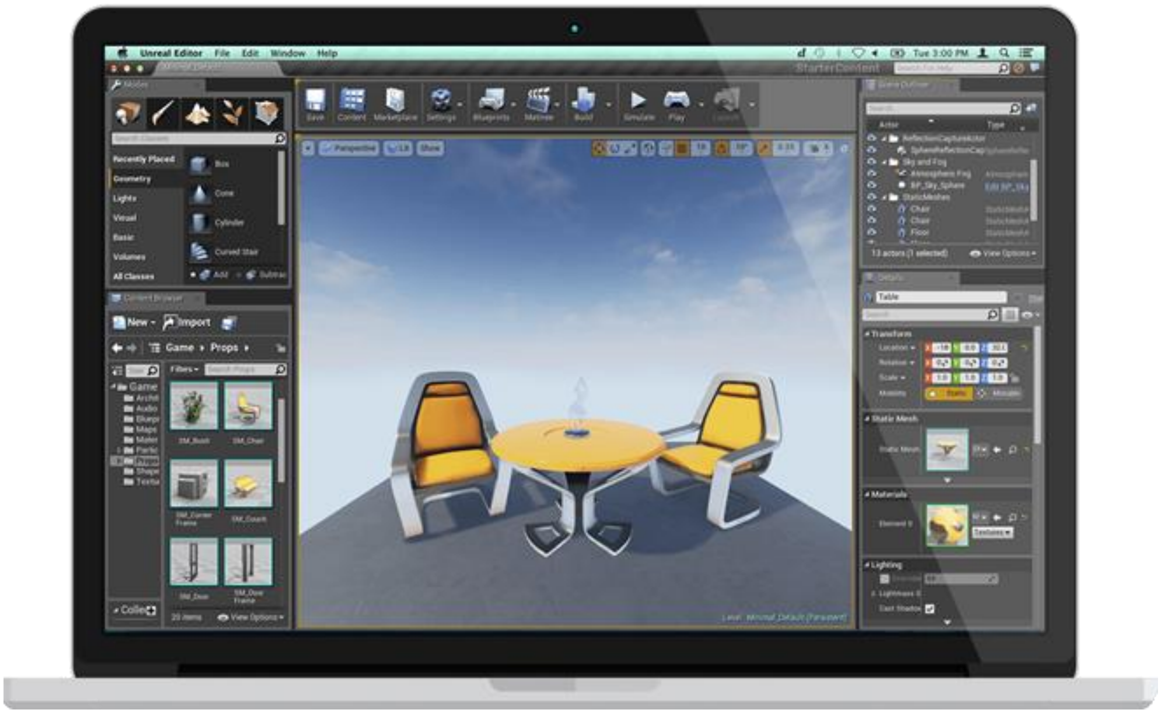


Figure 2-5 – The standard Unreal Engine 4 Editor window. On the left one can see tools for editing and adding objects to the 3D world. In the middle is the 3D environment, this can be interacted with, even used to play a game. One the right is the property windows for a selected object. This allows manipulation of an object in several ways.

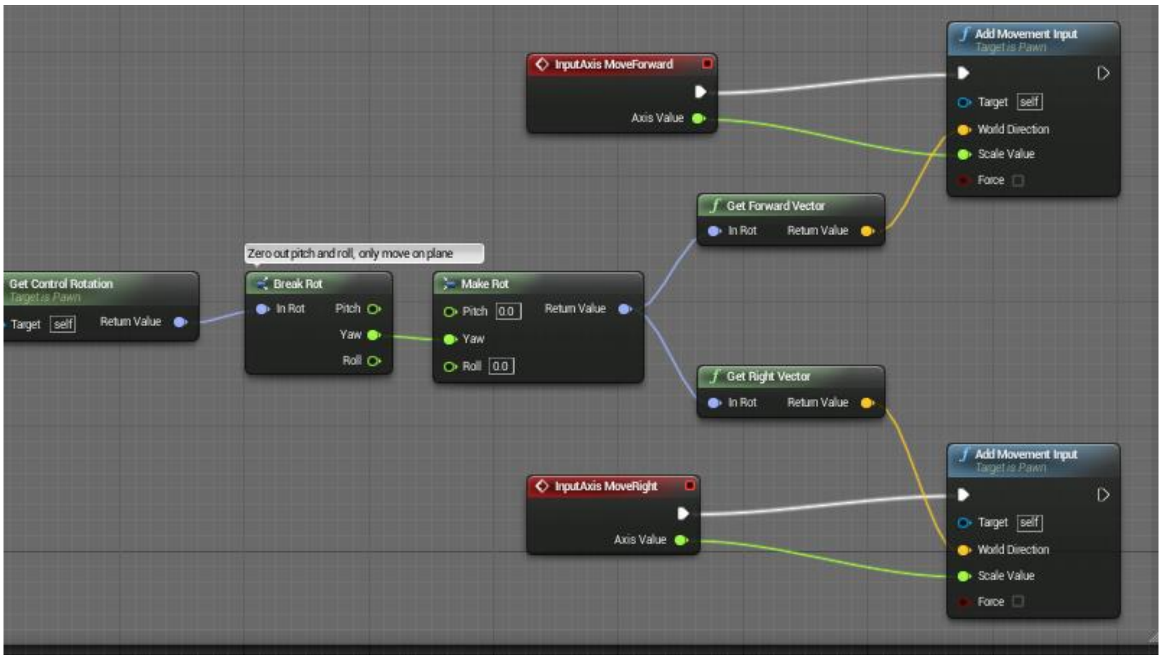


*Figure 2-6 – Unreal Engine's blueprint scripting system. This is the replacement for simple scripts in UE4. In the picture one can see a script that is run when either the **move forward** or **move right** action is executed (red nodes). The nodes connected by the different lines are functions or methods. The **white** lines are the execution path, while the **coloured** lines indicate input/output variables of different types. For example, green lines are decimal numbers.*

From this point it is assumed you are familiar with how Unreal Engine works[1]. This means that you understand what Unreal Engine 4 is, what the editor is used for, and how the blueprint scripts work. The rest of this section will describe how UE4's default AI system can be used. This means one needs to look at how BTs are implemented in UE4.

2.3.1 Behaviour Trees

The behaviour trees in UE4 consists of two elements, a **blackboard** and the **behaviour tree.** The blackboard works as the AIs memory and stores information for the BT to use. The BT is the AI's processor. It makes the decisions, and then makes an actor act upon those decisions. The previous section described event-driven BTs. This is what is used in UE4. This means one get the advantages this brings. Additionally, Epic Games have made some additional improvements to the BT implementation in UE4. This subsection will describe the differences from the standard model more closely.

Changes from the Standard Model

Figure 2-3, in section 2.2.1, showed an illustration of a behaviour tree. There it was explained that the figure was of a BT implemented in UE4, and that it varied a bit from the explanation given in that section. The next paragraphs will explain those differences and what they mean.

In the standard model for behaviour trees, **condition** are leaf nodes, which simply does nothing other than succeed or fail. This have been replaced by **conditional decorators**. Making conditions be decorators rather than tasks has a couple of significant advantages. First, conditional decorators make the behaviour tree UI more intuitive and easier to read. Since conditionals are at the root of the sub-tree they are controlling, you can immediately see

[1] A better introduction to UE4 can be found on Unreal Engine's webpages, www.unrealengine.com.

what part of the tree is “closed off”, if the conditionals are not met. Also, since all leaves are action nodes, it is easier to see what actual actions are being ordered by the tree. In a traditional model, conditionals would be among the leaves, so you would have to spend more time figuring out which leaves are conditionals and which leaves are actions. Another advantage of conditional decorators is that it is easy to make those decorators act as observers (waiting for events) at critical nodes in the tree. This feature is critical to gaining full advantage from the event-driven nature of the trees.

Standard behaviour trees often use a **parallel** node to handle concurrent behaviours. The parallel node begins execution on all of its children simultaneously. Special rules determine how to act if one or more of those child trees finishes (depending on the desired behaviour). Instead of normal parallel nodes, UE4’s BTs use **simple parallel** nodes and a special node type that is called **services** to accomplish the same sorts of behaviours.

There are **three** types of nodes which provide the functionality that would normally come from parallel nodes. They are **simple parallel nodes**, **services** and the **decorator “observer aborts” property**.

- **Simple parallel** nodes allow only two children: one which must be a single task node (with optional decorators), and the other which can be a complete subtree. The node is relatively simple in concept compared to traditional parallel nodes. Nonetheless, it supports much of the most common usage of parallel nodes. Simple parallel nodes allow easy usage of some of the event-driven optimizations in UE4. Full parallel nodes would be much more complex to optimize.
- **Services** are special nodes associated with any composite node (selector, sequence, or simple parallel). They can be registered for call-back every X seconds and perform updates of various sorts that need to occur periodically. For example, a service can be used to determine which enemy is the best choice for the AI pawn to pursue, while the pawn continues to act normally in its BT toward its current enemy.
- The **decorator “observer aborts” property** allows conditional decorator nodes to observe values to abort operations. This replaces the need to have parallel nodes that acts as conditional checkers.

Behaviour Tree Blueprints

The BTs in UE4 uses the blueprint scripting system. This means that one gets a visualization of the BT and can visually debug it in real-time. An example of a BT, as it is implemented in Unreal Engine 4, was illustrated in figure 2-3 in section 2.2.1. More examples will be given in chapter 4.

2.4 Cyclist Behaviours

This section will give an overview of the rules and laws that cyclists have to follow and how cyclists behave towards them. The section also introduces cyclist types and how one can group cyclists together in categories. The project will use the Norwegian traffic rules for cyclists.

If every cyclist followed the rules to the same extent as drivers of other vehicles do, then it would be a matter of implementing the rules for a cyclist agent. The situation is not that simple. Every cyclist follows the rules that suits them and use infrastructure that they are comfortable with. Some cyclists use all the rights given to them, while others are more conservative and careful. This variation is what is meant by different cyclist personalities and one needs to understand them better in order to model different behaviours.

2.4.1 The Norwegian Cyclist

Cycling is becoming an important transport option in Norway. In the biggest cities it serves as an important traffic congestion reducer. Cycling, combined with other public transport options, is supposed to stand for all the increased traffic into cities the coming years. This means that bicycles needs to be used both for work and leisure trips (The Norwegian National Transport Plan, 2016).

Today, in Norway, the most common reason to use a bicycle is the practical benefits and time saved compared to other transport alternatives. It is also for economical and health reasons for many. An average trip is four kilometres and the most common trip is related to school or work. The most active cyclist-group are aged from 13-17 years old, due to their lack of other private transportation alternatives. Cyclists are most active in June and the other summer months. December and January are the least active months, due to the Norwegian winter. Despite this, winter cycling has become more common, and it helps decrease traffic in big cities throughout the year. The risk when cycling have been drastically reduced the last years. Since the middle of the 90’s to 2010, the number of badly injured and dead have been reduced by 40%. The most common accidents happen in narrow streets. And the elderly is over-represented in the accidents. (Krekling et al., 2014) and (Espeland and Amundsen, 2012).

2.4.2 Norwegian Cyclist Laws

One needs to understand the laws and rules that applies to cyclists. This will be essential when modelling an autonomous cyclist. The AI agents should try to follow the rules to the extent of the simulated personality.

All cyclists are motorised drivers in Norway. This means that they need to follow the same traffic rules as normal road traffic. There is no requirement for a driving licence to use a bicycle, but it is important that the cyclist knows and follows the laws. The rules and laws presented in this subsection is taken from NPRA’s webpage about cyclist rules and laws (NPRA, 2016), Syklistene.no (2016) and ("Sykkelhåndboka", NPRA, 2014).

Road Types

A cyclist’s route can go along several different roads, but one can roughly categories these in four groups.

- Mixed roads. This is normal roads that contains cyclists and other vehicles.
- Roads with cycle lanes. This is an extra lane next to the mixed roads that is meant for cyclists.
- Cycle roads and pavements. This are roads meant for cyclists, or pedestrians and cyclists.
- Pedestrian roads. This is roads meant just for pedestrians (and cyclists under certain conditions).

Most of the rules for cyclists applies for all the different road types, but some rules are different for each type. This can bring some confusion for some cyclists, as there is no formal process or test needed for riding a bicycle in Norway. That is, as long as one follows the laws.

Yielding Rules

The most confusing rules are the yielding rules. The yielding rules means that you should yield for traffic on roads that crosses the lane you are in. This is also true for cyclists. The yielding rules are different depending on what road type you are on and what road you are crossing or entering. Here are some examples:

- When cycling on a cycle road or pavement, and crossing or entering another road, one must yield for the traffic on that road.
- Vehicles that enters or exits roads that are not available for general traffic, must yield for cyclists on cycle roads, pavements or road shoulders. This could be roads coming from gas stations or parking lots.
- Cyclists needs to yield when entering a mixed road from a pavement, cycle road or road shoulder. If the vehicle is entering or exiting a side road, then the vehicle must yield for cyclists on pavements.

This does not cover all situations, but it should give an idea where the confusion comes from.

Roads with Cycle Lanes

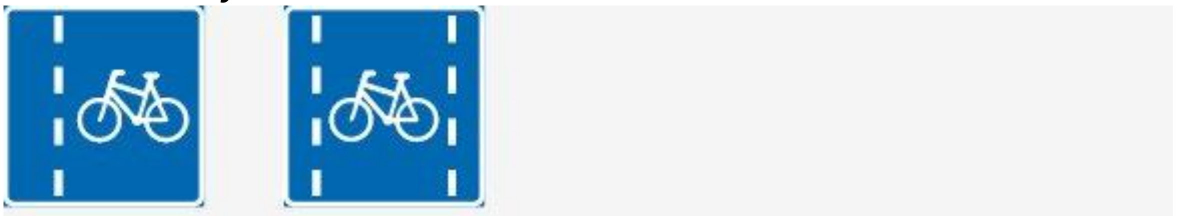


Figure 2-7 – Sign indicating that there is a cycle lane on one or both sides of the mixed road.

Cycle lanes are only for cyclist traffic. You can only use the cycle lane on the right side of the road. If there is only a cycle lane on one side of the road, then you can use that in both directions if there is no sign that indicates a traffic direction. You cannot stop or park in a cycle lane.

Cyclists on mixed roads and cycle lanes got the same yielding rules as normal vehicles. That is, they yield for traffic from the right.

Roads with Public Transport Lanes

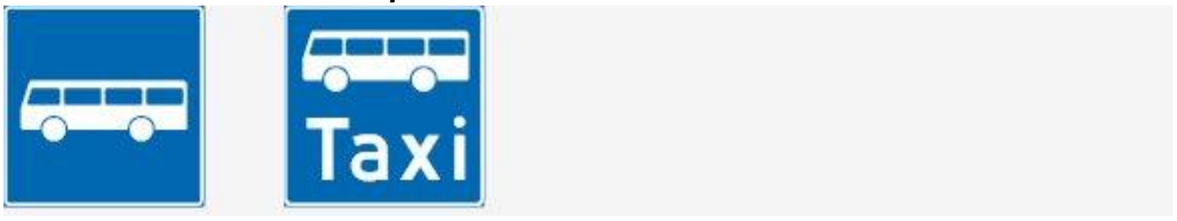


Figure 2-8 – Sign indicating public transport lanes.

Public transport lanes can be used by cyclists.

Road Shoulders

Road shoulders can be used by cyclist. Cars should pass cyclists with a safe distance between them. Around 1,5 meters is acceptable.

Cycle Roads

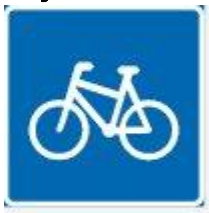


Figure 2-9 – Sign indicating a cycle road.

Cyclists can use a cycle road. Other vehicles can only use it with special permissions. One can cycle in both directions on a cycle road if there are no markings.

Pavement

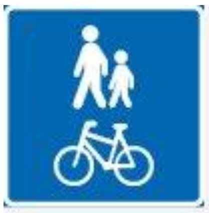


Figure 2-10 – Sign indicating a pavement

Cyclists and pedestrians share pavements. Cyclists should use the right side of the pavement. Pedestrians can walk on both sides.

Pedestrian Roads

Pedestrian roads are meant for pedestrians, but cyclist can use them under certain conditions. They can be used if the traffic load on these roads are light. Cycling must also not increase the risk for accidents or be of any hinder for the pedestrians.

One-way Roads

One-way roads apply for all motorised vehicles. The exception is if the cyclist uses a pavement that goes in the opposite direction or if a sign indicate cyclists can cycle both ways.

Roundabouts

Cyclists can use roundabouts. Here the normal rules for roundabouts apply. One must yield for vehicles in outer lanes. When entering the roundabout, one must yield for traffic in the roundabout.

Traffic lights

Traffic light got priority over traffic signs and the right-hand rule. As a cyclist, one must follow the light on both the mixed roads, cycle roads and pavements.

2.4.3 Legal Bicycle

For a bicycle to be legal, it must follow these rules:

- The bicycle need two brakes. One on the back and one on the front wheel.
- The bicycle must have a red safety reflector behind.
- The bicycle must have a white or yellow reflector on the side of the pedals (or pedal arms).
- The bicycle must have a bell.

- The bicycle must have a light in the front and back. A white or yellow light in the front and a red light in the back. This is only needed in dark or low-sight conditions.

Cyclists ignore most of these rules, and shops are allowed to sell bicycles without this equipment. If the police stop cyclists lacking said equipment, they can be fined for breaking the law.

2.4.4 Cyclist Accidents

This subsection looks at different cyclist accidents. This should give a better understanding of different dangerous situations related to cycling. This can help with simulating different behaviours or making test scenarios.

The accidents that are listed in this section is found in an analysis done by the Norwegian Public Roads Administration (NPRA) (Krekling et al., 2014). A summary of the most interesting founds will be presented. Their analysis builds around the reported cyclist accidents in Norway between 2005 and 2012.

All reported accident

The analysis done by the NPRA shows men are twice as likely as women to be in a traffic accident. Even though the number of cyclist for both sexes are almost equal. They reason that this is due to men's willingness to take more risks.

The most exposed group are those between 10 and 14 years old. This is likely due to their lack of understanding and knowledge of traffic situations. The practice in Norway is to allow children to cycle to and from school, from the age of 10. This why those younger than 10 is less exposed to accidents.

Most accidents happen in "T" or "X" intersections (62%) and on normal road sections (30%). Most accidents with injuries or deaths happen in these areas. Almost 50% badly injured or killed passengers, and 62% of lightly injured passengers, comes from these accidents. After this, the accidents vary over different areas. Most accidents happen where the speed limit is 50 km/h. 71% of accidents involving badly injured or killed passengers, and 84% of accidents involving lightly injured passengers, happen here.

2.4.5 Cyclist Behaviours

This subsection will present one way to categorize cyclists. The work in this subsection is based on the work done by Herfindal (2015), information from Sykkelhåndboka by the NPRA (2014) and two interviews[1].

Cyclists could be said to have different knowledge, experience and skill-level. They use this to choose their desired route to their goal. Each cyclist has some different desires when they choose a route. These desires can be described as: navigability, safety, security and environment.

- **Navigability** – How good is the route based on factors like traffic, road conditions, length and elevation. This makes the route more efficient.
- **Safety** – How safe does the cyclist feel when they are following this path. This is very subjective as each person feels safe in different situations. The safer a cyclist feels with a route, the more likely they are to use it.

[1] The interviews were with two transportation researchers that work with cyclists in their research. One is a transportation planner at the NPRA and one is a transportation researcher at SINTEF.

- **Security** – This is related to how secure the route is. For example, is there a cycle lane, a bridge over a heavily used road, traffic lights or good road crossings along the route. The infrastructure must be secure, but must also make the cyclist feel secure, due to the above mentioned point about safety.
- **Environment** – This says something about the environment the cyclist's route goes though. If a route takes one minute longer, but goes next to some nice scenery, then some cyclists would choose that route instead. The same would append if the cyclist tries to avoid certain environments.

There is no standard way to categorize cyclist behaviours, but there are some typical cyclist groups. Each group have some unique features, and can be used to make different personalities.

- **Transport Cyclist** – They normally uses a bicycle for transportation to and from school or work. They normally have special clothing, sufficient safety equipment and decent knowledge of the rules for cyclists. They feel safe in most situations. Efficiency is important when choosing a route, instead of safety and environmental reasons.
- **Leisure Cyclist** – They might use the bicycle occasionally, often without sufficient safety equipment. They lack some knowledge about cyclist rules. There is much variation in their safety preference. They could be said to prefer each of the mentioned desires equally.
- **Child Cyclist** – Children lack much knowledge about the traffic rules. They might not want to use safety equipment and cannot read the traffic situation properly, due to lack of experience. They prefer routes that are safe and secure, and care less about efficiency and environmental desires.
- **Tourist Cyclists** – They often use the country roads and roads without pavements or cycle lanes. They might not be familiar with the Norwegian cyclist rules. They can be classified as "leisure cyclist" when in towns. Out of town they will use the roads that are available, but will prefer routes with a nice scenery.
- **Professional/Exercise Cyclists** – These are often using the mixed roads for competitions or exercise. They usually have high speeds and use the mixed roads freely, according to their needs. In cities, they can be classified as transport cyclists. Outside of cities, they prefer navigability and scenery, as they feel safe in most situations.

3 Research Method

This chapter will present the research method that is used in this project. The research method should test if, and how well, the project’s goal is fulfilled. The project has two different sides that should the evaluated: the technical implementation and the AI agent’s ability to simulate believable cyclist behaviours. A case study will be used to gather data for these evaluations.

3.1 Case Study

In a case study, one usually do a qualitative study of individuals, which in this project is cyclists. The different cyclists will be simulated by the AI agent. Each cyclist will do a set of tests. There will be taken recordings of the text for later analysis. Each cyclist will do the tests three times to make sure that the AI agent produces consistent result. Normally this would not be possible in a case study, due to the subjects learning from the first round of tests. In this case one can erase their memory, thus enabling the tests to be run several times.

3.1.1 Test Cases

Tests will be used to evaluate the different cyclists. The tests will have increasing difficulty and complexity. Both the cyclists’ personality and quality of the implementation should be tests. The tests should therefore be designed with this in mind. Each test should be done in a street or city environment. This could for example be an intersection with traffic-lights, pavements and cycle lanes.

There will be six tests in total. First there will be three static tests: a basic test, then two more advanced tests. A static test means that there will be no moving actors. These tests should give a reasonable baseline to build the evaluations on. Next there will be three dynamic tests. These tests will have moving actors that interact with the 3D environment and the cyclist. Again, the three tests will be divided into one basic and two more advanced tests. The results from these tests will better determine how well each personality behaves. It will also help with the objective evaluation of the implementation. Table 3-1 to 3-6 present each test.

Test 1	**Basic Static Test – Lane Choice**
Description	The cyclist will have three lane choices. Mixed road, cyclist lane and pavement. There will be nothing that blocks the lanes. There will be no traffic. The goal is just the reach the destination.
Behavioural Test Reason	The test will show that the cyclist makes simple choices.
Systems Test Reason	Basic testing of the BT and its underlying systems. Basic EQS test and path-finding test.
Test Steps	1. Present the cyclist with some lane choices. 2. Observe the choice the cyclist makes.
Expected Results	The cyclist will take the lane based on their personality.

Table 3-1 – Description of test 1.

Test 2	**Advanced Static Test 1 – Intersection**
Description	The cyclist will approach an intersection where two roads meet. The cyclist will either use the mixed road, the cyclist lane or the pavement. There will be no traffic and the traffic lights will be turned off. The goal will be to interact with the intersection the reach the goal.
Behavioural Test Reason	The test shows if the cyclist can make reasonable complex choices. The cyclist should show reasonable behavioural traits.
Systems Test Reason	More complex test of the BT. More variables to consider. Should test most sub-systems that handles static environments.
Test Steps	1. Let the cyclist choose the lane they are comfortable with. 2. Let the cyclist approach the intersection. 3. The cyclist will try to reach a goal in either direction in the intersection. 4. Observe the choices the cyclist makes to reach to goal.
Expected Results	All cyclists should reach the goal. The path will vary based on their behavioural traits.

Table 3-2 - Description of test 2.

Test 3	**Advanced Static Test 2 – Crossing the road**
Description	The cyclist will be cycling on the right side of the road. Either on the pavement, cyclist lane or on the mixed road. Their destination is on the left side of the road. There is no traffic. There is a road crossing up ahead of the cyclist. The road crossing if further ahead than the destination. The cyclist must reach the destination.
Behavioural Test Reason	This will test if the cyclist will use the road crossing or take the risk of crossing the lane in the opposite direction.
Systems Test Reason	It will test the path-findings cost estimation. It also tests the BT further by making it take some choices.
Test Steps	1. Let the cyclist choose a lane. 2. Let the cyclist do its thing and find a way to the goal. 3. Observe the choices the cyclist makes to reach the goal.
Expected Results	The cyclist will reach to goal. Paths may vary based on personality

Table 3-3 - Description of test 3.

Test 4	**Basic Dynamic Test – Trafficked Lane Choice**
Description	The cyclist will choose from three different lanes. Either mixed road, cyclist lane or pavement. There will be traffic in the lanes. This test is the same as the basic static test, just with added traffic.
Behavioural Test Reason	Test to see how the cyclist behaves with moving traffic. Will the cyclist behave any different from the static test.
Systems Test Reason	The test will use all the previous systems in addition to the EQS to detect traffic. The pathfinding should account for traffic to make different choices.
Test Steps	1. Let the cyclist choose the lane they are comfortable with. 2. Let the cyclist move along the lane. 3. Observe how the cyclist interacts with other actors.
Expected Results	The cyclist will react to the traffic and find a safe way to the goal. This means interacting with the traffic in a reasonable way. The path chosen should vary depending on the personality. The cyclist should avoid crashes. Most dangerous situations should be avoided.

Table 3-4 - Description of test 4.

Test 5	**Advanced Dynamic Test 1– Intersection with Traffic**
Description	There will be a X-intersection with several lanes and traffic. The cyclist will be using the lane of their choice when approaching the intersection. The cyclist lane merges with the mixed road in the intersection. The cyclist will be going to the right in the intersection. The intersection will be light regulated. There will the traffic in the lanes.
Behavioural Test Reason	The test will let the cyclist choose several ways to deal with the traffic. How does the cyclist use the options available to get to its goal.
Systems Test Reason	The test will test several parts of the AI system at once. Both path-finding and the response to other actors should play an important role. This tests all the underlying systems to make sure they function as intended.
Test Steps	1. Let the AI choose a lane on the right-hand side. 2. Let the cyclist approach the intersection. 3. Observe the choice the cyclist makes to reach their destination.
Expected Results	The cyclist will either use the road crossing or the mixed road.

Table 3-5 - Description of test 5.

Test 6	**Advanced Dynamic Test 2 – Crossing a trafficked road**
Description	This will be the same test as Advanced Static Test 2. The difference will be that there is traffic on the mixed road. The goal is to reach the goal on the other side of the road. The cyclist can use all means to reach to goal.
Behavioural Test Reason	This will test the personality of the cyclist more accurately. How willing is the cyclist to take risks in order to reach to goal faster.
Systems Test Reason	In addition to the EQS and other underlying systems, this test can have very varying results based on a cyclist's behavioural traits. This means that this intersection between all the systems are tested.
Test Steps	1. Let the cyclist choose a lane on the right hand side. 2. Let the cyclist make choices to reach the goal 3. The cyclist will reach the goal on the left hand side of road, on the pedestrian path.
Expected Results	All cyclists should reach the goal on the left hand side of the road. The cyclist will choose a path based on the current traffic and their personality.

Table 3-6 - Description of test 6.

3.1.2 Test Subjects

Chapter 2 introduced a way to categorize cyclists into types. Each type had some specific characteristics. These types will be used to categorize the cyclists in the case study. This means that the AI agent will try to simulate these different cyclist types. A good choice for the tests is to have cyclists with different desires. The case study will therefore look at the **transport**, **leisure** and **child** cyclist. Together they cover most of the different desires described in chapter 2. Table 3-7 to 3-9 will give an analysis of the different cyclists' expected behaviours.

Analysis of the Transport Cyclist's Expected Behaviours	
Test 1	Expect the cyclist to use the mixed road or the cycle lane. Expect high speeds throughout the scenario, as there is no traffic.
Test 2	Expects the cyclist to use the cycle lane or mixed road. Might follow the light regulation, but might break the rules if there is no traffic.
Test 3	Expects the cyclist to cross the road diagonally to reach to goal as fast as possible.
Test 4	Expects the cyclist to use the cycle lane or the mixed road, depending on the traffic. Expects high speeds.
Test 5	Expects the cyclist to use the cycle lane or mixed road. Expect the cyclist to either use the road crossing or intersection. Efficient choices.
Test 6	Expects the cyclist to use the mixed road, then cross over to the goal if there is no traffic. Might also use the road crossing as this is faster if there is much traffic.

Table 3-7 – Test analysis for a transport cyclist. Expected result for each of the test scenarios.

Analysis of the Child Cyclist's Expected Behaviours	
Test 1	Expects a medium to high speed. There is no traffic. Expects that they choose the pavement throughout the scenario.
Test 2	Expects medium to high speed up to the intersection. Expect the cyclist to use the road crossings all the way. Expect the cyclist to wait at the light at the road crossing.
Test 3	Expect medium to high speed up to the road crossing. Expect the use of the pavement and road crossing. Expect them to follow the traffic lights
Test 4	Expect slow speeds past the pedestrians. Expects the cyclist to use the pavement or the cycle road.
Test 5	Expects the cyclist to use the pavement or bicycle road all the way. Expects slow speeds over the road crossing. Expects them to follow the traffic lights.
Test 6	Expect them to use the cycle road or the pavement up to the road crossing. Expect them to follow the light regulation.

Table 3-8 - Test analysis of a child cyclist. Expected result for each of the test scenarios.

Analysis of the Leisure Cyclist’s Expected Behaviour	
Test 1	Expects cyclist to use the cycle road or pavement. Expects high speeds.
Test 2	Expects the cyclist to use the cyclist lane or the pavement. Expects the cyclist to follows light regulation if on cyclist path. Expects cyclist to NOT follow light regulation if using the pavement.
Test 3	Expects the cyclist to cross the road as soon as possible or at the road crossing at medium speed. Expect they will ignore the traffic light, if using the road crossing.
Test 4	Expects the cyclist to use the cycle road. Expects the cyclist to have medium to high speeds.
Test 5	Expects the cyclist to follow the cyclist path to the intersection. Might use the mixed road or the pavement to get past the intersection. Will follow the lights.
Test 6	Expect the cyclist to use the cycle road to the road crossing. Expect them to follow the lights if there is traffic. Might cross the road before the road crossing if all traffic drives past them.

Table 3-9 - Test analysis for a leisure cyclist. Expected result for each of the test scenarios.

3.1.3 Evaluation Method

The tests will be recorded in by a screen-recording software[1]. A video recording of the tests will form the basis for the evaluations. The next step is to evaluate the results. As mentioned, there are two kind of evaluations: the implementation and the AI agent’s ability to simulate believable behaviours.

The implementation can be evaluated by a technical analysis of the finished AI agent. This should determine how good the solution is implemented. The analysis will look at the AI agent’s abilities compared to the project’s goal. This can be done by analysing test results produced by the case study. The tests should tell what the AI agent can do and how well it does it. One can then draw conclusions by comparing the project’s goals, the implementation and the test results.

The AI agent can be evaluated by having it simulate cyclists in different traffic scenarios. The results can then be compared with the expected behaviour of real cyclists. This will be a subjective evaluation of how believable the AI agent makes the cyclists behave. The quality of the AI agent will be determined by the choices it makes for the different cyclists. The video recordings of the tests will also be presented to two transportation researchers that work with cyclists in their research. This will eliminate some of the subjective interpretations of the results. This is the same researchers that was interviewed about cyclist behaviours for section 2.4.5. They will evaluate the videos by answering questions about the tests and the cyclist behaviours. The questions can be found in Appendix A.

[1] Screen recordings will be done with Nvidia Shadowplay. See http://www.geforce.com/geforce-experience/shadowplay for more details.

3.1.4 Limitations

The goal of the project is to show that one can simulate different cyclist behaviours by using behaviour trees. This means that the tests are meant to show that this is possible. The tests will therefore exclude many scenarios which would introduce more variables or more complex environments. This means that several tasks, that one would expect a cyclist to perform, will be untested. For example, the tests contain no elevation and there are no road crossings without traffic lights. This is something that one would expect cyclists to use.

The tests also limit how advanced one can say that the AI behaves. One cannot know if the AI can handle more complex tests than the most complex test scenario. Test five and six have been set as this projects goal when it comes to complexity. If the AI can manage these tests, then one would need new tests to know what the absolute limitations of the AI is. One could make a very hard test, but this would result in too much time spent on making the tests. The proposed tests are meant as a balance between how much time one will spend on preparing the tests and how much time one will spend when making the AI.

Case studies usually relies on subjective interpretations of the results. This is also true in this project. It is not good to conclude upon a result based on a very subjective discussion. This is why the results will be verified by two researchers that work with cyclists and transportation on a daily basis. If their opinion about the test result is the same as mine, then the result is at least verified by three independent parties. This could limit the validity of the projects results.

4 Design and Implementation

This chapter will explain how the autonomous cyclist agents and the test scenarios are designed and implemented. Section 4.1 will look closer at how the test scenarios are designed. Section 4.2 will explain how the scenarios was implemented into the simulator. Section 4.3 will explain how the cyclist model from the existing simulator is controlled by the AI agent. Finally, section 4.4 will explain how the AI agent gives personalities and different behaviours to the cyclists.

4.1 Designing Test Scenarios

This section looks at the design of the test scenarios. It explains the reason why one needs the different tests, what will happen in each test and what models must be made in order to run the tests.

First of all, the tests are what will be used to evaluate most of the work in this paper. Therefore, the tests should test both the AI agent's abilities to control the cyclist model and its abilities to simulate different personalities. The tests must be able to test the basic behaviours first, and then they can gradually increase in difficulty. This ensures that the project can be evaluated, even if the project's goals are more difficult to achieve than anticipated.

4.1.1 Designing the Test Scenarios

This subsection explains why each of the different scenarios was chosen as a test. Each of the scenarios are either a static or dynamic scenario. The static scenarios will test if the AI makes good choices without any distractions. They give a baseline for the AI agent's ability to simulate a cyclist. On the other hand, in the real world there will be a dynamic environment. This means that the cyclist should be able to make reasonable choices, even in scenarios with moving traffic. This is what the dynamic scenarios will test. Three unique scenarios were presented in the previous chapter. Each scenario will be run twice, once with traffic and once without[1]. This gives a total of six tests.

Test 1 – Choosing a Lane

Test 1 is meant as a basic test to show that the AI agent works. The cyclist will have a choice of three lanes over a straight distance with no obstacles. The goal is just to go in a straight line from the start to the goal. This will be the absolute baseline for the AI agent's abilities.

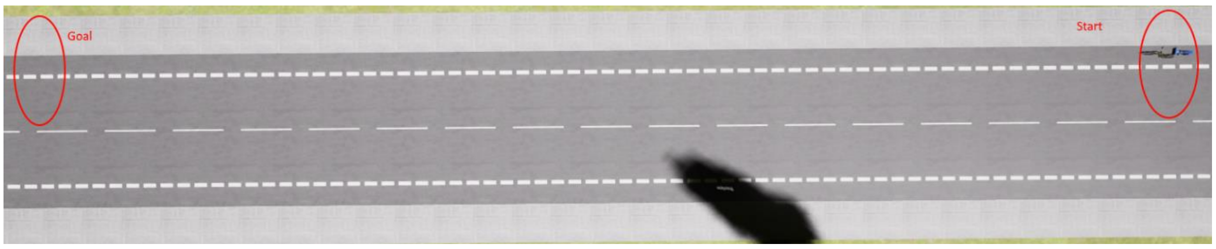


Figure 4-1 – Test 1 with the start and goal marked with red circles.

Test 2 – Crossing an Intersection

Test 2 will further test the AI agent's ability to understand the surrounding environment. The cyclist must find a way from one side of an intersection to the other side. This is again a static

[1] Additionally, each test scenario will be run three times to make sure one get a consistent result. This was explained in chapter 3.

test, but it offers some more complexity. The cyclist must navigate through an intersection that is regulated with traffic lights. This gives several new choices. First the cyclist must choose to use the road crossings or the mixed road, then the cyclist must either follow the traffic lights or ignore them. The last option is the speed the cyclist choose. This introduces several new problems for the AI agent to solve. The AI must interpret the traffic lights, as well as interact with the different parts of the intersection.

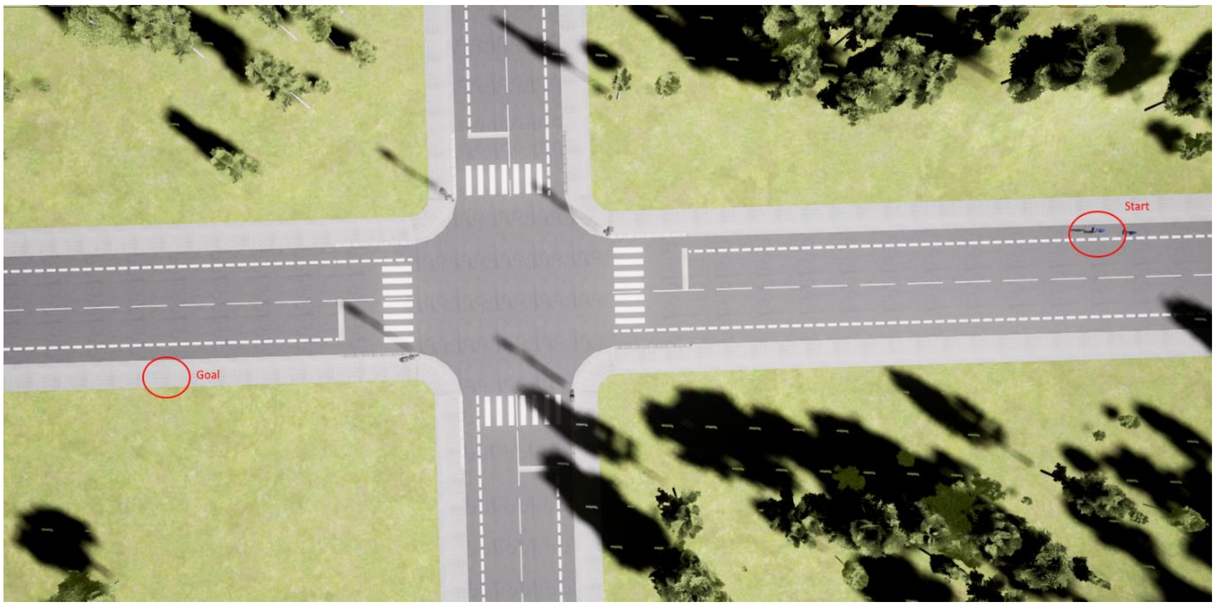


Figure 4-2 – Test 2 with the start and goal marked with red circles.

Test 3 – Crossing the Road

Test 3 tests a combination of the cyclist's personality and the AI agent's ability to understand the scenario. In this test the cyclist starts on the right side of the road. The goal is to cross the road to the other side. This can be done by using a road crossing or by cycling straight over road. Here the cyclist must decide if they want to follow the traffic rules, and to what extent they want to do so.

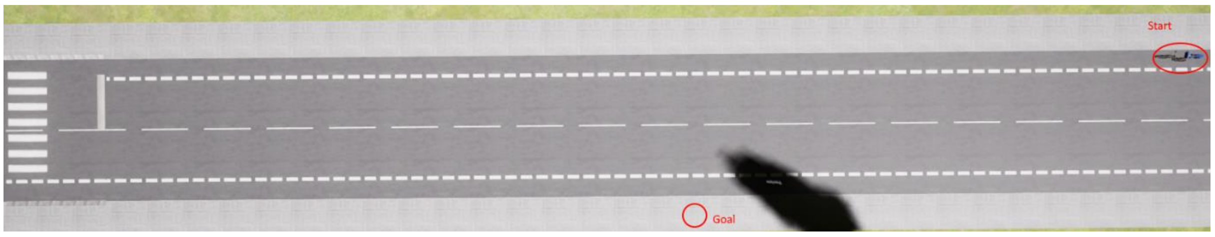


Figure 4-3 - Test 3 with the start and goal marked with red circles.

Test 4 – Lane Choice with Traffic

Test 4 is the first dynamic test. The scenario is the same as test 1, but with traffic. This means that there will be some pedestrians and cars in the different lanes. The goal is still to choose a lane and reach the goal at the end. The reason for this test is the check the AI agent's ability to understand traffic, as well as to test the effect of traffic on each cyclist personality.

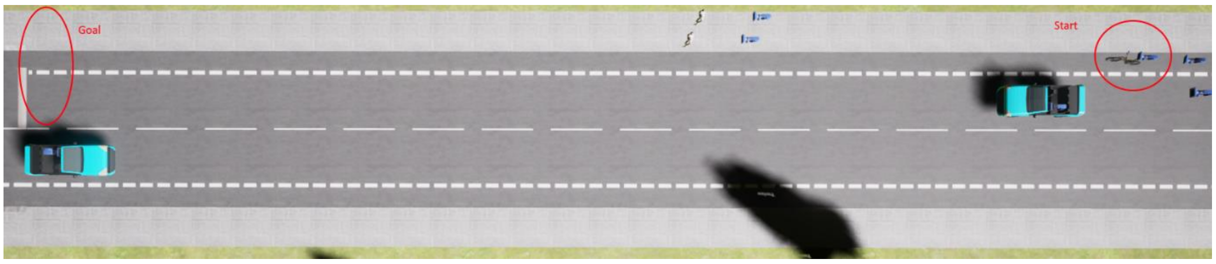


Figure 4-4 - Test 4 with the start and goal marked with red circles. This is the first test with some traffic.

Test 5 – Crossing an Intersection with Traffic

This test has the same scenario as test 2, but with traffic. The intersection will now have cars and pedestrians that the cyclist must interact with to get to the goal. This test further tests the AI agent’s ability to understand and interact with traffic.

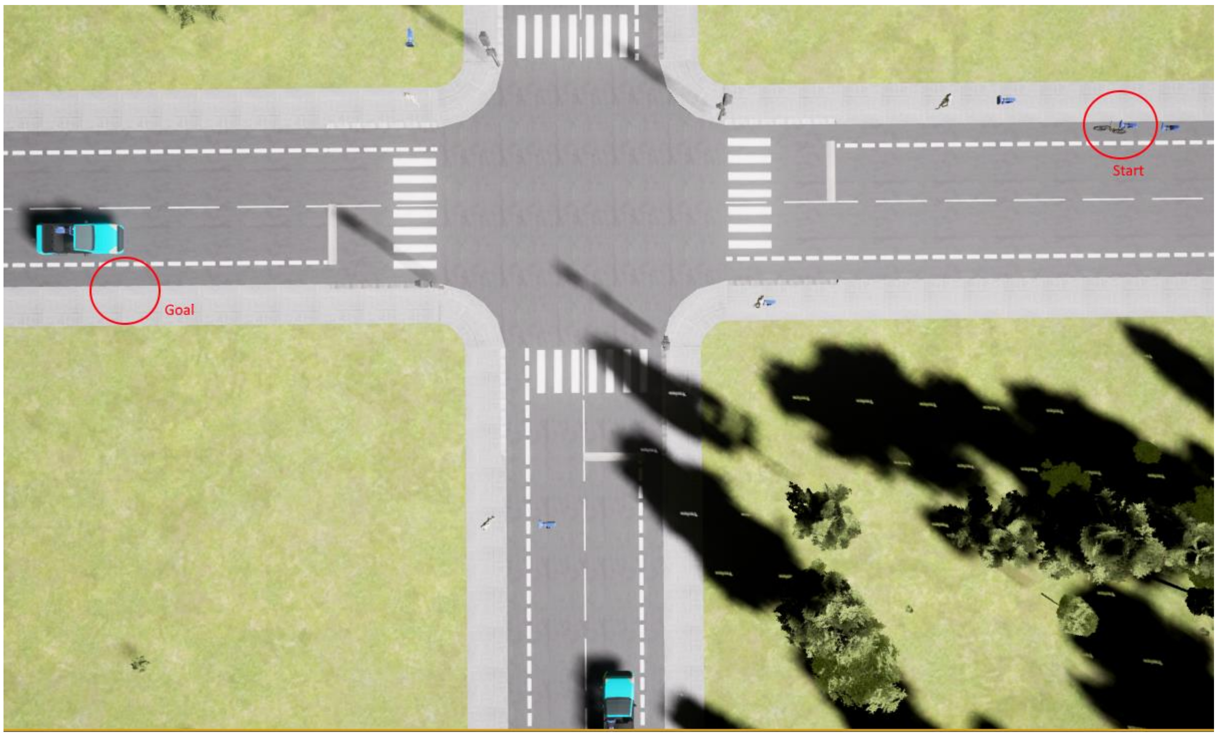


Figure 4-5 - Test 5 with the start and goal marked with red circles. There will be several pedestrians and cars moving around that the cyclist must interact with.

Test 6 – Crossing a Road with Traffic

Test 6 will confirm the results from test 5 with a different scenario. The scenario is the same as test 3. The cyclist's goal is to get to the other side of the road, either by using a road crossing or by just cycling straight over the road. This time there will be dynamic traffic in the lanes. This will further test the AI agent’s reasoning and the cyclist’s behaviours when they are in scenarios with traffic.

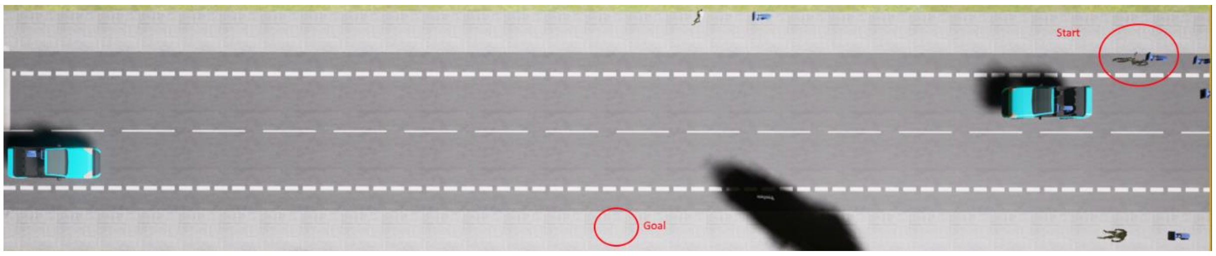


Figure 4-6 - Test 6 with the start and goal marked with red circles. The traffic might be placed differently in the tests. This is to better show the capabilities of the AI.

4.2 Implementing the Test Scenarios

This section will give a brief overview of how the tests were implemented. One need to design the 3D models, import them to Unreal Engine 4 and script the tests.

4.2.1 Making the 3D Models

The tests require several different models. Together they make up what have been referred to as the 3D environment. The most important parts are the terrain/road, the cyclist model and other traffic models, and the miscellaneous objects like road signs, traffic lights and road markings. One can additionally add trees, buildings and surrounding terrain. This could be useful, depending on the use-case of the finished system.

The Cyclist Model

The cyclist model was already included in the simulator. This was one of the reasons I wanted to use the existing simulator. The 3D model needed some adjustments before it could be used in this project. This involves making a copy of the existing model, adjusting some values that controls how the bicycle is balanced, removing some unneeded code related to manual transmission and the hardware controller, updating the animations, and making the model ready to be controlled by the AI agent.

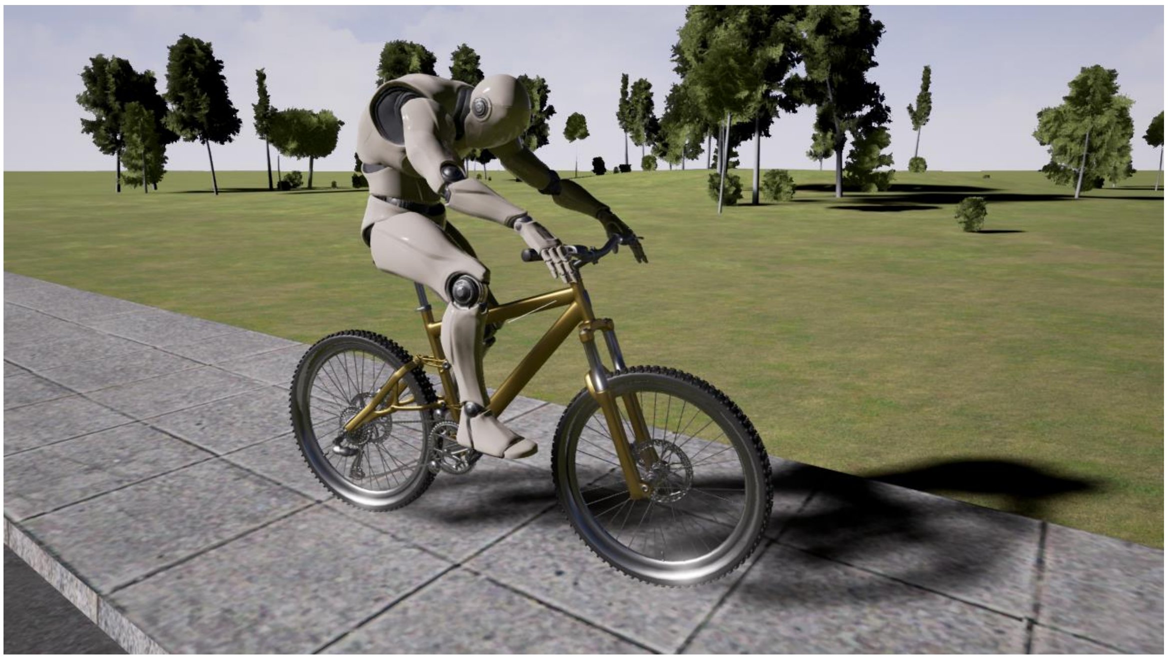


Figure 4-7 – The cyclist model that will be controlled by the AI agent.

The Car and Pedestrian Models

Unreal Engine 4 includes a premade car model and a human model. These are used in the project when pedestrians or cars are needed. These models are already animated and ready to use. Some code was added, so that an AI agent can control them. They will use a modified version of the AI agent. This means that the cars and pedestrians will work in all the scenarios that the cyclist works in. This will help to show that the project's solution can work with other models. As a bonus one gets cars and pedestrians that could, in theory, have different personalities.

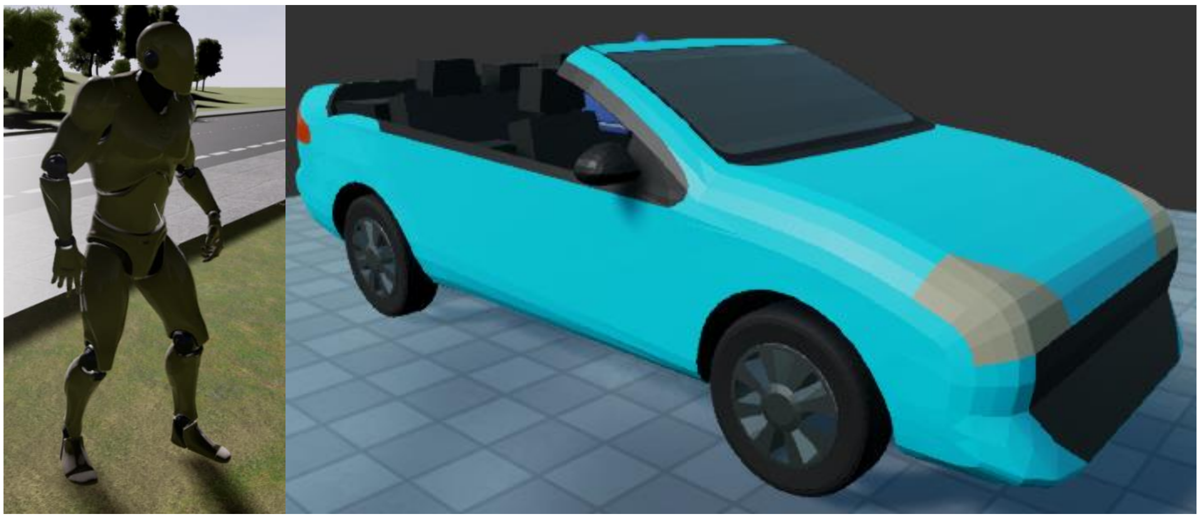


Figure 4-8 – Pedestrian and car models used in the scenarios. These are 3D models made by Epic Games as part of Unreal Engine 4.

The Road Model

The existing simulator has a cyclist model, but was missing a proper 3D environment. This was solved by using a 3D modelling software that can make road models. The software is based on the RoadXML standard.[1] To save time, it was decided to only make one road model. Most of the test scenarios could be set up by using the whole, or parts of, this model. The scenarios were adjusted a little to work with the planned model. First I had to make drawings and plans of the models. This could then be made with the modelling software. The drawings included planning what lanes was needed, how the roads should be connected, where road crossings should be, if there should be traffic lights, and the width and shape of the different roads/lanes.

This was then made in the modelling software by Jo Skjermo at SINTEF (he was also my supervisor for this project). Figure 4-9 shows the models that was received from the software. The models had to be imported to Unreal Engine 4. Then the models needed to be adjusted and finished before they could be used in the test scenarios. This meant that the traffic lights had to be modelled to function, both as standalone lights and together with other lights. The road had to be finished by adding some terrain, adding the road crossings and adding the traffic lights. Finally, the whole model had to be scripted so that it could be understood by the AI agent. This is described in the next subsection.

[1] RoadXML is an open-source project that gives a structured way to describe 3D models of roads for traffic simulators. 3D-modelling software based on RoadXML can produce an XML-file containing this information. For more information, see http://www.road-xml.org/.

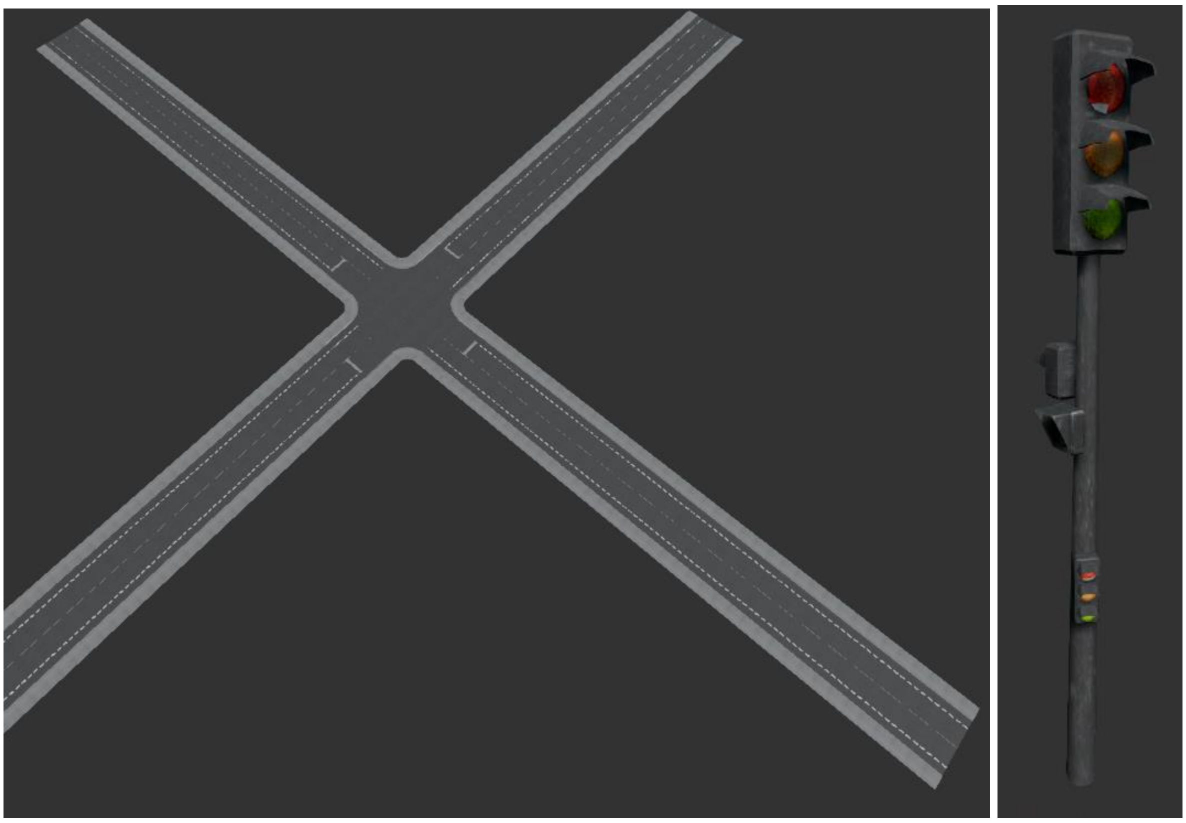


Figure 4-9 – These are the resulting models from the 3D modelling software. The road comes with three lanes. Mixed roads, cycle lanes and pavements. The traffic light is just a static model; It is not a functioning light.

4.2.2 Scripting the Scenarios

Scripting the scenarios involved connecting all the models in the 3D environment together into a working system. The goal was to make the AI agent and cyclist model as separated from the environment as possible. This makes it easier to make new environments later.

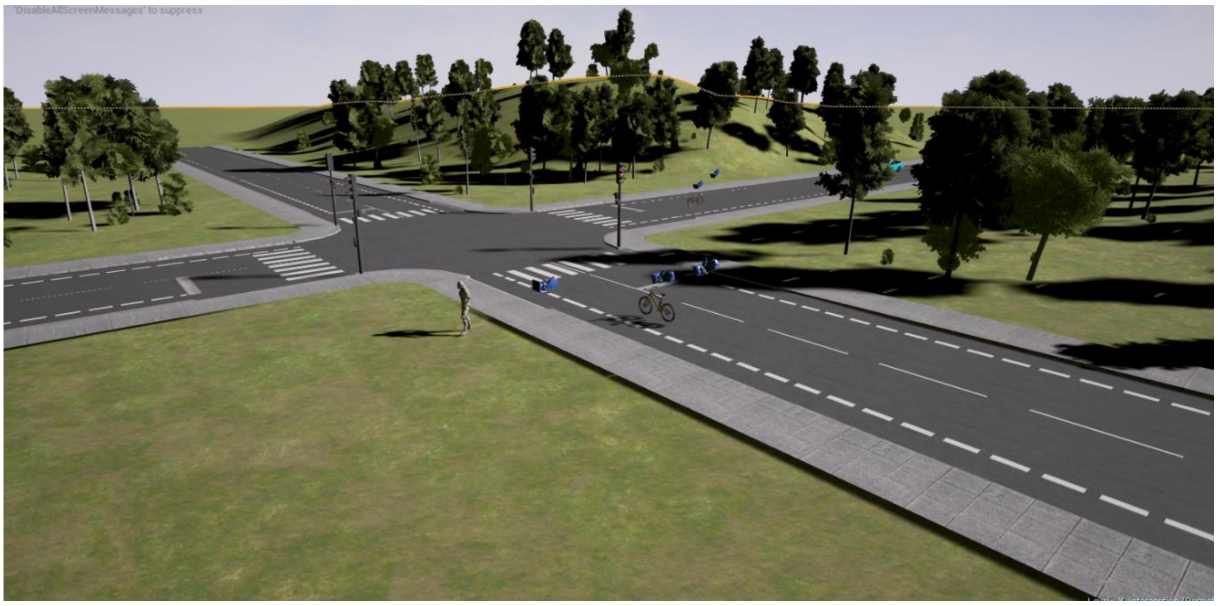


Figure 4-10 – The finished 3D environment for the test scenarios. This includes working traffic lights, surrounding terrain with vegetation[1], and some other features like lighting and a skybox that is part of Unreal Engine 4 already.

Traffic Lights

The traffic lights will be used by the pedestrians, the cars and the cyclists. This meant it required the following functionalities: it had to display light signals, it had to communicate with other traffic lights in the same intersection, and the lights had to be able to tell when a car, cyclist or pedestrian wanted to use the traffic light.

A fully optimized and functional light-regulated intersection is a complex system. The following assumptions was made to make it easier to implement a working intersection:

- None of the traffic-lights in the intersection can be green at the same time. This means that the main lights (those that control the road) will take turns to stay green.
- The road crossing will turn green if the pedestrians or cyclists signal the light. The road-crossing light turns green after the next time the main light is green for that traffic light.

This made the intersection easier to implement. Each light goes through a light-cycle, then it sends a signal to the next light in the intersection that it can turn green. The next light will then have its light-cycle, and so on.

Trigger Areas

Trigger areas are areas in the 3D environment this is marked with 3D rectangles. They are invisible to the naked eye, but the AI agent can use them to identify where it is. An area was made by adding a rectangular box over a part of the 3D environment, for example at each side of a road crossing. When the AI agents moves inside of this trigger area (trigger box), they will know that there is a road crossing nearby.

The AI agent needs three types of trigger areas. See figure 4-11 for a visual example.

[1] Thanks to unreal engine forum member fighter5347 for his free foliage kit. See https://forums.unrealengine.com/showthread.php?59812-Free-Foliage-Starter-Kit for more information.

- The **intersection** area. This is a trigger box that covers the whole intersection. This is primarily used to tell the AI agent that this area contains some sort of intersection.
- The **road crossing** area. This is a trigger box that covers the pathway on each side of a road crossing. This is used to tell the AI agent that it is nearby a road crossing.
- The **road intersection** area. This differs from the **intersection** area. This area is primarily used to tell the AI that it is on a road or cycle lane, and that they now have reach the intersection where they might need to stop.

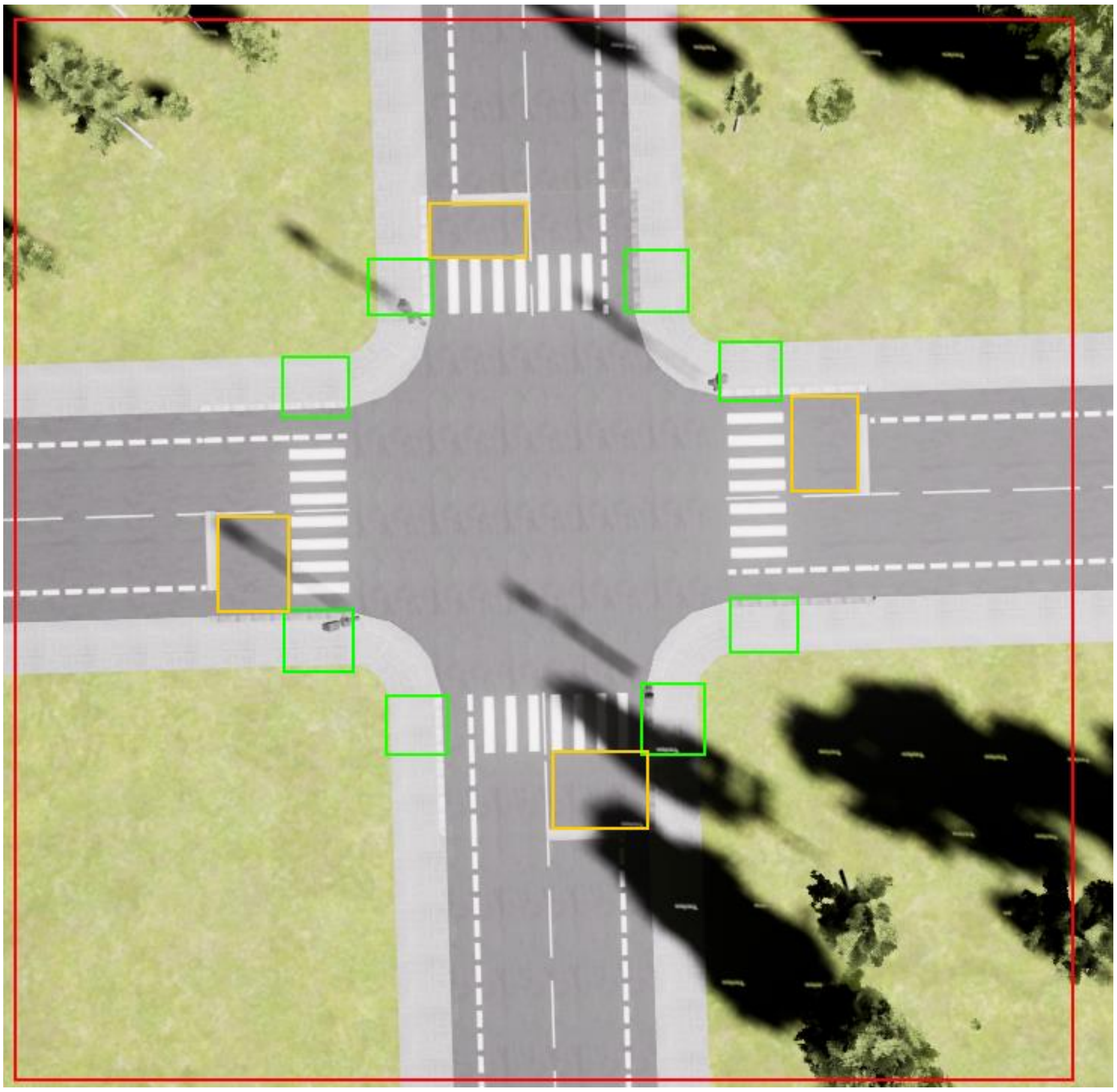


*Figure 4-11 – Trigger Areas. Everything inside a coloured square represents an area. The red square marks the **intersection** area. The yellow squares mark the **road intersection** areas. The green squares mark the **road crossing** areas.*

Movement Areas

The last thing that was needed was a way to recognise the areas that can be moved upon. This works must like the trigger areas. A rectangular box is drawn over the environment. That part is now marked as a certain type of surface. For example, the cycle lanes must be marked as a "cycle lane" surface.

The 3D environment got four types of surfaces. See figure 4-12 for a visual example.

- The **pavements**. This is the surface that marks all the pavements used by the pedestrians or cyclists.
- The **cycle lanes**. This is the surface that marks where the cycle lanes are.
- The **mixed roads.** This is the surface that marks all the mixed roads.
- The **lane direction markings.** This surface is used to mark that traffic moves in the other direction on the other side of the marking.

Each movement area has a set of values associated with itself. This is used by the AI agents to modify how much it costs (or how much they prefer one surface compared to another) to move over and enter that area.

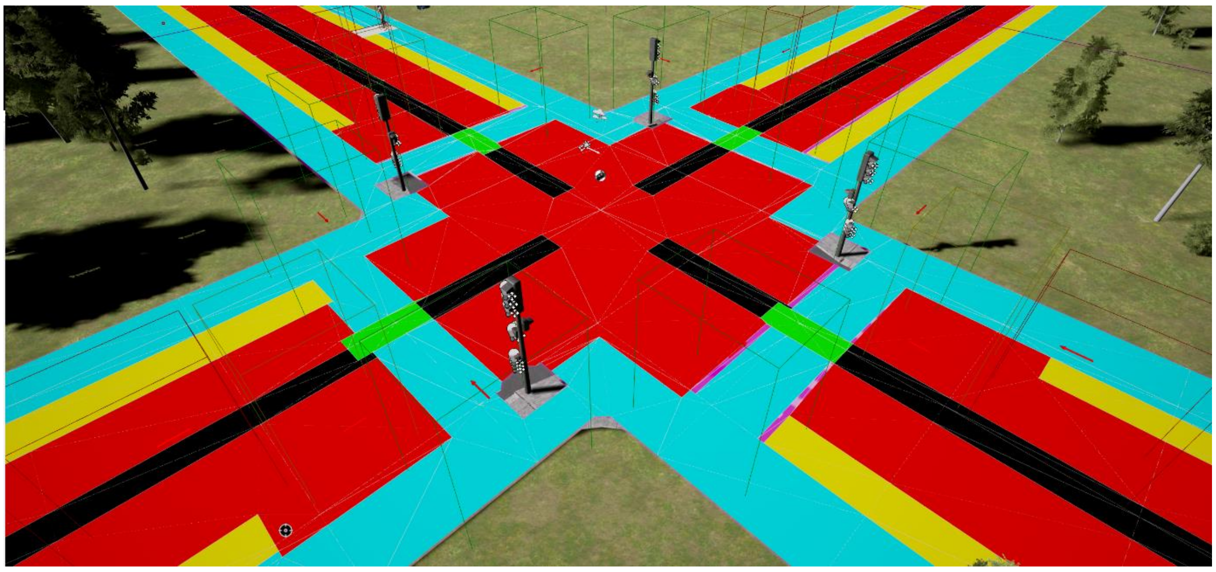


*Figure 4-12 – Movement areas in the 3D environment. The blue areas are the **pavements**, the yellow areas are the **cycle lanes**, the red areas are the **mixed roads**, and the green and black areas are the **lane direction markings**.*

The trigger areas and movement areas are the only thing the AI agent needs in order to interact with the 3D environment. One can prefabricate much of this to make it easier to create new 3D environments and scenarios. For example, the traffic light model comes with three trigger areas; Two road crossing triggers and a road intersection trigger. This makes it easy to add a new traffic light to an environment. In the same way the whole model of the intersection can be made to include the traffic lights and the movement areas. In this way one only needs to make the models once.

4.3 Controlling the Cyclist

This section will explain how the AI agent autonomously controls the cyclist model. The previous section described the requirements for implementing a 3D environment. It described how the AI needed to know what surface it is on, what area it is in, and a way to interact with the traffic lights. Together, section 4.3 and 4.4 will explain why the AI agent needs this information and how the AI agent uses this information to simulate human behaviours for the cyclist model.

4.3.1 The AI Agent

The AI in Unreal Engine is contained in an AI-Controller class. In this thesis I have referred to an instance of this class as an **AI agent**. One AI agent can control one model in the 3D environment. There can be several models in a 3D environment and each model is controlled by an AI agent.

In chapter 2 it was explained how the existing simulator uses a real bicycle to control a cyclist model in the 3D environment. The AI Controller will replace this hardware bicycle, that was operated by a person, by providing the same inputs autonomously. The cyclist model is therefore said to be controlled by an AI Controller. Combined, the **cyclist model** and the **AI Controller** is what have been referred to as an **AI agent**. The AI agent got access to all the input for a cyclist model. This meant that the AI agent needed a way to control these inputs.

The AI agent got two components. These components work together to control the cyclist and simulate a given personality. The first component is the **path-following component.** This component is responsible of directly controlling the cyclist model. This is done by sending the desired input values to the cyclist model. For example, how much should be handlebar be turned and should the handbrakes be pressed in. The second component is the **behaviour tree.** This component is responsible for simulating different personalities for a cyclist. For example, it controls what lane and speed the cyclist prefers. These components are again built up of smaller parts, which will be explained later. The AI agent (or rather the AI Controller) is used to abstract these components into a single entity. Figure 4-13 shows an illustration of the components. This shows how the two components communicates between themselves, and also how they communicate with the 3D environment and cyclist model.

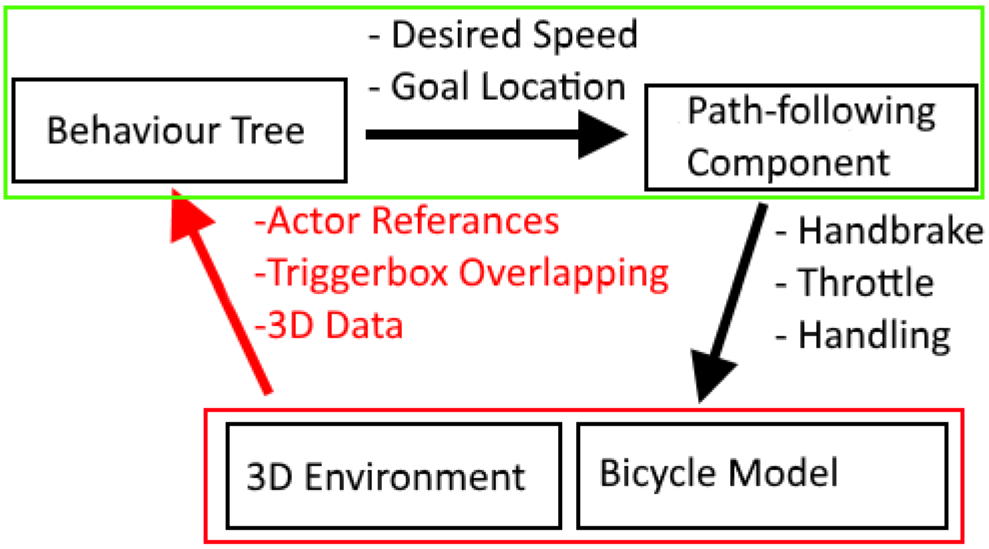


*Figure 4-13 - The different components in the projects implementation. The **red** square represents the 3D environment and the cyclist model. These are physical models contained in the 3D environment. The **green** square represents the AI agent. These components are part of the AI Controller. They control the cyclist model and queries the 3D environment for information. The arrows and text explains what information is sent between the different components.*

4.3.2 Controlling the Cyclist

The AI agent got access to all the inputs on the cyclist model. Practically, this means it can control the **throttle**, **handbrakes** and **handlebar**. I have assumed that the cyclist will have a goal to reach. The AI agent must use the inputs to navigate the cyclist through the 3D environment to reach this goal. Controlling the cyclist boils down to finding the best path to a goal, and then actually moving to that goal. This is done by the **path-following component**.

Finding a Path

Unreal Engine 4 uses a shortest-path algorithm to find the shortest path from an actor to a goal. In the previous section it as explained how the 3D environment was separated into **movement areas**. This made it possible to specify a different cost for moving into and though the different areas. For example, it could cost more to cycle on a mixed road, as there are more rules and laws to follow, compared to a pavement. This might be true for some cyclists, but not for others. Each cyclist type will therefore define their own preferences. These preferences might change over time, dependent upon factors like traffic. The shortest-path algorithm will take this cost into account when finding a path to the goal. The result of running the algorithm is a set of straight lines, starting from the cyclist and ending at the desired goal. The lines form a path though the different movement areas, dependent upon the cyclist's preference of using a given area.

Following a Path

One now has a set of straight lines, that forms a path from the cyclist to the cyclist's goal. Unfortunately, Unreal Engine have no way for a vehicle to follow this path. The built in AI in UE4 is meant for humanoid characters and uses a different movement system than vehicles. This is where the path-following component comes in. It will interpret this path and make the cyclist model follow it. The path-following component uses the throttle, the handbrake and the handlebar to accomplish this.

Controlling the Throttle

It makes sense that a cyclist will follow a given speed that they are comfortable with. This might be the speed limit, or some other value. The cyclist model allows you to control the throttle, or pedals, based on a value ranging from -1.0 (brakes/reverse) to 1.0 (full throttle). For a cyclist this translates to how hard they are pedalling in a given gear, thus one can ignore reverse. The cyclist will have a desired speed, in km/h, that they want to follow. The problem is that there is no relationship between the throttle input and this desired speed. This is solved by using a **proportional–integral–derivative controller** (PID controller). The idea of using a PID controller to control the throttle is taken from the work done by Johansen and Løvland (2015). From an outside perspective the PID controller takes in an **error value** and a **delta time** as input, and returns a suggested adjustment to a **control variable** in return. The **error value** is the difference between the **desired value** and **current value** of the measured variable, while the **delta time** is the time since the last measurement. In this case, the error value will be the *desired speed in km/h* minus the *current speed in km/h* for a cyclist. The output value will be a suggested adjustment to the throttle, based on the error in speed, in order to reach the desired speed.

For example, the current speed of the bicycle is 15 km/h. Due to some traffic up ahead the cyclist wants to slow down to 7 km/h. The throttle is at 0.54. The input to the PID controller will be -8 km/h (7 km/h – 15 km/h). The PID Controller will most likely suggest a negative adjustment to the throttle, in order to reach the desired speed of 7 km/h.

A PID controller continuously calculates an error value as the difference between a desired value and a measured variable. The controller attempts to minimize the error over time by the adjustment of a control variable, such as the throttle in this thesis, to a new value determined by a weighted sum:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\partial\tau + \ K_d \frac{\partial e(t)}{\partial t}$$

where $K_p, K_i\ and\ K_d$, all positive, denotes the coefficients for the proportional (P), integral (I) and derivative (D) terms.

- *P* accounts for present values of the error. In this case the different in the current and desired speed.
- *I* account for the past values of the error. If the current value is too weak to reach a desired speed, error will accumulate over time, and the controller will respond by applying a stronger change.
- *D* account for the possible future value of the throttle, based on the current rate of change.

The coefficients (Ks) must be adjusted to get the following properties:

- **Stability**. This means that the throttle makes the speed converge toward the desired speed. This is often related to the oscillation around the desired value. In this case the speed needs to reach the desired speed without over-/undershooting it. For example, the speed needs to reach 15 km/h without first going to 20 km/h, then down to 12 km/h, then up to 17 km/h, and so on until it is stable around 15 km/h.
- **Quick reaction**. This means that the throttle must react quickly to changes in the desired speed. It must be able to quickly go from 15 km/h down to 0 km/h, then up again to 10 km/h. This must be done at the same time as the system is stable, or this will give very twitchy behaviour for the cyclist.

Often it is enough to only have a P, PI or PD controller. This is done by setting the missing coefficient to zero. A **P-controller** can be enough to control a system, but it can often lead to oscillation due to over/undershooting. For example, the throttle is quickly increased to 1.0 to accelerate from 0 to 15 km/h. When it reaches 15 km/h, the throttle is still at 1.0. This causes the car to continue to accelerate past 15 km/h. This happens due to lag in the system. The system uses some time to react to the changes in the throttle. This can be fixed by using a **PD-controller.** This uses the current rate of change of the error value, to predict the future value of the throttle. Without explaining the math, this adjusts the throttle to faster stabilise the current speed around the desired speed. A **PID-controller** is not necessary in this case, as you do not care about the previous values of the error (the I in PID). This is due to the frequent and big changes in the desired speed.

So far, the PID controller (or PD controller in this case), would work without any knowledge of what it is adjusting. It just saves the previous error and does the calculation. If one also has a **minimum** and **maximum** adjustment value, then one could clamp the suggested adjustment value between those. This makes it possible to control the growth rate per adjustment call, which makes it easier to tune the PD controller.

All that remains is to adjust the **minimum** and **maximum adjustment values** and the **coefficients for P and D** in order to get the desired behaviour for the cyclist's speed. That is a **stable** system with **quick** reactions. This must be done by trial and error by testing the system. One also needs another system that tells the PD controller the desired speed. This is part of a cyclist's behaviour and will be sent to the path-following component, and thus to the PID controller, by the behaviour tree component.

Controlling the Handbrakes

There is no need for an extra system to control the brakes. One can just apply the brakes when the desired speed is zero, and apply the brakes a little, when the throttle is zero. This is due to the PD-controller that handles the throttle. This makes the cyclist stop as fast as possible when it is needed.

Controlling the Handlebar

The cyclist model that is used in the simulator can only turn by using the handlebar. It will not turn by leaning the bicycle to the sides, which would happen in the real world. This makes it much easier to controller the bicycle. Like the throttle, the handlebar is adjusted by setting an input variable between -1.0 and 1.0. Here -1.0 is the maximum turning angle to the left, 0.0 is straight forward and 1.0 is the maximum turning angle to the right. This sounds like a good use for another PID controller, which one could use in theory. But there is no need to make this more complex than necessary. One can instead use another system. This system is based on the **maximum turning angle** and the **delta angle** between the **current movement direction** and the **desired movement direction**. This will be used to set the input to the handlebar.

The **maximum turning angle (MTA)** is the biggest angle one can turn the handlebar (I will use 85 degrees in this project). The **current movement direction** is the forward vector of the cyclist. The **desired movement direction** is the vector between the next path goal (the end of the current straight line that the cyclist is following) and the cyclist's current location. These two vectors are then used to calculate the **delta angle (DA)** between the current direction and the desired direction.

The handlebar input is adjusted by clamping the **DA** to a value between -1.0 and 1.0. This requires the following rules. Keep in mind that 0 degrees means straight forward.

1. **If (DA > -MTA) and (DA < MTA)**. Clamp the DA to a value between -1.0 and 1.0, by linear distribution, and set it as the input to the handlebar. For example, if MTA equals 85 degrees, then -1.0 equals 85 degrees to the left, -0.5 equals 42.5 degrees to the left and 0.0 makes the bicycle go straight forward.
2. **If (DA < -MTA) and (DA > -180 degrees).** Set the input to the handlebar to -1.0.
3. **If (DA > MTA) and (DA < 180 degrees).** Set the input to the handlebar to 1.0.

In point 2 and 3, the angle must be limited between -180 and 180 degrees. This is because it would be better to turn the other way, if the delta angle is less/more than +- 180 degrees.

This system now got direct control over the handlebar, it is precise and reacts instantaneously. The problem is that it might be a bit unrealistic. For example, you would not turn the handlebar 85 degrees to the left at a speed of 20 km/h. This would also not happen immediately. To fix this there is a system that combines the throttle and the handling.

Combining the Throttle and Handlebar

It should take some time to turn the handlebar. This is already implanted in Unreal Engine’s vehicle system. This means you can adjust how fast the handlebar should turn. For example, one can say that the handlebar will take one (1) second to turn from 0 to 85 degrees to the left.

Another system already built in to UE4 is the “handling-curve” graph**.** This allows one to limit the turn of the handlebar based on the current speed of the bicycle. For example, at 20 km/h one cannot turn the handlebar more than 20 degrees to the left or right. Then, the slower one goes the more one can turn the handlebar, so at 0 km/h one can turn it 85 degrees (if this is the maximum turn angle).

The last feature that is needed is a way to link the throttle to the handling. The idea is to make the cyclist slow down before they turn. This is done before the **desired speed** is sent to the PD-controller with this formula:

$$a = d - (d * |(h)| * 0.9)$$

where **a** is the *adjusted desired speed* that is sent into the PD-controller, **d** is the *original desired speed* and the absolute value of **h** is the *current handlebar input* (which is a number between -1.0 and 1.0). If the cyclist wants to turn they will slow down, and therefore turn faster as a result due to the handling-curve described above. It also makes sense to slow down some before the cyclist turn, but only as a portion of how much they want to turn. The 0.9 in the formula makes sure the cyclist moves forward when the cyclist wants to turn 90 degrees or more in either direction.

Avoiding Traffic

So far there have been no mention of traffic or how this is handled. This is because the path-finding system automatically handles this, but with some limitation that must be mentioned. This is done by adding a different area type (like the different lane areas) around each type of actor. For example, all cyclists will have a **cyclist area** around themselves. This must be **actor type** specific, and not actor specific. This means you only need one new area for each new actor type, instead of one new area for each new actor you add to a scenario. A car, cyclist or pedestrian actor will thus have its own movement area around itself. This area will be assigned as off-limit for other traffic types. The path-finding algorithm will then try to find a path that goes around these areas.

This sounds like a perfect solution, but the problem is that actors of the same type cannot detect each other. The reason is hard to explain, but the solution is a little easier, so try to follow the reasoning. All that needs to be done is to move the starting point for the path-finding algorithm. It must be moved forward to a location just outside of the area surrounding the actor. This makes it possible to set that area to off-limit. This is because the starting point for finding a path, from the actor to a goal, is at the centre of the actor. Thus, if the area type is set off-limit without moving the starting point, then the path-finding algorithm will be confused. This is because it cannot find a path though the off-limited area. Unfortunately, the starting point is not moved in the current implementation, so actors of the same type cannot detect each other.

4.4 Simulating Cyclist Behaviours

The previous section described how the system for controlling the cyclist model works. It mentioned how the path-following component controls the cyclist. The other component in the AI agent is the **behaviour tree**, which is used to give the controlled cyclist model a personality. This section will first explain what the behaviour tree components does. Then it will go into details on how the behaviour tree component works. By the end of this section it should be clear how the AI agent autonomously controls a cyclist model and how it gives the cyclist a personality.

4.4.1 Behaviour Tree and Blackboard

There are two important parts to the **behaviour tree** component in UE4. There is the **behaviour tree** itself and the **blackboard.** The blackboard stores data for a given behaviour tree. This means that any node in the behaviour tree can access that data. This is the easiest way to share data between nodes in the BT. The blackboard can store any kind of data. It could for example be locations, reference to other actors in the 3D environment, booleans or enums.

There are two main tasks that the behaviour tree must do to get a working AI agent. Both have been explained earlier. One is that the path-following component needs a desired speed to work. This means that the BT must find a desired speed. The other task is to find a goal, start the pathfinding and adjust the cost of moving over different movement areas. Those two tasks are the minimal requirement to get the AI agent to control the cyclist model. Some of the other tasks the behaviour tree will do are:

- Look for and interact with traffic.
- Observe the environment. For example, look for intersections, road crossing, buildings or traffic lights.
- Interact with traffic lights.

Desired speed

In the previous section it was explained how the cyclist was controlled. The input that was required to the path-following component was the desired speed to the PD-controller. A node in the BT will therefore find and update the desired speed. There is some information you need when updating the desired speed. This information will be used to decide what the desired speed should be. For example, it would be nice to slow down if the cyclist is in an area with an intersection, or if there are pedestrians in front of the cyclist. This information will be found by other nodes in the behaviour tree.

Pathfinding Adjustments

The previous section also talked about pathfinding and following. It explained how the cost of different movement areas would affect the path that the cyclist would take. A node in the behaviour tree will adjust these costs. For example, if there are cars on the road, it is most likely better to use the cycle lane. The node can adjust the cost so that the cyclist will switch to the cycle lane. Other nodes will look for factors that might affect this cost. This could be the same nodes that looked for information for adjusting the speed.

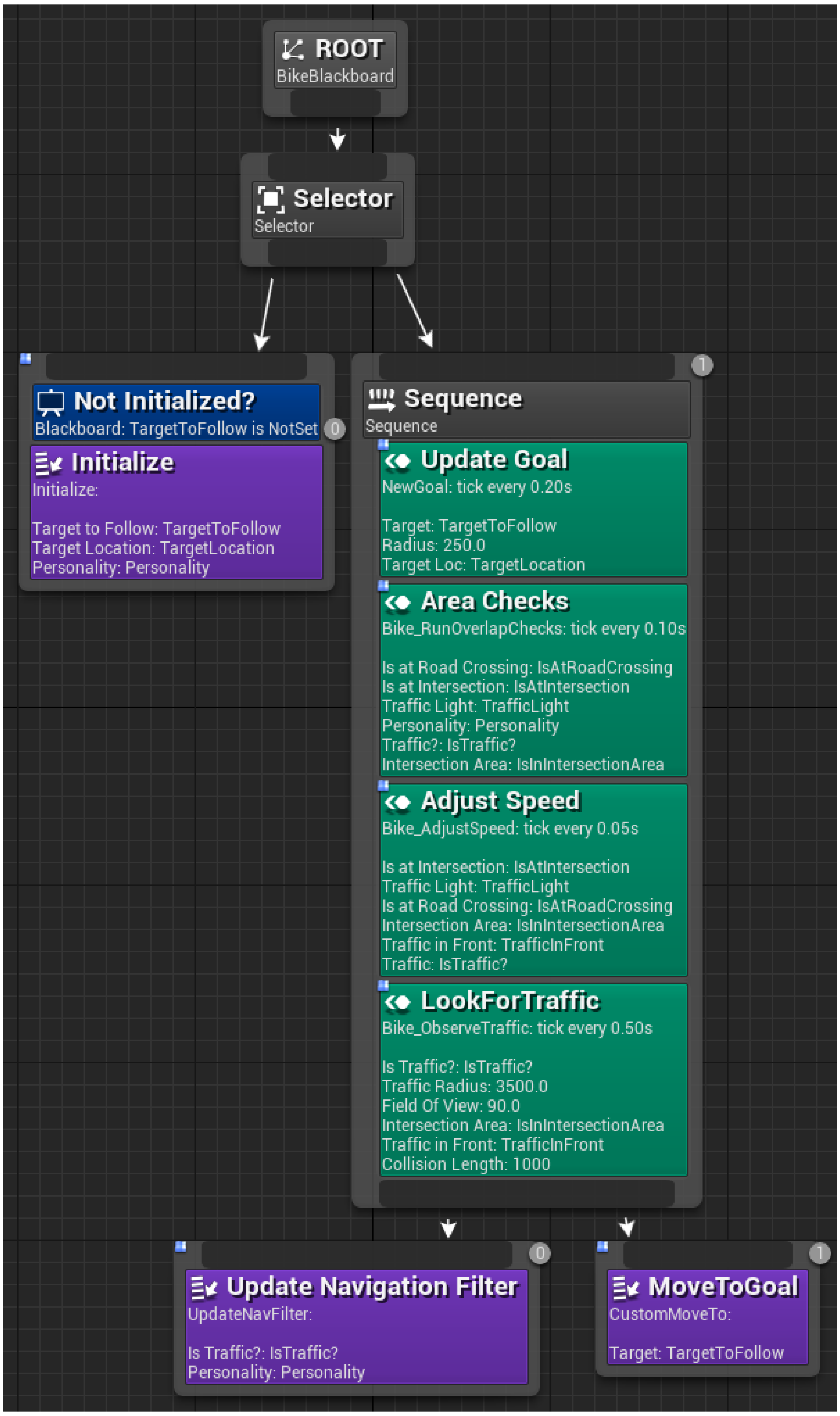
Personalities

So where does the personalities come from? The simple answer is, by adjusting the desired speed and the costs of using different lanes (movement areas). The longer answer is a bit more complex. Each cyclist model saves an enum value which defines its personality. This value is either **child**, **leisure** or **transport** cyclist. The behaviour trees use this personality to adjust other variables. For example, a child will have a different desired speed than a transport cyclist. Another example might be that a child reacts more strongly to traffic than other cyclists, so they will be more careful if there is traffic.

4.4.2 The Cyclist Behaviour Tree

This subsection will present the behaviour trees that is used in the solution. Figure 4-14, on the next page, shows the finished behaviour tree that is used by the AI agent. Each node in the tree will be explained in detail.

Each iteration of the BT starts in the **root** node. See figure 4-14. The flow then goes from top to bottom and from left to right when the tree is traversed.


ROOT
BikeBlackboard

Selector
Selector

Not Initialized?
Blackboard: TargetToFollow is NotSet

Initialize
Initialize:

Target to Follow: TargetToFollow
Target Location: TargetLocation
Personality: Personality

Sequence
Sequence

Update Goal
NewGoal: tick every 0.20s

Target: TargetToFollow
Radius: 250.0
Target Loc: TargetLocation

Area Checks
Bike_RunOverlapChecks: tick every 0.10s

Is at Road Crossing: IsAtRoadCrossing
Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Personality: Personality
Traffic?: IsTraffic?
Intersection Area: IsInIntersectionArea

Adjust Speed
Bike_AdjustSpeed: tick every 0.05s

Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Is at Road Crossing: IsAtRoadCrossing
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Traffic: IsTraffic?

LookForTraffic
Bike_ObserveTraffic: tick every 0.50s

Is Traffic?: IsTraffic?
Traffic Radius: 3500.0
Field Of View: 90.0
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Collision Length: 1000

Update Navigation Filter
UpdateNavFilter:

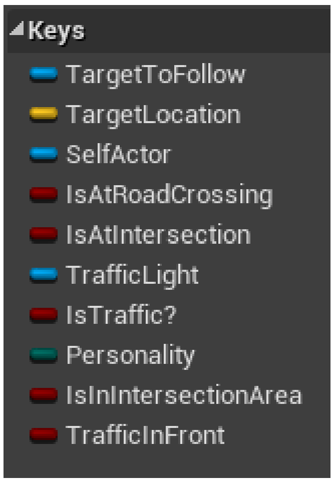
Is Traffic?: IsTraffic?
Personality: Personality

MoveToGoal
CustomMoveTo:

Target: TargetToFollow

Figure 4-14 – The finished ***behaviour tree*** *used in the tests. The grey frames indicate a node in the BT. The purple squares in a node are* ***tasks****, the green squares are* ***services****, and the blue square is a* ***condition****. The text in each square are input values to a node. You can assign values to the input, usually taken from the* ***blackboard****, by clicking at a node in the editor.*

The behaviour tree needs a **blackboard.** This is used to store and retrieve different values. These values can be accessed by all the nodes in the BT. The nodes only need the specific key to get a given value. Figure shows the blackboard keys used by the BT.



Keys
- TargetToFollow
- TargetLocation
- SelfActor
- IsAtRoadCrossing
- IsAtIntersection
- TrafficLight
- IsTraffic?
- Personality
- IsInIntersectionArea
- TrafficInFront

*Figure 4-15 – The **blackboard** with keys that can be used to store/retrieve data. The **blue** keys can access Objects (For example, a reference to a traffic light or car). The **red** keys can store Boolean values. The **yellow** keys can store a 3D vector value (a location or a direction). The **green** keys can store enums, which in this case is a personality type.*

As one can see there are some different values stored in the blackboard, but most of the stored data is Boolean values (red nodes) that will determine different conditions.

- **TargetToFollow** – This is the goal for a cyclist. It can be a moving target or a static location that the cyclist must reach or follow.
- **TargetLocation** – This is the location of the goal.
- **SelfActor** – This is just a reference to the cyclist itself.
- **IsAtRoadCrossing** – This is a Boolean that is true of the cyclist is at a road crossing <u>that it wants to cross over.</u>
- **IsAtIntersection –** This is a Boolean that is true if the cyclist is in the mixed lane at an intersection.
- **TrafficLight –** This is an Object reference to the traffic light that is connected to an intersection or road crossing.
- **IsTraffic?** – This is a Boolean that is true if the cyclist can see any traffic in an intersection area or a given distance around themselves. This is limited by the field of view for the cyclist.
- **Personality –** This is the given personality of a cyclist.
- **IsInIntersectionArea –** This is a Boolean that is true if the cyclist is in an area that contains an intersection.
- **TrafficInFront –** This is a Boolean that is true if there are pedestrians in front of the cyclist. This is limited to pedestrians that are no more than X meters in front of the cyclist.

All these values are constantly accessed by the nodes in the BT. Most specifically they are updated by the different **service** nodes, but used by any node that needs them.

4.4.3 The Selector Node

A **selector** node succeeds when one of its children succeeds. This means it will execute its children from left to right. If any of them succeeds the Selector node stops and returns its status to the parent.

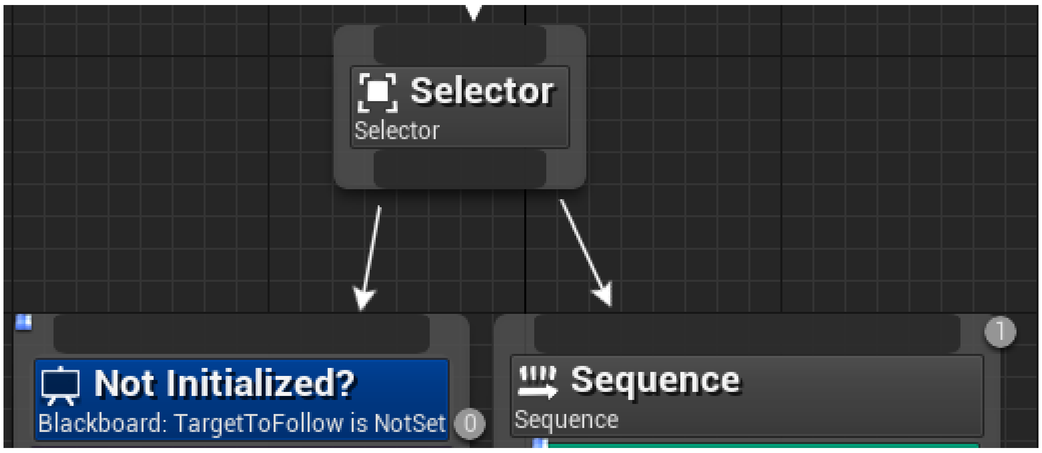
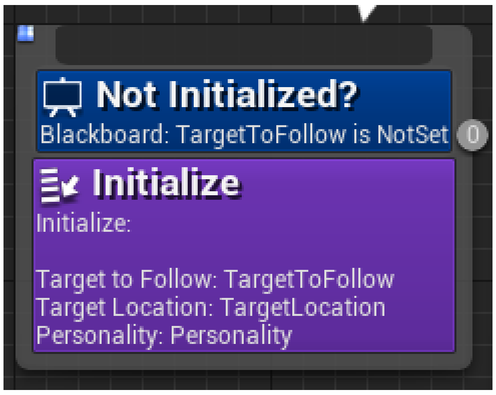


Figure 4-16 – Selector node from the behaviour tree in figure 4-14. It succeeds if either child node 0 or 1 succeeds (number in the grey circles).

The purpose of the selector node is to check if all the essential parameters in the **blackboard** is set. This is done by running the **initialize** node (Child 0 in Figure 4-16)**.** If this node succeeds, then the BT was not initialized, so next time the BT gets to this node it will fail (since the BT is initialized this time). The selector will execute the next node (Child 1 in Figure 4-16) when the initialize node fails.

4.4.4 The "Initialize" Node

This node is responsible for initializing all the essential values that is stored in the blackboard. These values are the **target** that the cyclist should try to reach, the **location** of that target and the **personality** of the cyclist. The next node in the BT requires all these values to be set, thus the need of a selector node as the parent.



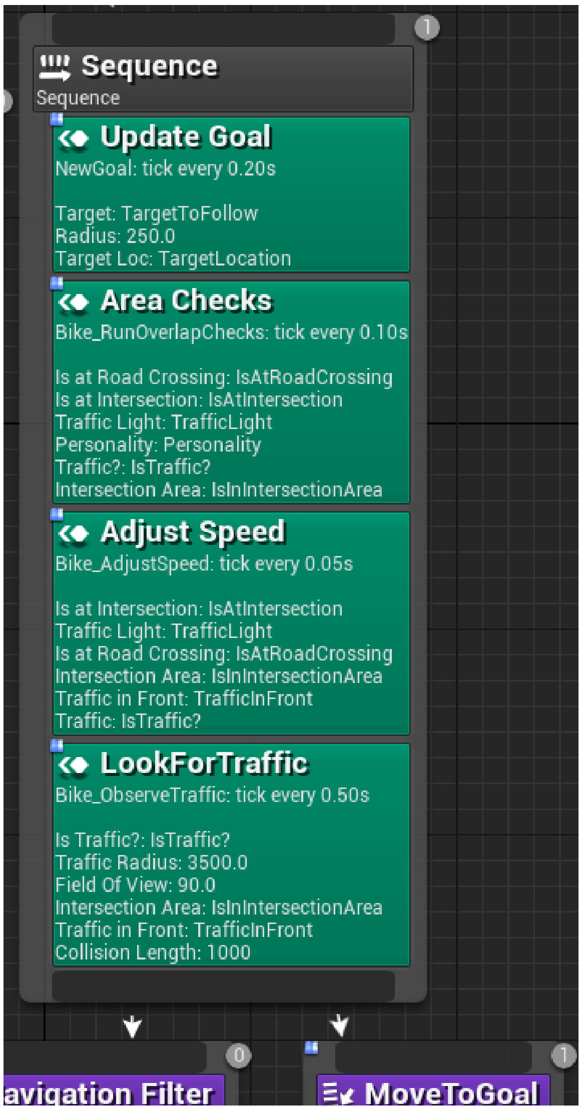
Not Initialized?
Blackboard: TargetToFollow is NotSet
0
Initialize
Initialize:
Target to Follow: TargetToFollow
Target Location: TargetLocation
Personality: Personality

Figure 4-17 – The initialize node from the behaviour tree. The text represents input parameters, which usually are different keys from the blackboard.

The task nodes are not pre-defined. This means they must be implemented/written manually. This works like any other method, you got some pre-defined functions and need to write the rest. For example, get/set-functions for variables or a GetActorLocation function for returning a specific actor location in the game world. These must be combined together with some custom code to form the finished initialize node.

4.4.5 The Sequence Node

This is the most important node in the BT. This node will be executed with each run of the BT, as long as the BT is initialized. It is a **sequence** node with four **services** and two **tasks**. A sequence node succeeds if all its children succeed. In this case the two task nodes **UpdateNavigationFilter** and **MoveToGoal**. See figure 4-14.



Sequence
Sequence

Update Goal
NewGoal: tick every 0.20s

Target: TargetToFollow
Radius: 250.0
Target Loc: TargetLocation

Area Checks
Bike_RunOverlapChecks: tick every 0.10s

Is at Road Crossing: IsAtRoadCrossing
Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Personality: Personality
Traffic?: IsTraffic?
Intersection Area: IsInIntersectionArea

Adjust Speed
Bike_AdjustSpeed: tick every 0.05s

Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Is at Road Crossing: IsAtRoadCrossing
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Traffic: IsTraffic?

LookForTraffic
Bike_ObserveTraffic: tick every 0.50s

Is Traffic?: IsTraffic?
Traffic Radius: 3500.0
Field Of View: 90.0
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Collision Length: 1000

*Figure 4-18 – The sequence node from the behaviour tree in figure 4-14. The two child tasks can be seen at the bottom (grey circles with numbers). The **green** squares are services. The text inside a service node indicates input values or blackboard keys that the service node will use.*

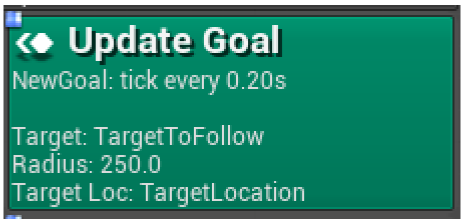
The reason for this being a sequence node is that you always want to update the **navigation filter**, which controls the cost of moving over different movement areas (or lanes), before you call the MoveToGoal node. Therefore, if the UpdateNavigationFilter node succeeds, then the MoveToGoal node is executed**.**

4.4.6 The Service Nodes

The service nodes are called every X seconds for as long as the parent node (or any of its children) is executing. In this case that means they are called every X seconds after the BT is initialized, since the sequence node always is executing at that point. The service nodes work on separate threads and does not interrupt the other nodes in the BT or each other. Each service has a different purpose, but they all read or update values in the blackboard. What values are updated is based on the input keys to each service.

The Update Goal Service

This service is quite simple. All it does is to check the distance to the goal location. It will find a new goal if the cyclist is within the accepted radius from the current goal. Each goal point can store a follow-up goal point. This will be used as the next goal for the cyclist. The goal is then updated in the blackboard by updating the **TargetToFollow** and **TargetLocation** values.



Update Goal
NewGoal: tick every 0.20s

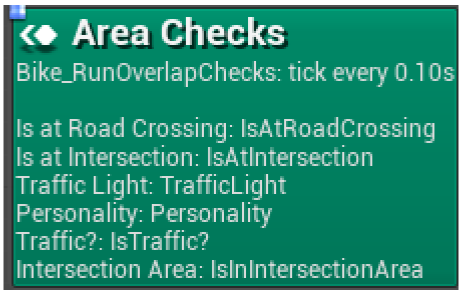
Target: TargetToFollow
Radius: 250.0
Target Loc: TargetLocation

Figure 4-19 – The update goal service. It is called every 0.2 seconds. It got three inputs: Two blackboard keys and a float number.

This feature can be useful. For example, if the cyclist needs to go from A to B, but must reach C along the way, then A can store goal C and C can store goal B. This is necessary if C is not along the best (or relative shortest) path from A to B for a given cyclist.

The Area Check Service

This service will use the trigger areas described earlier to determine different conditions. The blackboard values updated by this service are **IsAtRoadCrossing**, **IsAtIntersection**, **IsInIntersectionArea** and **TrafficLight.** Each value is described earlier in this section. All this service does is to update these values based on the conditions stated previously.



Area Checks
Bike_RunOverlapChecks: tick every 0.10s

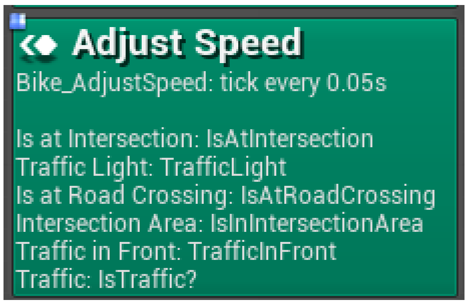
Is at Road Crossing: IsAtRoadCrossing
Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Personality: Personality
Traffic?: IsTraffic?
Intersection Area: IsInIntersectionArea

Figure 4-20 – The area check service. This must be called as often as possible. In this case every 0.1 second. The service got several input-keys from the blackboard.

The Adjust Speed Service

This service is responsible for updating the desired speed for the cyclist. This is the desired speed that is sent to the path-following component to make the cyclist move. This service takes several factors into consideration. Each will be described to better understand what happens.

- If the cyclist is in an intersection area, they might be more careful. This is done by lowering the speed of the cyclist in these areas.
- The speed is also adjusted by personality. For example, a transport cyclist will probably hold a higher speed then a child.
- Pedestrians in front of the cyclist. The law in Norway says that cyclists must be careful around pedestrians. This can be achieved by slowing down.
- The desired speed is set to zero if the cyclist wants to stop in an intersection or road crossing.
- Traffic can affect how fast the cyclist moves. This must be adjusted here.


Adjust Speed
Bike_AdjustSpeed: tick every 0.05s

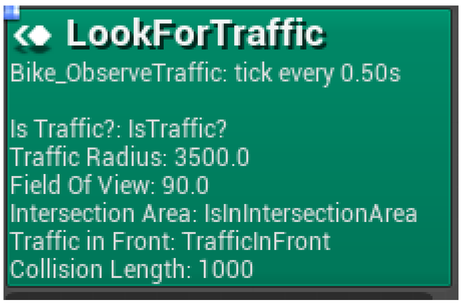
Is at Intersection: IsAtIntersection
Traffic Light: TrafficLight
Is at Road Crossing: IsAtRoadCrossing
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Traffic: IsTraffic?

Figure 4-21 – The adjust speed service. This is called every 0.05 seconds. It has several input-keys from the blackboard.

Each factor must also be considered together. Some factors override other, while some can be combined to have different effects on the speed. There are many possibilities, but each one makes the cyclist's behaviours more realistic.

The Look for Traffic Service

The name of the service explains the task quite well. This service works as the cyclist's eyes and will look for traffic, both around and in front of the cyclist. If there is traffic it is saved in the blackboard with the **IsTraffic?** and the **TrafficInFront** keys. There are some limitations to if the cyclist will see the traffic. This is related to the field of view and line of sight.


LookForTraffic
Bike_ObserveTraffic: tick every 0.50s

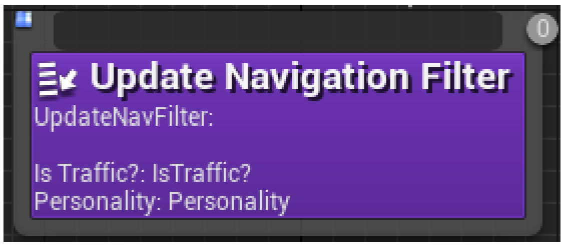
Is Traffic?: IsTraffic?
Traffic Radius: 3500.0
Field Of View: 90.0
Intersection Area: IsInIntersectionArea
Traffic in Front: TrafficInFront
Collision Length: 1000

Figure 4-22 – The "look for traffic" service. This is only called every 0.5 second. It got some input-keys form the blackboard and some float number to control field of view, radius for traffic detection and pedestrian detection.

When the cyclist is in an intersection, they can see all traffic in their line of sight. If they are not in an intersection, then only traffic in their field of view and in their line of sight is seen. If any of these conditions are meet, then the **IsTraffic?** value is set to true. **TrafficInFront** is only set to true of there is pedestrians in front of the cyclist. Here the pedestrians must be directly in front of the cyclist and closer than ten meters.

4.4.7 The “Update Navigation Filter” Node

This node is used to update the cost of moving over different movement areas. It is the first child of the sequence node. This is to make sure the movement costs are updated before the path-following component finds a path and starts to control the cyclist model.



UpdateNavFilter:

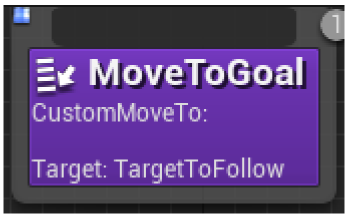
Is Traffic?: IsTraffic?
Personality: Personality

Figure 4-23 – The “update navigation filter” node. This is called once for every iteration of the BT. It uses two input-keys from the blackboard to do its work.

The node uses the **personality** and **traffic?** value stored in the blackboard. This is because the cost of using different movement areas (or lanes) is related to the cyclist’s personality. For example, transport cyclists like to use the cyclist lane, so that is cheaper to move over. While the child cyclist prefers the pavement. The cost is also related to traffic. For example, if a leisure cyclist sees a lot of traffic, then the cost of using the mixed road is higher.

4.4.8 The “Move to Goal” Node

This is the node that starts the pathfinding for a given cyclist. This node will finish as soon as the cyclist starts to move, but the path-following component will continue to operate.



CustomMoveTo:

Target: TargetToFollow

Figure 4-24 – The “Move to goal” node. It is only called if the “Update navigation filter” task is successful. It just uses one input-key from the blackboard, but it also got access to the cost of moving over different areas.

Whenever the BT reaches this node again, and if the cyclist still is trying to reach the goal, it will just update the goal. This means that the goal can be updated at any time, and the path-following component will be told of the changes.

5 Results

The results of the research are split into two parts. The first is the case study and its results and the second is the simulator, with the AI implementation, in Unreal Engine 4. The code is included in the ZIP file with the thesis or is available at GitHub (Abrahamsen, 2016b). The address is: https://github.com/runeabrahams1/Unreal-Engine-4-Bicycle-Simulator-with-AI. Instructions for installing and using the code can be found in the GitHub repository or in Appendix B. It is not necessary to install the code, unless you want to inspect it closer or use the solution.

The case study consisted of six tests, each done by the different cyclist personalities. These tests were recorded and analysed to get the results presented in the chapter. Each video was then presented to the two transportation researchers as mentioned in chapter 3. The questions in Appendix A was asked during and after the presentation. The results from the presentations will be used to influence the final results and conclusions.

5.1 The Test Videos

It is recommended to watch the videos of the tests as soon as you have read section 5.1. This makes it much easier to understand the results and discussion. There are six test videos included with this thesis as a ZIP file. Alternatively, the videos can be found on YouTube (Abrahamsen, 2016a). The address is: https://www.youtube.com/playlist?list=PLyQ2lVssCiGuufULltdKYA5pcY2pwBqiy. Each video presents one of the test cases in the case study. The videos start by naming the test case. Then each cyclist type is put through the test under the same conditions. The test subjects have no memory of the tests, due to them being computer-simulated cyclists. This means that one can run the tests several times for each subject to verify a consistent result. Each cyclist has therefore taken each test three times to verify the outcome. This is not part of the videos, as the outcome was the same each time, with some very minor differences.

5.2 Test Results

This section will present the result of each test case, while section 5.3 will present the results for each cyclist type. This section gives an overview of how well the cyclists performed in each test. Each different cyclist completed the tests. This means that the result depends on how well the cyclists did in the tests.

The overall results are very good. There are some technical issues, but nothing that have a major impact on the results. The biggest problem was that the cyclists moved toward the driving direction in some lanes. This happened in test five when some cyclists used the cycle lane in the wrong direction. There cyclists could also have adjusted their speed better, based on their personality. This is part of the implementation already, but it not utilized very well. The animations could also be improved to make the simulations more realistic. For example, the cyclist could choose to walk over a road crossing, use signs when turning or move their head when looking around.

5.2.1 Test 1 – Choosing a Lane

This test was meant to show that the basic AI agent with personalities were working. All the cyclists completed this test without problems. Nothing unexpected happened. The cyclists picked a lane they preferred and completed the test. This shows that the AI agent can control the cyclist model and simulate simple personalities.

5.2.2 Test 2 – Intersection

The goal was to enter an intersection and cross to the other side. All the cyclists completed this task without problems. The test shows that the AI agent can understand more complex environments or simulate more advanced behaviours. The cyclists also picked different solutions in the test. The cyclists showed clear differences by using different lanes and crossing the roads with or without a green light.

5.2.3 Test 3 – Crossing the Road

This was meant as a test to how that the systems work in a different scenario then test 2. All the cyclists completed this scenario without problems. This confirms that the AI agent also works in a different scenario. The cyclists also picked reasonable solutions based on their personality.

5.2.4 Test 4 – Choosing a Lane in Traffic

This was the first test with traffic. It was meant to give a basic confirmation that the AI agent can understand traffic. All the cyclist completed this test without any major problems, though there were some minor issues. The issues were related to navigating around the traffic in time, but it has little impact on the overall results. The result showed that the cyclists can navigate and make choices with traffic around them.

5.2.5 Test 5 – Intersection with Traffic

This is probably the most complex test in the set. The cyclists had to navigate through an intersection with traffic. The test was completed by all the cyclists without any major problems. There were some minor technical problems, but overall the cyclists made reasonable choices. This shows that the AI agent can understand complex scenarios, even with traffic. The cyclists also made reasonable choices based on their personalities. This shows that the AI agent can simulate believable personalities that work as intended with traffic.

5.2.6 Test 6 – Crossing a Road with Traffic

This test is meant to give a different scenario than test 5. This is to confirm that the systems work in different scenarios involving traffic. All the cyclists completed with test without any major problems. The minor problems, like last time, was technical. This did not affect the overall results. The tests show that the AI agent can control the cyclist model in traffic scenarios, and that the cyclists have reasonable behaviours.

5.3 Cyclist Results

This section looks at the results for each cyclist type. It presents how each cyclist did in the tests. The goal by presenting this is to give better understanding of how well each cyclist type performed in the tests. This will help with reaching a conclusion for the project.

5.3.1 Child Cyclist

The first cyclist personality that is simulated is the child cyclist. From the tests it is clear that the child is the most careful of the cyclists. The child always uses the pavements and road crossings, and they wait for the traffic lights to turn green. The transportation researchers also thought the child cyclists was the most believable. The child made the most reasonable choices throughout the tests. See table 5-1 for a better explanation of the results for each test. The table contains the **combined result** of the transport researcher's verifications/meanings, an analysis of the videos and the literature research done as part of this study.

Test ID	Result
Test 1	In the first test the child makes a reasonable choice of lane. They then stick to that until they reach to goal. This is what would be expected.
Test 2	The choice of route was good. The child stops at traffic lights and waits for a green light. There were some technical issues. For example, the cyclist moved out into the grass and could have walked next to the bicycle over a road crossing. Overall the cyclist made good choices and the results was as expected.
Test 3	The choice of route was good enough for a more careful child. A more experienced child could have chosen differently. It might have been better to cross the road directly. For a more careful child it was a good result.
Test 4	The choice of route and lanes was good. The child followed the pavement and avoided all the traffic. This is what was expected. There were some technical issues. The cyclist started to avoid traffic a bit late and almost ended up in the cyclist lane. This could indicate some issues with how traffic is avoided.
Test 5	This choice of route and lane was good. There were minor technical issues like the previous tests. No new issues. The behaviour was good, nothing unexpected. Overall a successful test.
Test 6	The child stayed on the pavement throughout the test, which was good. With traffic this made sense. The cyclist avoided the traffic and waited for a green light. No new problems, just the same minor technical problems as earlier. Overall a good and reasonable result.

Table 5-1 - Table with test results for the child cyclist. Each result is based on the transport research's verifications/meanings, an analysis of the videos and the literature study.

5.3.2 Leisure Cyclist

In the videos, the second cyclist type is the leisure cyclist. The results indicate that the modelled behaviour simulates a more careful leisure cyclist. The spectrum of behaviours expected from the leisure cyclist was big, as mentioned in the literature study. This is discussed more in the test results in table 5-2. The transportation researchers also agreed that the cyclist behaved like a leisure cyclist. The spectrum of cyclist variants between a careful child cyclist and an aggressive transport cyclist is quite large. In the future it might be of interest to add some more cyclist models between these two.

Test ID	**Result**
Test 1	In the first test the leisure cyclist makes a reasonable choice of lane. They then stick to that until they reach to goal. This is what would be expected.
Test 2	The result is as expected of a more careful leisure cyclist. They used the cycle lanes and pavements, but stayed of the mixed road. A more aggressive leisure cyclist would probably have used the mixed road to take some shortcuts. This could also depend on the conditions of the cyclist lane. For example, if there is water or dirt in the cycle line, then using the road should become more viable. This test assumes that the cyclist lane is in perfect condition. Overall the results were good, but the factors above should be kept in mind.
Test 3	The cyclist did a reasonable choice by just crossing the road. This is as expected. No new problems in this test. Overall a successful test.
Test 4	The cyclist makes reasonable choices for most of the test. The leisure cyclist would probably use the mixed road to avoid the obstacles in the cycle lane, not the pavement as they did in the test. This happened due to the car approaching from behind, but the cyclist should have seen this. This might indicate that more complex scenarios can become problematic.
Test 5	The cyclist made reasonable decisions. It makes sense that the leisure cyclist chooses the cycle lanes and pavement. It also made sense for the cyclist to wait for the traffic light when it saw traffic. They then crossed the second road crossing on a red light since the traffic had passed. This could be correct behaviour, but it is also possible they would have waited for a green light. Again, this point back to that the leisure cyclist type should be divided into two or more types.
Test 6	The cyclist makes reasonable decisions. They do not cross into the mixed road when there is traffic. When the traffic has passed they cross into the mixed road to reach the goal faster. Here the cyclist could have cycled to the road crossing instead and then crossed the road there. Again, both seams correct for a leisure cyclist.

Table 5-2 - Table with results for the leisure cyclist. Each result is based on the transport research's verifications/meanings, an analysis of the videos and the literature study.

5.3.3 Transport Cyclist

The transport cyclist is set to represent an efficient and more aggressive cyclist. The test results show that this is the case. The transport cyclist uses the options available to optimise the path to the goal. The researchers agree that this is the case. There are some minor issues in some of the tests; these are presented in table 5-3.

Test ID	**Result**
Test 1	The cyclist uses the cycle lane. This is as expected, but they could also have used the mixed road, as there is no traffic. Both options are reasonable given that the conditions of the lanes are equal. If the conditions of the mixed road and cycle lane are different, then the cyclist should choose the lane that is in best condition. For example, if there is water in the cycle lane, then they choose the mixed road.
Test 2	The cyclist uses the cycle lane to begin with. See the comments from test 1 about choosing a lane. Then the cyclist uses the mixed road to move past/though the intersection. The researchers agree that this is reasonable. After that, the cyclist chooses to use the cycle lane for the final piece of road. This is wrong, as there is a cycle lane on both sides of the road. They are therefore moving in the wrong direction. They should have crossed over to the pavement instead.
Test 3	The cyclist did a reasonable choice by just crossing the road. This is as expected. No new problems in this test.
Test 4	The cyclist starts and stays in the cycle lane. This is reasonable, but they could just as well have used the road. They then avoid the pedestrians by using the mixed road. This is also reasonable. In a real world scenario, the pedestrian would probably have moved out of the way, but in this scenario the pedestrian could just as well have been a traffic cone.
Test 5	The cyclist decides to use to mixed road in the intersection. This is a good choice due to the amount of traffic, but could be different if there was more traffic. Then the cyclist crosses diagonally over the intersection. This is also reasonable in this specific scenario. What would happen if there was traffic coming in the opposite direction and they got a green light at the same time? This indicates that there should be more complex tests and that more complex scenarios could be a problem.
Test 6	The cyclist makes reasonable decisions. They do not cross into the mixed road when there is traffic. When the traffic has passed they cross into the mixed road to reach the goal faster. It also makes sense that the cyclist would be in the mixed road from the start, but this scenario would probably end with the same result.

Table 5-3 - Table with test results for the transport cyclist. Each result is based on the transport research's verifications/meanings, an analysis of the videos and the literature study.

6 Discussion

This chapter will evaluate the result of the project. This evaluation is based on the content presented so far in the thesis, together with the implemented code and the test videos. Section 6.1 will discuss the results and then section 6.2 will conclude the project.

6.1 Project Analysis

The project's goal was to make an autonomous agent which could control a cyclist model in a simulator. The cyclist had to interact with several traffic scenarios in a realistic manner.

The literature study showed that cyclists can behave in many different ways. The AI agent therefore had to simulate different cyclist types. It was decided that three different types would be simulated. Six different traffic scenarios was made to test the different behaviours. The results showed that the autonomous agent can control the cyclist model. It also showed that the different cyclist types behaved has expected in the scenarios. Some issues were pointed out, but these did not affect the results in a negative way. This means that the AI agent can simulate cyclists in the scenarios that was included in the tests. The issue is that these scenarios only test a very small set of all possible traffic scenarios.

Chapter 4 explained some of the limitations that the AI agent has when it must deal with traffic. Chapter 7 will also discuss some other improvements that currently limits the AI agent's abilities. With other words: The implemented solution got several possibilities for improvements. Therefore, one can conclude that the current solution is a prototype of an autonomous cyclist agent. This prototype fulfils the project's main goal to an extent. The question then becomes: How well has the project's research questions been answered?

6.1.1 Analysis of the Research Questions

Chapter 1 presented three research questions in addition to the project's main goal. Evaluating how well these have been answered will help to conclude the project.

Question 1

How can realistic cyclist behaviours be simulated?

- What separates different cyclist's behaviours from each other?
- Can one categorize different cyclists into groups based on behaviours?

The first question asked how one can simulate a realistic cyclist. This has been answered throughout the thesis. Chapter 2 explained how you could divide cyclists into types, what separated the types from each other and how different cyclists might behave. Chapter 4 explained how an AI agent in implemented to autonomously control a cyclist model with the use of behaviour trees. The tests, that was defined in chapter 3, then confirms that the AI agent have managed so simulate believable behaviours for the different cyclist types. The analysis of the tests uncovered some potential improvements to the implementation. These will be discussed in chapter 7 as potential future work.

Question 2

What is a good simulation of cyclist behaviours?

- When is a simulation of cyclist behaviours good enough?
- What defines a good or bad simulation?

- How can one test the quality of a cyclist’s behaviour?

The second question have also been answered throughout the thesis. The question asks about the quality of a cyclist’s behaviours. The thesis discusses the different cyclist personalities and what separates them in chapter 2. A good cyclist behaviour should thus stay true to what is expected of the given personality. Chapter 3 then presents how one can test the quality of the cyclists’ behaviours. The chapter also explains what is meant by a good simulation and how this can be based on subjective meanings. The results were therefore verified by two external experts. This was done by presenting the recorded videos of the results to two transportation researchers which works with cyclists in their research.

Question 2 also indirectly asks if the finished solution provides a good simulation of different cyclists. This has been answered throughout the thesis by presenting how the solution is implemented, how the simulated cyclists have been tested, and by presenting and analysing the results of the tests and the whole project.

Question 3

Which technologies must work together to make an autonomous cyclist agent?

- Which technologies are necessary to control the cyclist?
- Which technologies can be used to simulate different behaviours?
- What degree of connection is needed between the environment and the agent?

Question 3 asks about the technological aspects of the solution. The thesis has presented the different technologies used to make the autonomous cyclist agent. This was explained in chapter 4. The chapter explained which different technologies were needed and how these were connection together. The connectivity to the 3D environment was also explained.

Further, it was mentioned how the cars and pedestrians in the scenarios also could be controlled by the same AI Controller. These were given no personality, but this could be added in the future. This showed some of the flexibility of the system.

Question 3 also implicitly asks how good the technologies work together. Chapter 4 explain how the technologies where connected. This was then tested with the predefined tests from chapter 3. Chapter 5 presented the results of these tests. This showed that the solution works, since the solution produced good test results for all the cyclist types, with some minor issues. Improvements to these issues will be discussed in chapter 7.

6.2 Conclusion

The project has resulted in an autonomous cyclist agent which can simulate three different cyclist types. The cyclist types are transport cyclists, child cyclists and leisure cyclists. Each cyclist participated in a case study where they had to complete six test scenarios. The tests were meant to test their behaviours in different traffic scenarios. The results showed that each cyclist behaved as expected of their personality type. This was then verified by two external researchers with knowledge about transportation and cyclists. They were showed videos of the tests and agreed that the cyclists behaved mostly as expected.

The autonomous AI agent was implemented in an existing simulator with a cyclist model. The AI agent was built up of two components. One was meant to autonomously control the existing cyclist model, while the other component was mean to give the cyclist model different behaviours. The behaviours were chosen by using a behaviour tree. This was a good choice of technology. The behaviour tree made it much clearer to understand what the AI was doing at all times. This was partially due to how behaviour trees are structured and partially due to how Unreal Engine visualize this structure.

After evaluating the project, I conclude that the project's result is a prototype of an autonomous cyclist agent. The project's goal was never to make a fully functional AI agent; It was to answer how one could simulate different cyclist behaviours with an autonomous AI agent. The thesis has presented one way this can be solved. The results show that this solution works, but that there are several ways that it can be improved. These improvements will be presented in the next chapter.

7 Future Work

This chapter will discuss potential improvements to the solution and suggest some research subjects which can be used for future Master Thesis'. Section 7.1 will discuss potential improvements to the project. Then section 7.2, 7.3 and 7.4 will propose some potential research projects that can be based on this project's result.

7.1 Improving the Simulator

This section presents some of the problems with the solution and discusses how to improve them. Some of these problems were discovered during testing, while some are related to the implementation.

7.1.1 Visual Improvements

There are many ways the visuals in the project can be improved. This subsection will present some improvements, which can have a direct impact in the perceived realism of certain scenarios.

Animations

During the tests the external researchers pointed out that they expected the cyclist to do certain things in many situations. For example, the child cyclists should walk next to their bicycle when crossing a road. This is just visual effects that can be solved by animations. The AI agent could set the speed of the cyclist to 5 km/h, which is a normal walking speed. Then one could play an animation of the cyclist walking next to the bicycle. The behaviour tree could easily start this animation, as long as a variable could tell the behaviour tree what state the cyclist should be in. Some other visual improvements the could be solved by animations are: looking around, standing on the ground when the cyclist stops, or use hand signs in traffic. These would all improve the perceived realism of the cyclists, and could be useful when more realism is needed.

Texture Improvements

Improving the textures is another way to improve the realism of certain scenarios. The idea here is to change the textures completely, not just adding details or increasing the resolution. For example, all the cyclists use the same rider model. An easy improvement would be to change the model based on the cyclist type. A child cyclist would then have a smaller bicycle and rider, and the rider should be replaced with a model of a child. These are all relatively easy to do in Unreal Engine 4. You would need a rider model with the same bone structure[1] as the original rider. Then you could switch the textures, while you use the same skeleton. Now you can replace the rider with the new child rider. Since the skeleton is the same, it would work without any more adjustments. The same process could be applied for the bicycle as well. It requires a certain skeleton. If the new bicycle model got this, then it can replace the mountain bike in the simulator with something else.

[1] Bone structure refers to the number of bones and joints in the character model. This is usually decided in the creation process of a 3D model and can be changed by an artist. This model is made in 3D modelling software and is hard to change in Unreal Engine 4's Editor.

3D Environment Improvements
Another visual improvement could be to add more models and details to the environment. This could be buildings, decorative items or whatever would improve the visuals of a scenario. This involves making the 3D models, importing it to Unreal Engine and adding any scripts. For example, a house might have lights which comes on after dark, so a script would be added to enable the lights when needed. Adding new models to the simulator could help with making new scenarios. This could be useful in several of the use-cases for a simulator.

7.1.2 Technical Improvements
This subsection will present some improvements to the implementation. They are meant to improve the results of the tests or make it possible to make more complex scenarios.

Road Directions
The transport specialists both agreed that if there is a cyclist lane on both sides of a mixed road, then one should always use the one on the right side of the direction you are traveling. This is not taken into consideration when implementation the solution. An improvement would therefore be to add this. This would not affect the results of the tests in any significant way. For example, the only time this is an issue is at the end of test 5. Implementing this rule would make the transport and leisure cyclist use the pavement for the last section of the test. For other scenarios this could have a bigger impact, and the law says one should not use the oncoming lanes, so this should be improved.

Lane Conditions
The condition of the different lanes would also affect the choice of lane for a cyclist. For example, a bicycle lane might be dirty or wet. If this happens then the cost should be increased for that lane. This would affect the choice of lane for all cyclists. A transport cyclist would use the road, if the cyclist lane was dirty, but would not care what they used if both lanes had the same conditions. This can be implemented either statically or dynamically. One could just add another road type to the environment, or one could implement conditions on the roads. This will be discussed in the next paragraph.

Dynamic Costs
The prototype uses static states which are switched between at runtime. This solution works with the complexity of the current solution, but will be hard to scale. For example, in the current solution the cost of using a certain lane is decided by switching between cost-sets[1]. These are switched based on conditions like: if there is traffic or if the cyclist is in an area with an intersection. This will not be very efficient as soon as one needs more than two sets of costs for each cyclist. Therefore, a good improvement would be to use dynamic lane-costs. This means that each cyclist only got one cost-set, but that it is updated in runtime. This allows for much more complex calculation of the cost of each lane. Several variables could be considered, including the conditions of the lane, the speed of the cyclist and the cyclist's personality.

RoadXML
The 3D models of the road used in this project comes with a XML-file. This contains information about the whole road model. The data is based on the RoadXML project. This is

[1] These cost-sets are NavigationFilter classes that contains two key-value pair for each road type. One value for the cost of entering a lane and another for traveling over one unit of the lane. Each cyclist got two sets in the current implementation. One set for when it is less traffic and one for when it is more traffic.

an open-source project which can be used to describe road models and generate road descriptions. The XML-file contains useful information such as road-type, road-width, road-markings, road-direction, which type of actor a lane is meant for and much more. This data could be used to calculate the cost of using a certain lane, decide what direction is allowed to travel in, or even automatically make finished road models that could be used in the simulator. This could be a project on its own, but could also just serve as a better way to estimate certain values.

Multiple Cyclists

The prototype only allows one cyclist in a scenario. One could add more, but the cyclists would not try to avoid each other. This happens due to the way collision is handled. One could solve this by implementing a system which detects when a dynamic object is traveling toward the cyclist. Then the object, which could be another cyclist or car, would try to communicate with the cyclist and reach a decision about who should move to avoid a collision. This is inspired by the way Johansen and Løvland (2015) makes their cars avoid each other in their work. This can be used for cars, cyclists and pedestrians. Another solution to this problem could be to use Unreal Engines crowd management system. This controls crowds of actors by suggesting paths that does not crash into each other. Combining this system with a dynamic cost system could be a good solution.

More Dynamic Speeds

The cyclists already adjust their speed based on where they are or if there is traffic. An improvement would be to adjust this more often based on their personality. A transport cyclist will probably have higher speed than a child. This could further be used to simulate what would happen in scenarios where several cyclists interact with each other. The solution already got support for this. One needs to change the desired speeds on a personal basis in addition to a situational basis. One could just use the personality key from the behaviour tree, then adjust the speed based on what personality was given.

Different Scenarios

In chapter 6 it was mentioned how the AI agent is limited to the specific scenarios. This is because the AI agent lacks support of many other traffic situations. For example, in this project all road crossings have traffic lights, so there is no system in place for dealing with intersections without this. This introduces the problem of yielding rules, which was discussed in chapter 2. The AI agent would have to detect traffic it must yield to, which again means it must learn all the yielding rules. The same would happen when other types of scenarios are made. New types of roads will introduce new rules which the AI agent would have to learn.

7.2 Traffic Simulator

The first research project consists of improving the simulator to include both pedestrians, cars and other vehicles. This project would need to improve the AI agent to support several cyclists, cars and pedestrians. One could then add support for personalities to these new AIs. This would require further testing and new scenarios, which would test all of these together. This project would be an extension of the current project. It would involve trying to solve many of the mentioned improvements. The use-case for the results would be much of the same as the one mentioned in this thesis.

7.3 Training Simulator

The goal of the project would be to see if a simulator can be used to increase a cyclist’s experience with cycling. This project would involve improving the simulator and AI agent so that it could be used to simulate several different scenarios. One could then try to use the simulator as a training-ground for inexperienced cyclists. The simulator would provide a safe environment for any scenario. Cyclist activity in big cities in Norway must be doubled, as mentioned in the literature study. One possible way to achieve this is by making cyclists more confident in their cycling abilities and to make them feel safer when using a bicycle.

This project could also make use of visual reality as an improvement for the simulator. This could increase the immersion for the cyclist in the scenarios. This would involve using the real bicycle that was part of the simulator used in this project.

7.4 Traffic Planner

This project would involve using the simulator to help with a road construction project. The goal would be to show that one could use simulators to make safer or better traffic solutions. This could involve modelling different solutions for a construction project. One would then use the simulator to test the different designs and discover previously unseen issues.

It could also be useful to find new ways to display different statistics from the simulator. One could record the speed of different cyclists, show the paths cyclists would choose without running the simulation, or record simulations from a first-person and third-person view.

References

2016. *THE NORWEGIAN NATIONAL TRANSPORT PLAN* [Online]. www.ntp.dep.no. Available: http://www.ntp.dep.no/English [Accessed 30.05.16 2016].

ABRAHAMSEN, R. 2016a. *Unreal Engine 4 - Cyclist AI Tests* [Online]. YouTube. Available: https://www.youtube.com/playlist?list=PLyQ2lVssCiGuufULltdKYA5pcY2pwBqiy [Accessed 06.06.16.

ABRAHAMSEN, R. 2016b. *Unreal Engine 4 Bicycle Simulator* [Online]. Github. Available: https://github.com/runeabrahams1/Unreal-Engine-4-Bicycle-Simulator-with-AI [Accessed 06.06.16.

CHAMPANDARD, A. J. 2007a. The Gist of Hierarchical FSM. Available: http://aigamedev.com/open/article/hfsm-gist/ [Accessed 07.06.16].

CHAMPANDARD, A. J. 2007b. On Finite State Machines and Reusability. Available: http://aigamedev.com/open/article/fsm-reusable/ [Accessed 07.06.16].

CHAMPANDARD, A. J. 2008. *Behavior Trees for Next-Gen Game AI* [Online]. aigamedev.com. Available: http://aigamedev.com/insider/presentations/behavior-trees/ [Accessed 30.05.16 2016].

CHAMPANDARD, A. J. 2012. *Understanding the Second-Generation of Behavior Trees* [Online]. aigamedev.com. Available: http://aigamedev.com/insider/tutorial/second-generation-bt/ [Accessed 30.05.16 2016].

CHAMPANDARD, A. J. 2007c. Understanding Behavior Trees. Available: http://aigamedev.com/open/article/bt-overview/#StatesorBehaviors [Accessed 07.06.16].

ESPELAND, M. & AMUNDSEN, K. S. 2012. National cycling strategy 2014-2023.

FLOREZ-PUGA, G., ´ıN, M. G. O.-M., ´ıAZ-AGUDO, B. D. & GONZALEZ-CALERO, P. A. 2008. Dynamic Expansion of Behaviour Trees. aaai.org.

GAMES, E. 2016. *Behavior Trees in Unreal Engine 4* [Online]. Epic Games. Available: https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/index.html [Accessed 30.05.16 2016].

HERFINDAL, T. K. 2015. *Fra A til B.* MSc, Norwegian University of Science and Technology.

JI, L. & MA, J. 2014. Behavior tree for complex computer game AI behavior. *Simulation and Modelling Methodologies, Technologies and Applications,* 60**,** 201.

JOHANSEN, S. I. & LØVLAND, A. S. 2015. *Flocking for Road Traffic Efficiency Improvement.* Norwegian University of Science and Technology.

KREKLING, A., SCHAU, V., NÆRUM, A. & HATLESTAD, R. 2014. Analysis of bicycle accidents. *In:* 294, N. (ed.) *NPRA reports.* Norwegian Public Roads Administration.

NPRA 2014. Sykkelhåndboka. http://www.vegvesen.no/_attachment/69912: NPRA.

NPRA. 2016. *Trafikkregler for syklister* [Online]. NPRA. Available: http://www.vegvesen.no/trafikkinformasjon/Syklist/Trafikkregler [Accessed 08.02.16 2016].

SYKLISTENE.NO 2016. Syklopedia. *Syklopedia.* syklistene.no.

Appendix A

This appendix contains the questions asked to the transportation experts. The questions are meant to get feedback on the quality of the tests in the case study. The specialists were showed the videos of the tests and asked the question either during or after the presentation. The questions are translated to English, but was originally in Norwegian.

Test Questions (X is replaced with each cyclist type):

Question 1 - How good did cyclist X do in Test 1
Question 2 - How good did cyclist X do in Test 2
Question 3 - How good did cyclist X do in Test 3
Question 4 - How good did cyclist X do in Test 4
Question 5 - How good did cyclist X do in Test 5
Question 6 - How good did cyclist X do in Test 6

General Questions (X is replaced with each cyclist type):

Question 7 – What is your overall opinion of how the cyclists behaved?

Question 8 – How was the route choice for cyclist X?

Question 9 - How well did the cyclists stay to their personality?

Question 10 – What could be improved to make the cyclists behave more realistically?

Question 11 – Do you have another other remakes about the cyclists?

Question 12 - Do you have any other remarks about the solution/prototype?

Appendix B

This appendix will give a short explanation on how to install and use to solution.

Installing the solution

1. Download Unreal Engine from www.unrealengine.com.
2. Install Unreal Engine with version 4.11
3. Get a copy of the project's code. Either from the included zip file or from GitHub.
4. Launch the BikeV3.uproject file. This will compile the project and open it in Unreal Engine.

Using the solution

When the project opens it will be set up to run a demo of a traffic scenario. The scenario includes a cyclist model, which will be controlled by a human player, and some other actors, which will be controlled by AI agents. To run the scenario in the editor:

1. Run the project by using "Play". This will play the game in the editor.
2. Control the cyclist by using "WASD" to steer and "Q" or "E" to change gears. The mouse can be used to look around. "C" will change the camera to first person.
3. Click "T" to run the AI cyclist, "Y" to run the AI cars and "U" to run the AI pedestrians.

The cars and pedestrians have one option. The option is where they should go when the AI is running. To change this:

1. Click on the actor model that needs its goal changed.
2. Locate the option menu for the selected actor (a menu on the right side by default)
3. Set the goal in the "Goal" variable. A goal is a unique actor type that can be placed and moved in the 3D environment.

The cyclist has two options. These are: where they should go when the AI is running and what personality they have. The goal is modified in the same way as for the cars or pedestrians. To modify the personality one can:

1. Click on the cyclist model that needs its personality changed.
2. Locate the option menu for the selected actor (a menu on the right side by default)
3. Select the personality from the dropdown list for the "Personality" variable.