# Pedestrians in Driving Simulators

Joey Deiman
Utrecht University
May 9th 2015

## *Section 1: Introduction*

Green Dino BV is a company located in Wageningen that largely focuses on driving simulators. They have a longstanding driving simulation project, currently on version 18. It has been distributed across the globe, from Australia to Sweden. A physical simulator can be seen in Figure 1. The current simulator is an ongoing project, and as such has its fair share of less-than-ideal code structure and documentation, which can significantly slow down development. Furthermore, the simulator is only available for Windows. Mainly because of the latter, the company is looking to start a new simulator project.

**Figure 1: A physical Green Dino simulator.**

For this new project, they have chosen Unity. One of the main advantages for Green Dino is that Unity provides platform independence. Especially the webplayer is attractive, as this allows for a far more widespread audience. And with the release of Unity 5 an option to build a WebGL version became possible, which means people don't even have to install the Unity webplayer anymore. Green Dino currently already has a "browser" version of the software, but this is actually just the simulator packed into a giant executable. This executable has to be downloaded and installed, and afterwards you can start it through the browser via a magnet link. However, this really is just a fancy way of installing the simulator and running it for dummies, as the browser only serves as a shortcut to the executable. This hassle for both Green Dino and the consumer will not be there in the Unity project, since no extra effort is required to do the web build, and no extra effort is required to install the software as a consumer.

A second advantage of switching is the sheer ease at which you can develop in Unity. Many helpful debug features are readily available and integrated into the work flow. A lot of common functionality, such as creating automatically updated objects in the world and integrated physics is taken care of behind the scenes, with a very comfortable interface to your own code. Rendering of objects, and changing objects on-the-fly is much easier than Green Dino's current setup. And last but not least: the editor GUI. Green Dino develops purely in C++ code, with outdated editors in separate executables to create scenarios. This will all be consolidated in Unity, as you have a multi-functional customisable 3D editor at your disposal. In short, everything will be easier and faster.

Currently, the Green Dino simulator only has highly scripted pedestrians, or even static pedestrians that do not move at all. The scripted pedestrians might be good enough to facilitate specific scenarios, but since the rest of the pedestrians are static, the world feels a bit empty. Adding properly simulated pedestrians can help alleviate these issues, and bring some more fidelity to the world. Furthermore, scenarios where a student has to interact with large amounts of pedestrians can be easily created by spawning many pedestrians in the crowd simulation, instead of individually scripting each one. Realistic pedestrians might also increase learning speed, but that is not the focus of this project. For the realistic pedestrians, a crowd simulation framework developed at Utrecht University will be used.

In summary, my assignment was to show that creating a prototype driving simulator in Unity using some Unity Asset(s) as a basis is relatively easy. Then, the UU-Crowd Sim has to be incorporated. Finally, I went on to do research on pedestrian-vehicle interaction. There were two parties who might commission a Unity simulator during my internship, those being Team Alert for a scooter simulator and Connexxion for a bus simulator. Should one of these parties enter negotiations regarding a simulator during my project, then the focus shifts towards that party.

The rest of this report will be structured as followed. Section 2 discusses the ups and downs of building a prototype driving simulator in Unity. Section 3 explains how the UU-Crowd Sim was incorporated into the prototype. Section 4 presents several aspects of

pedestrian-vehicle interaction, and which of those aspects I have implemented. Section 5 will conclude the report with a discussion and recommendations for future work.

## *Section 2: Building a Prototype Driving Simulator in Unity*

The starting point of the research was a week of Unity Asset research. A Unity Asset is basically a miniature software package, developed for use with Unity. The project would then be kick started by using a car physics package and a traffic simulation asset. Green Dino did some (very minor) research already, and came up with the car physics package *CarX* and the traffic package *TrafX* respectively. These packages by the same developer are very expensive, going upwards of €20,000+ for the most expensive *CarX* version. I therefore started looking at alternatives, and found a seemingly suitable alternative for both *CarX* and *TrafX* in the form of *UnityCar* and *Intelligent Traffic System* (*iTS*) respectively. These packages were 28-70 times cheaper than the *X* counterparts, with seemingly the same or better functionality. When comparing the web demos for the car physics packages side-by-side, they may have just as well been the same package. The only differences were a couple of non-important features that one lacks and the other has, or vice-versa. Comparing the traffic systems was not as easy, as only *iTS* had a proper web demo while *TrafX* only had a very limited amount of video footage. From what I could gather, the packages were nearly identical in functionality. It was clear that *iTS* blatantly copied the editor interface for placing roads from *TrafX*, as even the colors of the splines were the same. Appendix A shows comparison tables for the physics and traffic packages, with the most notable features. It clearly shows how well the packages stack up to each other.

In the end, the decision on which packages to use was simplified by the fact that the *X* people appear to have ceased sales of their physics and traffic packages. This was likely done in favor of development on their mobile game (which started as a demo for the physics engine). They seem to have not responded for about 2 years, judging from forum posts in their *CarX* thread. We therefore decided to go with the *iTS* traffic package (Figure 2). *UnityCar* was not purchased, as *iTS* appeared to incorporate good enough car physics. How well *iTS* managed to deliver on their demoed functionality will be discussed later in this section.

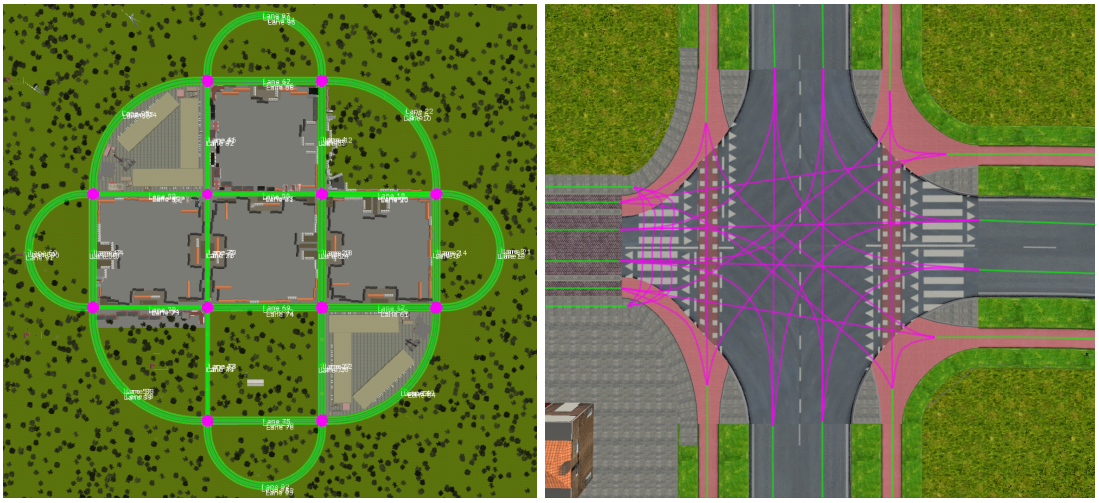


**Figure 2: iTS roads (left) and a specific junction (right)**

I then went on with building the prototype. The very first thing I did was building a data structure to allow for player location in the road net. Not only is this useful during development to test the road net, but it is also a vital component to be able to judge students' driving behaviour. Since the road net consists of a bunch of line segments, I decided to use a kd-tree on the endpoints of these line segments. As the road net is static during the simulation, this allows for fairly efficient location queries. The endpoints hold a reference to their line segment, in order to check if the query point is on a line segment in a tree search. I implemented my own kd-tree, mainly for educational purposes since I had never implemented a proper spatial data structure before and also because this allows for easier customisation than when using a library. If my implementation turns out to be too slow, a library like Boost can always be used later.

During the above stage, Team Alert started negotiations for a scooter simulator, so the focus of the prototype shifted into that direction. I therefore started to create a scooter rather than a car, since the prototype would have to work with scooters. After many different attempts, I could not get a proper 2-wheeled scooter to work with Unity's built-in wheel objects. It should be possible, but the scooter kept bouncing, making wheelies or sliding. In the end I faked it by using 4 invisible wheels and making the scooter lean purely visually through a script, as the scooter started to take too much time. Some extra faking was done to allow for sharper turns, in the form of torque applied to the front wheels when turning. Finally there is the endless tweaking cycle of the wheels' parameters as well as the stabilizers to prevent the vehicle from flipping or wiggling. This is something that I have not done much of after the prototype was fairly stable because it would take too much time for non-crucial payoff. Figure 3 shows the fake wheels.

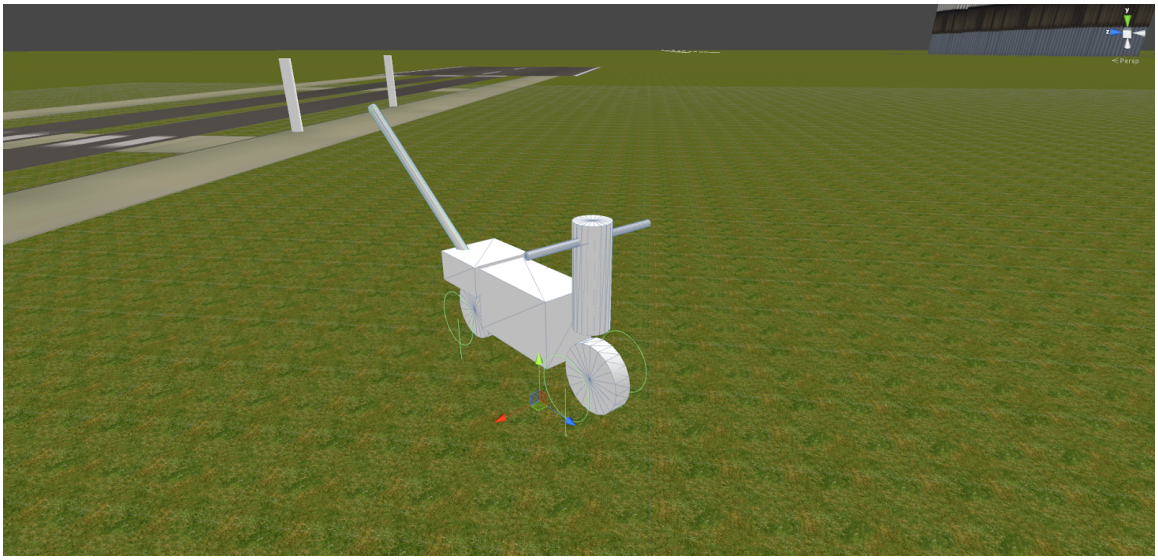


**Figure 3: Fake wheels (green circles) on a scooter.**

With a proper player vehicle and a way to locate him, I could implement detection of a few traffic violations. In particular: Speeding, driving on the wrong side of the road or off-road and crossing a red light.

As development went on, I was forced to dive into the *iTS* code more and more. The code of this package is very bad. There is no consistency whatsoever, not in the use of Camel Case or some other case, not in the use of white space, not in the use of naming things. The code is littered with poor English, vague names for variables and illogical class hierarchies. As such, I went on to rewrite more and more of the code as time went on, as well as general cleanup to make it easier to read. Thijs de Goeij, another GMT student, used his get-acquainted-with-Unity time by following vehicle tutorials, so credit for the physics rewrite go to him. In the end, the only things that remained were the road net editor and the road net that it produces.

## *Section 3: Incorporating the UU-Crowd Simulation*

When the development of the prototype had a working vehicle and traffic system, I also started to incorporate the UU-Crowd Simulation framework [11] (called UUCS from now on). The first thing to do was interface with the UUCS .dll file. This was done by simply mirroring all the API functions in C#. A bit of research was required to obtain the array of CharacterStepData (data for a character produced by a simulation step) objects, but I got it figured out (turned out that it kept failing because of an unrelated issue). After the whole UUCS initialisation process returned all ‘true’, I had to figure out a way to export a mesh in a Unity scene to a .env file. My first idea was to use line obstacles along the edges of the road, but due to errors in the mesh this did not work so well. After I was informed that there were also WalkableAreas, I simply used the triangles of the meshes in the form of walkable areas. These triangles are projected onto the ground plane, because for now there are no bridges or the like so layers are not necessary. Care had to be taken

when converting the triangles, as transformations of the object in Unity turned out to cause a mismatch between the recorded .env file and the actual visible mesh in Unity. After including the transformations of the GameObject containing the mesh, the .env file nicely fit the actual mesh in the scene. This did mean that a .env file for a mesh in a scene will not work for the same mesh in a different scene, if it has different rotation/scaling or if the pedestrians use a different parent transformation. That's why I decided to have everything related to the UUCS as children of the same GameObject, which allows me to transform the parent object, while the entire simulation transforms along with it. This works because all children are transformed according to the parent transformation. This also allows for multi-mesh support, as you can just have more child GameObjects under the parent, which will all have their triangles properly placed in the .env file alongside each other. Figure 4 shows the result of the .env generation.

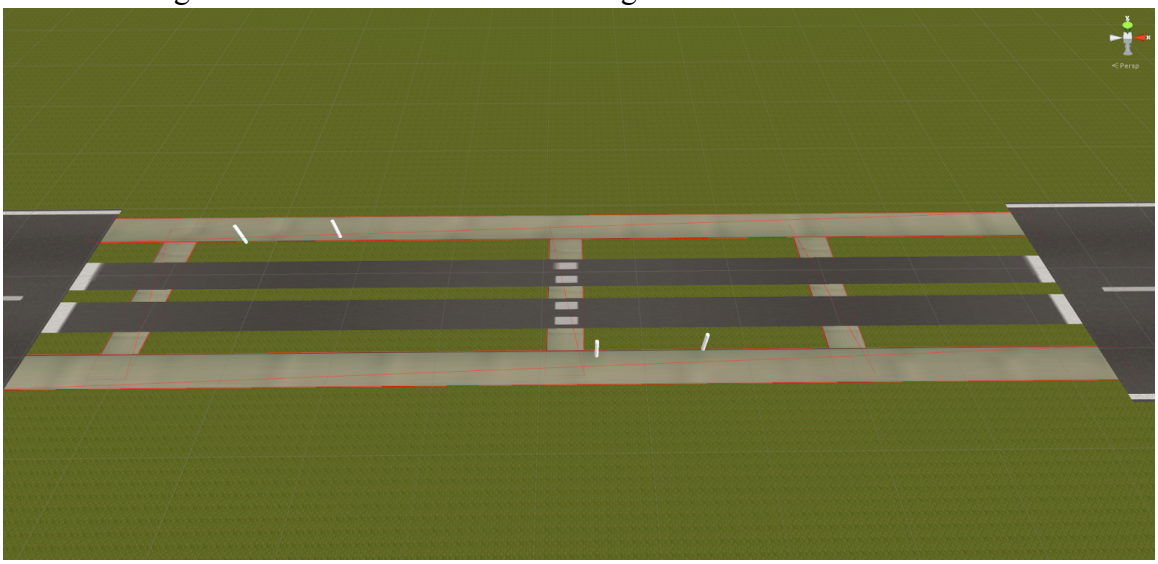


**Figure 4: .env generation (the red lines show the walkable areas).**

When spawning pedestrians, I have chosen for the simple approach of manually creating sources and sinks in the form of empty GameObjects, as finding a suitable spot randomly in the environment would be too much effort for a prototype. Since walkable areas are used, placing a source or sink outside of the walkable area will result in the inability of the character to move. When we successfully spawn a pedestrian, a new GameObject is created, which holds the id of the corresponding ECM character. This GameObject holds all the Unity-side data, such as the mesh and the collider. The GameObject will receive update events as simulation steps are done in the ECM framework. It will start moving towards the position it gets from the CharacterStepData struct. The character in Unity is able to react to situations in the world in one of two ways. The first is to simply change the goal in the ECM framework. The character will then just walk somewhere else at his normal pace. It can also be used to pause a character, by setting its goal to its current position. The second option is to completely delete the character from the ECM framework. We can then control the character fully in Unity. Once the event is over, we can put it back into the framework and have it continue towards its original goal. We do

have to take care that the character does not go out of bounds during this period, as re-inserting it later will then cause it to fail its path planning.

About 3 weeks before the end of the project, Unity 5 was released. It is a major improvement over Unity 4, and contains some specific enhancements that benefit the driving simulator, such as improved wheels and the ability to build to WebGL. After purchasing and installing Unity 5, the project was upgraded automatically without any manual intervention. However, when I ran the simulator, there was no crowd simulation. As it turns out, Unity went to 64-bits, while the crowd simulation .dlls and dependencies were all 32-bit. I did not see a 64-bit build option in the crowd simulation VS solution. We therefore decided to roll back to Unity 4 until the end of my project, after which the upgrade can occur, be it without crowd simulation. A 32-bit Unity 5 is available but I did not want to risk more issues when going from 32 to 64 later. After my project, the crowd simulation will not function until a 64-bit Windows version is made available. This version can however be easily built.

# *Section 4: Modeling Pedestrian-Vehicle Interaction*

## *Section 4.1: Introduction*

The field of pedestrian-vehicle interaction has seen a fair amount of research [4]. Insight in behaviour of pedestrians and vehicles can help in designing safer and/or more efficient roads, but they also allow for more realistic modeling of pedestrians in software. Nearly all interaction takes place when a pedestrian has the possibility of crossing the street. As such, nearly all research treats this area.

The research on interaction appears to target either the more statistical area [1][2][3], where the number and types of actions of pedestrians and vehicles are measured, or finding a proper behavioural model for the entire interaction [10]. I decided to do most of my research on the first aspect. The reason behind this is that the actions resulting from human decisions are more apparent in a simulator than the entire process leading up to that decision. The behavioural aspects are still important, mainly for the more subtle interactions like body language, but I did not find as much research on the matter, and their effect in a simulator is likely less noticeable. I did find a large study on the subject, which specifies a sequence of actions that happens around a conflict. This particular model seems very useful in the simulator due to its simplicity. In a later development stage, more study on the behavioural aspects is definitely beneficial. Especially if we can track every move the student makes and possibly have the simulated pedestrians react to that.

Section 4.2 considers the statistic related research and section 4.3 considers the Behavioral Sequence Model, which is an instance of a model that aims to model the entire course of interaction. The BSM is also the model that I have chosen to implement partially.

## *Section 4.2: Action Statistics Related Works*

The first thing to note is that nearly all papers naturally focus on some specific scenario, or small amount of scenarios [1][2][3]. Generally the number does not go in the double digits. Therefore, most individual research will give insight in for instance only signalized crossings [4], only in narrow urban environments [1], or only on a road where not a lot of vehicles pass [3]. Some only consider conflicts, which are situations where the path of pedestrian and driver would lead to collision if neither takes evasive actions while other also consider non-conflicts. Finally, there is a cultural aspect to consider, as observations in Japan may not be the same as observations in the Netherlands on an identical crossing. Even the culture in the same country can differ [3]. As such, I have decided to first discuss each scenario in a sub-section, which will only encompass a single paper per sub-section. I tried to pick ones that bring something useful to the table for the driving simulator. Note that the list of situations is not exhaustive in any way and I don't have the time to go over every paper on the subject, so many scenarios will not be discussed. After the specific scenarios, I will discuss some general action related research. Afterwards, I will discuss the usefulness of the research for driving simulators.

### *Section 4.2.1: Narrow Urban Environments in Japan*

Young-in et al. [1] did research in Japan, at 25 different locations in Tokyo. The width of streets in urban Japan is smaller on average than in the West. Sidewalks are also much smaller, and sometimes not even present at all. The observed locations all did have some kind of sidewalk, with different separation to the street (lines, bollards etc.).

The authors came up with a formula which takes the longitudinal and lateral distance between the 2 participants of a conflict and their relative speed, and calculates the probability that one of the participants takes evasive maneuvers. The formula is shown below. It is worth noting that they use both km/h and meters in the same formula, which is a little bit sloppy, and should have been converted to a consistent unit (as they are now probably compensated for by the coefficients)

$$\text{Probability of Conflict Avoidance} = 1 / (1 + \exp(-B_n))$$
$$B_n = C_{n0} + C_{n1}(X/S_R) + C_{n2}(Y/S_R)$$

$C_{ni}$ is a coefficient
X is the longitudinal distance between traffic modes in meters
Y is the lateral distance between traffic modes in meters
$S_R$ is the relative speed in Km/h

**Formula 1: Probability of Conflict Avoidance**

For the coefficients, they presented a table with values that work for a pair of traffic modes (combination of pedestrian and either another pedestrian, bicycle or car). They claim that the model is feasible for the observed locations. It is possible that the formula also works in other situations, which would prove useful in a simulation. Agents can

simply poll the cheap formula every so often, or maybe even every frame, and see if they would like to respond to another traffic participant or not.

### *Section 4.2.2: Unsignalized Mid-Block crossings in India*

Kadali et al. [2] did research on pedestrian crossing behaviour on a busy unsignalized mid-block crossing in India. The street has 2 lanes for each direction. Since this is an unsignalized crossing with no right-of-way for pedestrians, they focused on the gap acceptance of crossing pedestrians. A gap in this context means the length of time between 2 consecutive vehicles. Gap acceptance refers to how small of a gap people are willing to accept to attempt a crossing maneuver. Then there is the critical gap, which defines the average gap at which 50% of the pedestrians is willing to cross.

The authors tracked a decent number of pedestrian characteristics, including but not limited to age, gender, group size, statistics on how many times the pedestrian checked for gaps and whether or not the pedestrian rolls over gaps. This rolling behaviour means that the pedestrian moves to the left or right while crossing to combine 2 gaps of adjacent lanes more efficiently, as shown by path A in Figure 5. This is the only paper in which I came across this gap rolling phenomenon so it might be a cultural aspect, but it seems like an important action in dense urban traffic.

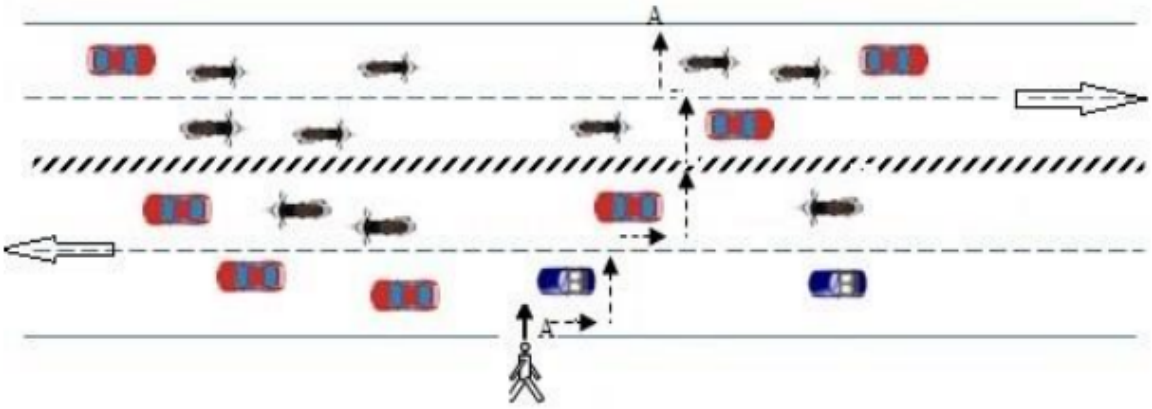



**Figure 5: Gap rolling.**

The authors found that the most important factors that influence the crossing behaviour of pedestrians are driving yielding behaviour, usage of rolling gaps, and vehicle speed. Usage of rolling gaps allows pedestrians to accept a gap of 2 seconds less than when not rolling. For the driving simulator, this gap rolling is not of much use in the general scenarios, since dual lanes with unsignalized crossings will not occur as often in simple scenarios. However, dual lanes do occur more often in real life, especially in dense urban environments like New York. The authors also noted that developing countries have many of these types of crossings. Therefore, for more real-life scenarios, this behaviour is useful in the simulator to prepare students for this complex situation.

### *Section 4.2.3: Urban Crossings in Small/Large Cities in Finland*

Himanen et al. [3] did observations in two cities in Finland. One was Helsinki (500,000 inhabitants) which is a fairly large city. The other is Salo (20,000 inhabitants), a much smaller city. This discrepancy in city size gives a nice comparison on how the locality of the crossing affects behaviour. The authors used a Multinomial Logit model, which is a model that allows someone to choose between behavioural alternatives. I did not look deeper into that model, as the model is too complex and would require too much time to research for what its worth. The results are promising though, so perhaps the model can be looked into further later, when the simulator has matured.

The authors found that most parameters affecting the decision to take evasive maneuvers are influential in both the small and large city. The most important parameters appears to be the distance that the pedestrian might have already crossed at the time of decision making. The further a pedestrians is from the kerb, the less likely it is for the pedestrian to take action and the more likely it is for the vehicle to take action. The speed of the car also has a large impact on the decision making, with higher speeds giving an increased chance for pedestrian action and a decreased chance for vehicle action. The number of pedestrians or vehicles in the group also had an effect (with a larger group of vehicles making pedestrians walk slower and vice versa), but less high. Finally, the locality had a major impact regardless of the other parameters. Vehicles are more likely to take evasive maneuvers in Salo than in Helsinki. Pedestrians are more likely to continue walking in Salo than in Helsinki. Appendix B show the probability curves for multiple situations for vehicle reaction and pedestrian non-reaction.

Besides the parameters that appear to affect the decision making, they also found a number of parameters that surprisingly did not seem to have an impact. The priority situation for vehicles, street width, existence of a refuge half-way on the crossing and many of the observed pedestrian parameters all had little impact. The authors attribute this to correlation between some variables, which make other variables look less significant in the Multinomial Logit model, as well as a possibly too small sample size of 799. This sample size problem already affected the research a bit in the first place, as they had to cluster some situations because their low observation count caused instability in the model.

The results from this paper are quite useful in the simulator. Their model can be used to determine when to start evasive action, much like the formula found by Young-in et al. [1] from section 4.2.1. However, the Multinomial Logit model incorporates far more parameters, and might therefore work reliably in different situations.

### *Section 4.2.4: General Influences on Actions*

Lord et al. [4] discuss a number of general remarks on (traffic related) actions taken by humans. I will briefly discuss the ones that may be useful for our simulation.

Katz et al. [5] and Thompson et al. [6] discuss vehicle speed behaviour when pedestrians are near. I can't speak for the validity of [6] since I did not have access to the paper, but

Lord et al. mention that Thompson et al. found some contradicting results with respect to Katz et al. I did have access to [5], so I will elaborate on this paper. Katz et al. did an experiment where a number of pedestrians approached a crossing and crossed. They measured how much a vehicle slowed down in different situations. Most notably for the simulator, it appeared that drivers slow down more when there are more pedestrians in range and when there is a marked crossing (zebra). Also worth noting is that drivers slow down more when people do not look at the vehicle, which is a little bit of body language research that may be of use. Thompson et al. did their research in England rather than Israel, and found that drivers did not slow down if pedestrians were approaching, according to Lord et al. This is another example of the locality/culture significantly influencing behaviour.

Bartz [7] did experiments to test the effect of the complexity of the main task on the reaction time to peripheral tasks. He found that a low complexity main task would result in worse reaction to peripheral tasks. This shows that doing a mundane tasks lowers a person's attention to other tasks as well. Bartz further theorizes that given previous research, the reaction time to peripheral tasks should worsen again as the main task gets more complex. He mentions that it is possible that the two effects are working against each other, and at the extremes one becomes much more prevalent than the other. For our simulation, this behavioural aspect of detecting things in the peripheral vision may not be the most practical for implementation's sake. However, if we would want very realistic agents, then this is definitely a future possibility. The main concern here is the performance, as doing constant precise peripheral checks for all agents might not be feasible in an online situation.

Hancock et al. [8] and Harmz [9] did studies on the mental load when going straight, left or right at a crossing when driving. I did not have permission to access [9], so I will not elaborate on that one. Lord et al. mention that it did have the same results as [8]. In the work of Hancock et al., experiments were done to find out how much workload driving straight and turning entail. The experiment was conducted by having someone drive around and complete tasks, while monitoring things like reaction time. They found that turning causes a significantly higher mental load than going straight. furthermore, left turning has a higher load than right turning. As with the research of Bartz, this particular research is not of vital importance in our current driving simulator. It may be worth exploring later however, possibly when tracking the student.

### *Section 4.2.5: Discussion of Action Statistics Related Works*

It is clear that there is a very large amount of different situations where pedestrians might cross the street. In theory, any pedestrian could decide to cross anywhere, even if it is not a clear crossing. There are many different types of proper crossings, some with traffic lights and some without. Different sidewalks and streets exist, with different markings and maybe even multiple lanes. Behavioural aspects play a role when crossing, which might be done different for each person. Culture and locality of the crossing can have a major influence on the interaction. And to make matters even more complex, research from different authors in different areas observes different variables and uses different

models to describe their situation. Because of these differences, any form of implementation that would want to capture all nuances would be very complex, is such a model is even possible. Not to mention the amount of parameters that have to be tweaked for every new situation.

For the purposes of an educational driving simulator, implementing this complexity is not worth the time investment. Most students will probably not notice, and even if they do, the added educational benefit might be very low. However, as mentioned before, specialized circumstances might prove useful in teaching students to drive in foreign countries, where many of the aspects involved with crossing pedestrians are different. However, the current Green Dino simulators are also used for research projects. With a more complex pedestrian-vehicle model, the simulators could also be used in social studies, and even studies on future road projects to test pedestrian safety. Many of the research papers already use increasing safety as a motivation. Going this route does mean that the driving simulator becomes more of a traffic simulator, with a student vehicle driving around just to test the system, or to research more deviating driver behaviour.

## *Section 4.3: Behavioral Sequence Model*

Snyder et al. [10] did a large study on general urban pedestrian safety. Amongst the subjects, they created something they call the "Behavioural Sequence Model" (BSM), which consists of steps taken by pedestrians and vehicles when a conflict occurs. The sequence of actions is: Search, Detection, Evaluation, Decision, Human Action, Vehicle Action. Note that the human action consists of the actual physical action of any human, so both pedestrian and driver. Vehicle action is the actual physical response of the vehicle to the human action of the driver, for instance having its steering wheel turned by a human which then turns the wheels. Appendix C contains the BSM in a flow chart, showing what the actions entail and what happens if they succeed or fail. As you can see, failure to execute any one action will mean that the following actions will not be executed, which in turn means no collision avoidance from the corresponding participant. If both pedestrian and driver/vehicle fail any action at any point in the sequence, there will be a collision. Failure can occur in two ways: execution failure or timed failure. Execution failure means that the action was simply not executed properly, or not at all. Timed failure means that the action has been executed properly, but too late.

The study continues with observation data from many sources (research, police departments etc.), from which factors that influence the actions in the BSM were extracted. Inatention can cause a failure in the detection state. Overload means that there are too many inputs for a person to pay attention to all of them at once, causing search or detection failures. Distraction can also cause detection failure, since a person can be focused on something that is not the immediate threat. Pedestrians running down the street can cause detection failures, as the drive may not have enough time to detect on such short notice. Obstacles in the environment can cause issues, as they hinder detection, even if thorough search is performed. Crossing a red light can cause the other party to fail the evaluation, as they had expected you to stop. It can also occur that both pedestrian

and driver execute all steps in the BSM correctly and timely, but take evasive maneuvers that cancel each other out.

The BSM is the model that I have chosen to start implementing into the prototype. The reason I did not go for a more elaborate system for implementing specific scenarios is the complexity reason described in section 4.2.5. The BSM is still a possible basis for the more complex situations, as its implementation will contain basic functionality such as pedestrians and vehicles observing each other, and being able to react in some way or another.

## *Section 4.4: Implementing Pedestrian-Vehicle Interaction*

Before I could implement the BSM model, I first had to prepare the current software. This brought to light a number of issues, most notably a lack of a clear function to bring the vehicle to a standstill at a certain position, as well as right of way related issues. Since fixing the standing issues would be a difficult task, I decided to rewrite the traffic AI entirely, which would also ease bug fixing and the implementation of future functionality.

The first step of implementing the human aspects of pedestrian-vehicle interaction is to create pedestrians and vehicles that strictly follow the rules and have a sixth sense to detect each other. This would also produce the base setup for crossings to use in the further implementation. I decided on using the Unity physics system with its triggers as my main mechanism for detecting future collisions, and detecting if someone is on the crossing or not. The reason behind this is that it is fast, effective and robust, and I had limited time. More elegant ways of doing it are most likely possible, but I will not indulge in those as they would take more time to properly design. A crossing class keeps track of who/what is on the crossing, and blocks traffic and pedestrians from crossing accordingly. It is aided by observer objects, which are triggers that keep track of incoming pedestrians and traffic. The crossing can either give priority to pedestrians or vehicles, causing the non-priority group to yield until no priority members are in the observers. If a non-priority group member is already on the actual crossing, the crossing gets blocked for priority members, as to avoid collisions. If a pedestrian or vehicle is already on the crossing, it will ignore the blocking signals of the crossing since we don’t want traffic or pedestrians stopping the middle of the crossing as this creates a deadlock (or collision) very quickly. Figure 6 show three crossings with their triggers.

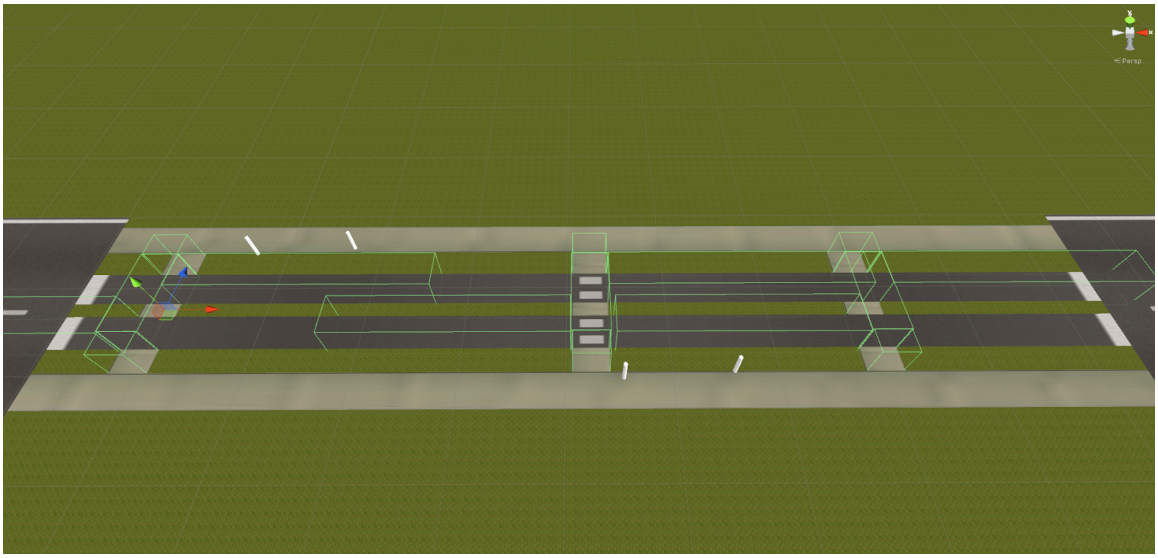


**Figure 6: Three crossings and their triggers.**

As noted in section 4.3, I have chosen to start an implementation of the Behavioural Sequence Model. It is a simple to grasp model, and has clear steps to implement. To find out if a step in the sequence succeeds, I use simple chances per step which are separately assigned per crossing. This allows for easy and intuitive tweaking, but it is not very realistic. Preferably, you would want to do actual visibility checks and proper cognitive calculations to evaluate the situation, but that implementation would have taken too much time. The implemented aspects of the BSM are described below.

Vehicles have no notion of the BSM yet. I focused on the pedestrians as they are easier to control, and I did not have time to possibly tweak the cars for the BSM. The pedestrians are always searching, which is visualized by the pedestrian's head turning from left to right and back. This means that Search will not fail if detection is not necessary. Only when a vehicle has to be detected (when it enters an observer on the crossing) is the Search success evaluated, at the same time as Detection. So when a pedestrian is at a crossing, for every vehicle that enters the crossing or the observers, we first do a Search and Detection check. If successful, the pedestrian will look at the vehicle, otherwise he will keep his search behaviour. If multiple vehicles have been detected, the pedestrian will look at the closest one. For an example of failing Search, imagine someone walking around while staring at their smart phone.

The Search and Detection result is then used when determining if a pedestrian reacts properly to a vehicle. It first checks if the pedestrian would have to take evasive maneuvers (currently only in the form of stopping before the crossing if a vehicle is approaching with right-of-way). We determine if the pedestrian successfully stops at the crossing by doing an Evaluation, Decision and Human Action check. If it fails, the pedestrian will simply ignore that vehicle. Note that the pedestrian can still stop for another vehicle if they successfully pass their BSM checks for that other vehicle. Figure 7 show the result of failing the BSM checks.

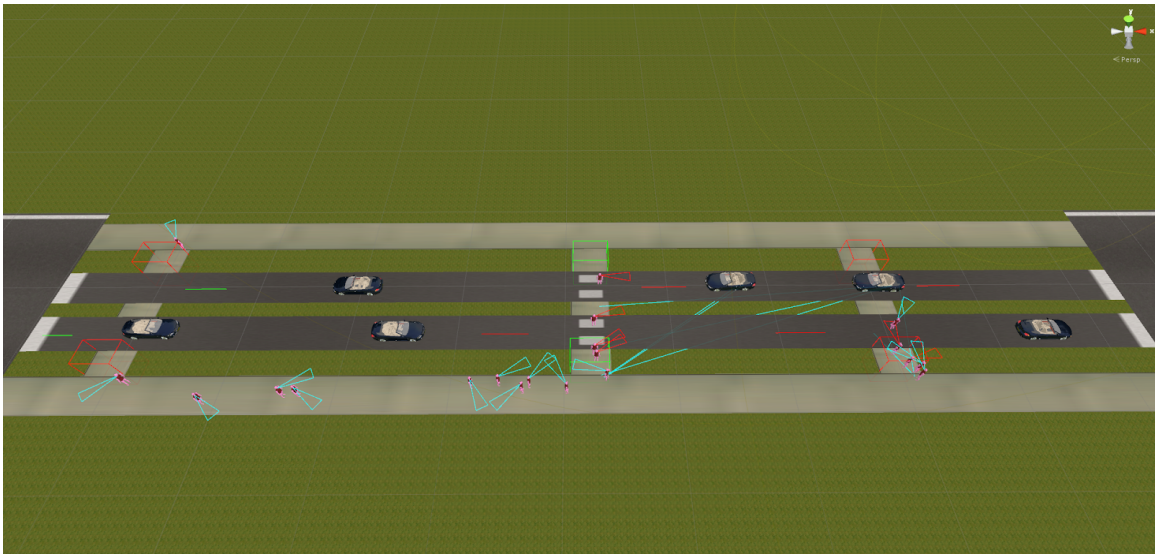


**Figure 7: The pedestrians in action. The center crossing has pedestrian right of way, the other two have vehicle right of way. On the right you can see a pedestrian crossing while he shouldn't because of the BSM.**

## *Section 5: Conclusion & Future Work*

A first prototype of a driving simulator in Unity was implemented. An existing traffic system, called Intelligent Traffic System (iTS), was used to kick start the project. It allows for the creation of logical roads and provides AI to drive on them, as well as a very simple vehicle physics system. More functionality was added to this system, and a large portion of the system has already been rewritten due to the poor code of iTS. The majority of the issues encountered were because of problems in the iTS. Nevertheless, it proved valuable as a kick starter, and the project would likely not have gone as far without it. The UU-Crowd Simulation has been incorporated, allowing for mesh conversion to a .env file and having pedestrians walk on these meshes. Research was done on the interaction between traffic and pedestrians. A full, realistic implementation of this interaction is infeasible for the short term, and likely wasted effort for educational purposes. There are simply too many parameters and scenario's to capture. A simple behavioural model, call the Behavioural Sequence Model has been deemed appropriate for implementation in the prototype. Pedestrians are currently the only ones who follow the model, vehicles still follow the traffic rules strictly. Pedestrians can search for and detect vehicles, and can either take evasive maneuvers in the form of stopping before stepping onto the crossing, or ignore a vehicle and keep walking. The goal to show the relative easy of implementing a driving simulator in Unity using some start Asset has been achieved.

Since this project has only produced a first prototype, there is still a long way to go for Green Dino in order to create a fully-featured driving simulator in Unity. The most important thing to do first is to rewrite the remaining part of iTS: the road network. Much can be used as an example, but the code is too messy and bloated to continue working

with it for much longer. The vehicle physics have to be tweaked further and further, as currently some weird behaviour is possible such as bouncing. The pedestrian-vehicle interaction can be further expanded, either by implementing more features of the BSM, by going for a more scripted route or maybe some more complex system. The crowd simulation framework connection currently only supports single layered environments, and does not take height into account even in a single layer. This can be improved. Many, many more aspects remain, such as possible automated road mesh generation, saving parts of the roadnet for reuse in different scenes, scriptable traffic etc. I think that the current prototype forms a solid basis for this further development.

## References


[1] Kwon, Young-in et al.: **Analysis of behaviours and interaction of pedestrians, bicycles and cars in narrow urban streets**. Joumal of the Eastem Asia Society for Transportation Studies, Vol' 2, No. 3, Autumn, 1997.

[2] Kadali, B Raghuram et al.: **Modelling pedestrian road crossing behaviour under mixed traffic condition**. European Transport \ Trasporti Europei (2013) Issue 55, Paper no. 3, ISSN 1825-3997

[3] Himanen, V. et al.: **An application of logit models in analyzing the behaviour of pedestrians and car drivers on pedestrian crossings**. Accid. Anal. & Prev. Vol. 20, No. 3, pp. 187-197, 1988.

[4] Lord, Dominique et al. **Pedestrian accidents with left-turning traffic at signalized intersections: Characteristics, human factors, and unconsidered issues**. 77th Annual Transportation Research Board Meeting, Washington, DC. 1998.

[5] Katz, A. et al.: **An Experimental Study of Driver and Pedestrian Interaction During the Crossing Conflict**. Human Factors, Vol. 17, No. 5, 1975, pp. 514-527.

[6] Thompson, S.J. et al.: **Driver Behaviour in the Presence of Child and Adult Pedestrians**. Ergonomics, Vol. 28, No. 10, pp. 1469-1474.

[7] Bartz, Alber E.: **Peripheral Detection and Central Task Complexity**. Human Factors: The Journal of the Human Factors and Ergonomics Society February 1976 vol. 18 no. 163-70.

[8] Hancock, P.A. et al.: **Driver Workload During Differing Driving Manoeuvres**. Accident Analysis & Prevention, Vol. 22, No. 3, 1990, pp. 281-290.

[9] Harms, L.: **Variation in Drivers' Cognitive Load: Effects of Driving Through Village Areas and Rural Junctions**. Ergonomics, Vol. 34, No. 2, 1991, pp. 151-160.

[10] Snyder, M. B. et al.: **PEDESTRIAN SAFETY The indentification of precipitating factors and possible countermeasures, Vol 1**. Research for 'Operations Research Incorporated', January 1971.

[11] Jaklin, N. et al.: **Way to go - A framework for multi-level planning in games**. 3rd International Planning in Games Workshop, 2013, pp 11-14.

## *Appendix A: Comparison Tables of Unity Assets*

A question mark denotes that a feature is very likely included but not 100% certain.
A ~ means so-so.
A + means outstanding.

| v ***Feature*** v | *CarX* | *Unity-Car* |
|---|---|---|
| ***Wheels*** | | |
| Models individual wheels | X | X |
| Tire pressure | X | X |
| Tire friction/grip | ? | X |
| Tire deformation | X | X |
| ***Engine*** | | |
| Engine inertia | X | X |
| Engine torque | X | X |
| Engine idle | X | |
| Engine brake torque | X | |
| Fuel consumption | | X |
| Engine limiters | X | X |
| ***Suspension*** | | |
| Spring/damper | X | X |
| Bump | X | X |
| Rebound | X | X |
| Camber | X | X |
| Different Types | X | |
| Wheel steering limits | | X |

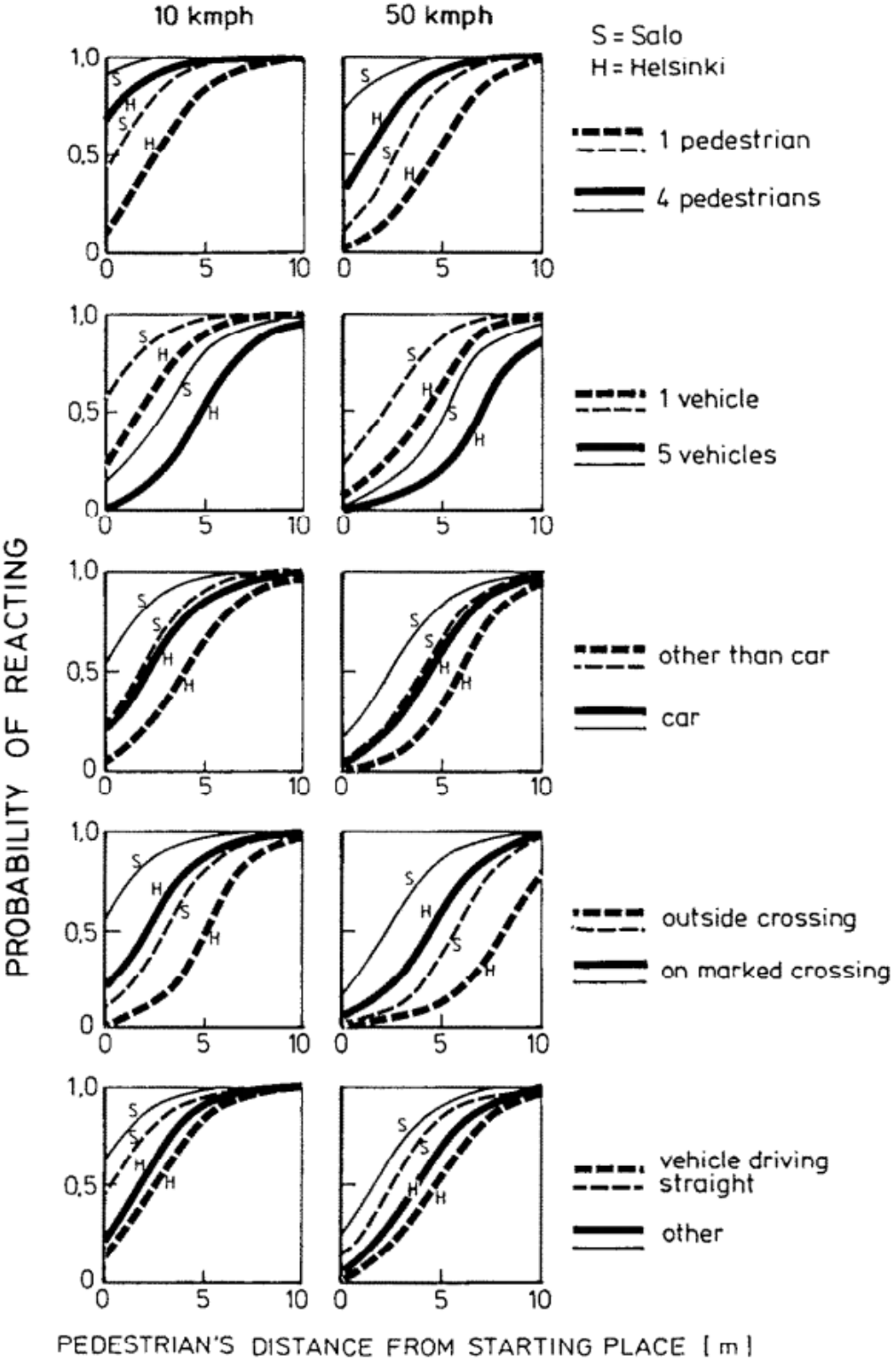
| v ***Feature*** v | *CarX* | *Unity-Car* |
|---|---|---|
| ***Transmission*** | | |
| Differential | X | X |
| Multiple gears | X | X |
| Gear ratios | X | X |
| Final Drive | X | X |
| ***Driving Aides*** | | |
| ABS (Anti Lock Braking) | X | X |
| TCS (Traction Control) | X | X |
| ESP (Stability Control) | | X |
| Steer Assistance | X | X |
| Automatic Transmission/Gear/Reverse | X | X |
| Automatic Clutch | X | X |
| Automatic Reverse | X | X |
| ***Other*** | | |
| Customizable brakes | X | X+ |
| Simpler mode for traffic | X | X |
| Drag | X | X |
| Support | ~ | ~ |

**Appendix A table 1: Car physics packages**

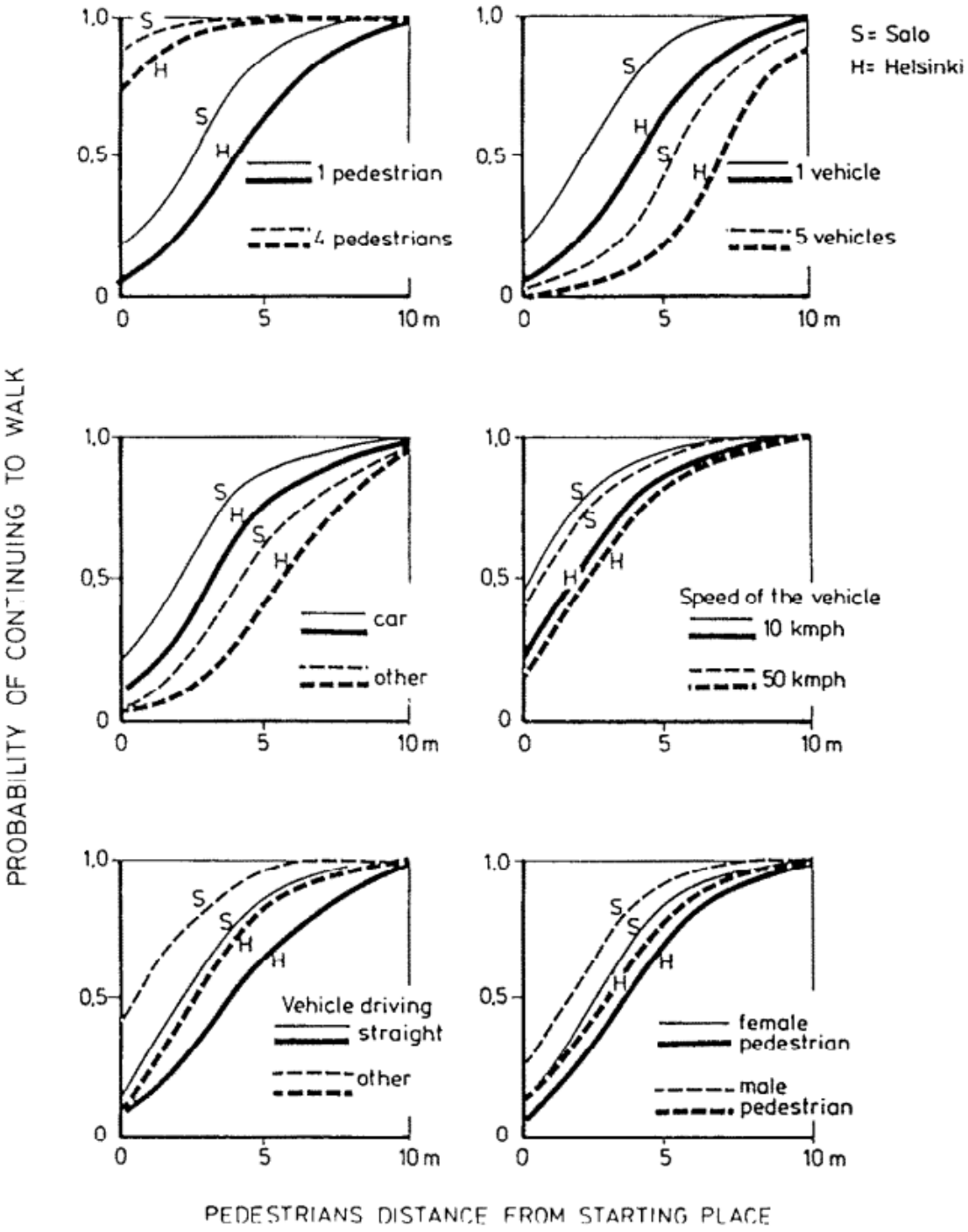
| v ***Feature*** v | *TrafX* | *iTS* |
|---|---|---|
| ***Roads*** | | |
| Different materials (asphalt, dirt etc.) | | |
| Different road user types (car, bus, bike etc.) | X | X |
| Road width | X | X |
| Speed Limit | X | X |
| Can handle round corners | X | X |
| Can handle spline roads | X | X |
| Generates road mesh | | |
| Priority roads at intersection | X | X |
| Can edit roads in Unity Editor | X | X |
| Can edit intersection corridors | X | X |
| ***Traffic*** | | |
| Traffic stops for player | X | X |
| Traffic stops for traffic | X | X |
| Traffic steers to avoid collision | | |
| Limit on traffic spawning | X | X |
| Traffic can be scripted | | |
| Traffic can return to path | X | X |
| Traffic can deviate from path on purpose for variation | | X |
| Traffic uses the road objects to observe others | ? | ? |
| Traffic uses the actual world to observer others (ray casts etc.) | ? | ? |
| Traffic uses blinkers before turning | X | X |
| ***Other*** | | |
| Seemingly good customer support | X | X |

**Appendix A table 2: Traffic packages**

## *Appendix B: Probability Graphs for Himanen et al.* [3]




**Appendix B Figure 1: Probability of drivers to react to pedestrians**




**Appendix B Figure 2: Probability of pedestrians to continues to walk**

## *Appendix C: Behavioural Sequence Model* [10]