CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Simulating Traffic Flows and Analysing Road Network Design

Investigating the relationship between road network design and traffic congestion

Bachelor's thesis in Computer science and engineering

MARTIN BLOM
FELIX JÖNSSON
HANNES KAULIO
MARCUS SCHAGERBERG
JAKOB WINDT

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Bachelor's thesis 2023

Simulating Traffic Flows and Analysing Road Network Design

Investigating the relationship between road network design and traffic congestion

MARTIN BLOM

FELIX JÖNSSON

HANNES KAULIO

MARCUS SCHAGERBERG

JAKOB WINDT

UNIVERSITY OF GOTHENBURG

CHALMERS

UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering

Chalmers University of Technology

University of Gothenburg

Gothenburg, Sweden 2023

Simulating Traffic Flows
Investigating the relationship between road network design and traffic congestion
MARTIN BLOM FELIX JÖNSSON HANNES KAULIO
MARCUS SCHAGERBERG JAKOB WINDT

Supervisor: Natasha Bianca Mangan, Interaction Design and Software Engineering
(if applicable) Advisor: Name, Company or Institute
Examiner: Name, Department

Bachelor's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2023

Simulating Traffic Flows
Investigating the relationship between road network design and traffic congestion
MARTIN BLOM, FELIX JÖNSSON, HANNES KAULIO, MARCUS SCHAGERBERG, JAKOB WINDT

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This document is *only* a LaTeX template. It is not meant to suggest a particular structure. Also, even if this document is written in English, is is not meant to suggest a report language. You can adopt it to your language of choice. In this document, the bibliography is hand made. However, we suggest that you strongly consider using BibTeX, to further automate the creation of the bibliography.

Keywords: put, here, keywords, describing, areas, the, work, belongs, to

Acknowledgements

If you want, you can here say thank your supervisor(s), company advisors, or other people that supported you during your project.

Martin Blom, Felix Jönsson, Hannes Kaulio, Marcus Schagerberg, Jakob Windt
Gothenburg, March 2023

Contents

List of Figures

List of Tables

Glossary

Agent: Autonomous systems that inhabits an environment and act based on predefined rules.
Agent Based Model (ABM): A computer simulation model in which agents interact with each other and their environment to produce emergent behavior.
Data Structure: A way of organizing and storing data in a computer so that i can be accessed, manipulated, and modified efficiently. Some common examples are arrays, stacks, and linked lists.
Information Visualization: Field that focuses on creating meaningful and easy to to interpret graphical representations of data.
MonoBehaviour: Base class for Unity scripts. Provides access to event functions such as Start(), Update(), and so on.
Pooling: A technique used in programming to improve performance by reusing objects instead of creating new ones.
Prefab: A reusable object in Unity that stores a configuration and can be used as a template for creating assets.
Scrum: The scrum agile project management framework provides structure and management of work and is popular among software development teams.
Scrum-boards: A bulletin board that keeps track of a backlog, the current sprint, and completed stories.
Story: In the scrum framework, a story is essentially a set of tasks that will result in a new or updated desired functionality/product.
Unity: Cross-platform game engine.
Unity Asset: A file containing reusable content that can be imported into Unity projects. Can be accessed through Unity's official platform or imported via third-party repositories.
User Testing: A method of testing and evaluating a product by observing and gathering data from real users.
UI: User Interface (UI) is the point between human-computer interactions. It is what is used for user interactions with the program.
UX: User Experience (UX) refers to the overall experience of the actual user of a product. The goal of good UX design is to create intuitive and enjoyable products.
C#: C# is a programming language developed by Microsoft that runs on the platform .NET Framework. C# is pronounced as "C sharp" and belongs to the programming language family of C.
C++: C++ is one of the most popular general purpose programming languages. C++ is pronounced as "C plus plus" and belongs to the programming language family of C.
A*: A* is a popular graph traversal and path searching algorithm due to it's completeness and optimal efficiency. A* is used to find the shortest possible path from one specified node to another.
Repository: A repository acts as a container that stores a projects files and their individual revision history.
Cubic Bézier Curve:

Wire Frames: Wire Frames depict how the UI layout will appear during different stages of the program.

1
Introduction

Make sure you have read the abstract of this template. This chapter presents the section levels that can be used in the template.

1.1 Section levels

The following table presents an overview of the section levels that are used in this document. The number of levels that are numbered and included in the table of contents is set in the settings file `settings.tex`. The levels are shown in Section 1.2.

Name	Command
Chapter	`\chapter{`*`Chapter name`*`}`
Section	`\section{`*`Section name`*`}`
Subsection	`\subsection{`*`Subsection name`*`}`

1.2 Related Work

1.2.1 Microscopic Traffic Simulations

There exists a plethora of different available tools for traffic simulation, which are in turn built upon different underlying models. In Nguyen's widely cited paper, he classifies the currently available simulations according to the following four categories with regards to their granularity of model: Macroscopic, Microscopic, Mesoscopic, and Nanoscopic. These tools allow researchers to answer complex questions and evaluate different scenarios in both real-time observations and through post-simulation data analysis.

Agent-based traffic models position themselves within the Microscopic category and allows for a highly realistic representation of traffic flow, where emergent behaviors such as congestion and bottleneck formation can occur due to the natural occurring interplay of the autonomous agents within the simulation.
Simulation of Urban MObility (SUMO) is a highly popular and freely available microscopic traffic simulation that was initially developed at the German Aerospace Center (DLR). It provides the users with the ability to model a range of transportation agents, including cars, buses, bicycles, and pedestrians, in both urban environments. The simulation is deterministic by default, but users have the option

to introduce stochastic processes in different ways, making it a highly versatile tool for traffic simulation and analysis.

The software offers various tools creating networks and editing these through a map editor which can also import and export network data from external sources. In addition to this, SUMO provides the user with features for visualizing the obtained data and analyzing it through various reports and plots. Users can also customize SUMO to accommodate their specific need through the application programming interface (API) and integrate the simulation with other software. SUMO is also capable of modeling emission based on vehicle type and speed.

Another popular for simulating traffic is the commercial software PTV Vissim designed by the German-based company PTV group which specializes in mobility and transportation solutions. It is designed to be quick and simple to set up with no scripting required by the user and comes with a highly customizable editor. The software is part of a larger suite named PTV Traffic Suite, which allows it to exchange data and collaborate across multiple platforms.

PTV Vissim offers a similar feature list to SUMO but differs in some important areas. Firstly, they are built upon different Car following models. SUMO implements the Krauss model which is based on the idea that drivers adjust their vehicle's speed and headway based on their perceived safety and comfort. Though a relatively easy to understand model, it has the disadvantage of assuming that the drivers only react to the speed and distance of the vehicle in front of them, and excludes a lot of factors such as the traffic signals, shape of the road, and driver psychology.

Meanwhile, PTV Vissim implements the Wiedemann models which share a lot of the same model parameters as the previously mentioned Krauss model but differ in their mathematical formulations and the way they calculate the acceleration of a vehicle. The model also introduces additional parameters, for example, a parameter for setting driver aggressiveness which regulates how risk-taking a driver is willing to be, and a parameter to regulate reaction time. Due to the additional parameters introduced here, the Wiedemann model is considered more realistic compared to the Krauss model, but is at the same time deemed to be more complex and requires a significant amount of parameter calibration.

Another crucial difference between the two simulation tools is that SUMO natively only supports graphical representation of a traffic environment in low detailed 2D, while PTV Vissim offers a feature rich 3D visualization. The latter provides a range of tools for customizing the 3D visualization, including options for importing third party 3D models, setting and creating custom textures, and defining various customized visual effects.

2 Theory

2.1 Unity

Unity is a cross-platform game engine used to create both 2D and 3D games. Unity supports a lot of features that speed up development time.

2.2 Bézier curves

A Bézier curve is a parametric curve between two points, that curves according to a set of intermediate points. The points are called control points, where the first and last point are the endpoints of the curve. A linear Bézier curve only has two points, which means that it is a line between the points. It is defined by the following function:

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, \quad 0 \leq t \leq 1$$

The parameter t is the ratio along the line, with $t = 0$ and $t = 1$ marking the endpoints. This is what is known as linear interpolation in mathematics. A linear Bézier curve is therefore simply a linear interpolation between the points P_0 and P_1. Let's define this as $P_0 \rightarrow P_1$. The quadratic Bézier curve consists of two linear interpolations:

$$A) \quad P_0 \rightarrow P_1$$

$$B) \quad P_1 \rightarrow P_2$$

It is then defined as the linear interpolation between these points, i.e $A \rightarrow B$. All linear interpolations in this case depend on the same t, which is what creates the curvature of the Bézier curves.

Since a quadratic Bézier curve has three points, it will have two endpoints as well as an additonal control point between them. By moving the control point, the shape of the Bézier curve can be altered. This is presented with the following examples, the first three of which have static endpoints demonstrating how the middle control point can be used to form the curve. The final example eludes to the fact that the control points can be placed anywhere without the requirement of any order.

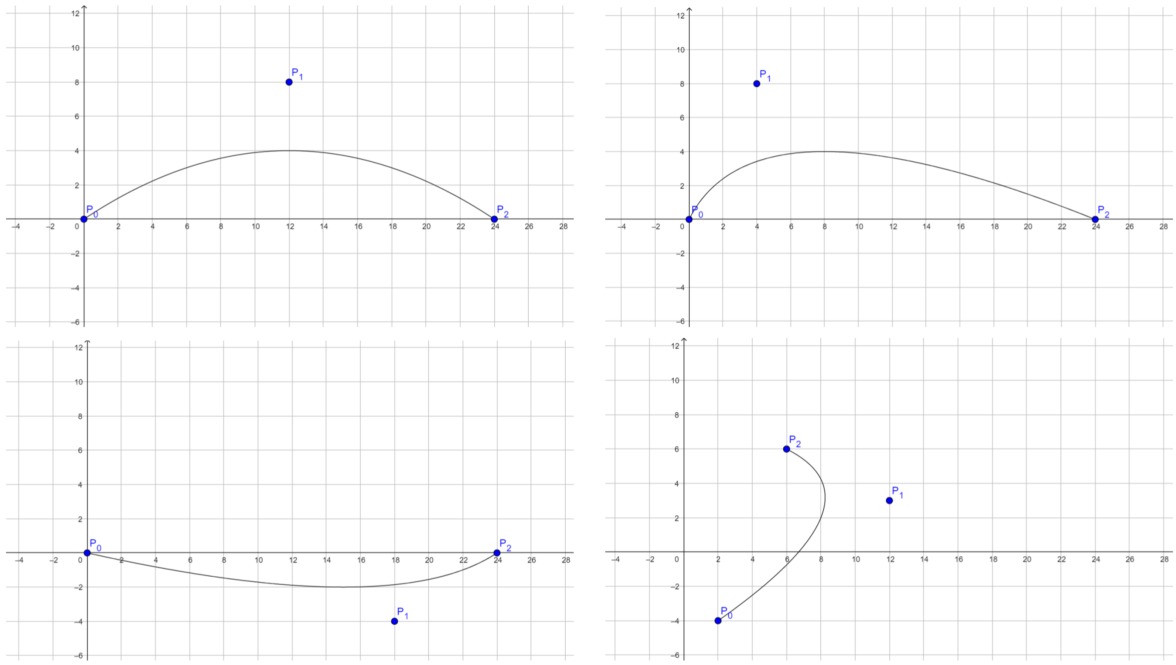

Figure 2.1: Four examples of quadratic Bézier curves

2.2.1 Cubic Bézier Curve

A cubic Bézier curve expands on the quadratic curve in the same fashion as the quadratic expanded on the linear Bézier curve, adding another layer of linear interpolations. A cubic Bézier curve has four control points, two of which are endpoints.

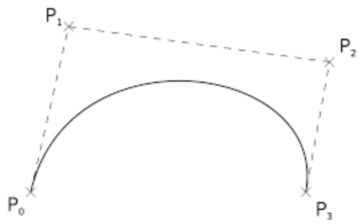

Figure 2.2: Cubic Bézier curve with control points P_0, P_1, P_2 and P_3

The cubic Bézier curve can be defined by the formula[1]:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3, \quad 0 \leq t \leq 1$$

Properties of the Cubic Bézier curve relevant to this paper are the following:

1. The endpoints P_0 and P_3 lay on the curve
2. The curve is continuous, infinitely differentiable, and the second derivatives are continuous.
3. The tangent line to the curve at the point P_0 is the line P_0P_1. The tangent to the curve at the point P_3 is the line P_2P_3.
4. Both P_1 and P_2 lay on the curve only if the curve is linear.

5. A Bézier curve is contained within the convex hull of the control points. For the application in Unity, this means that a Bézier curve is completely contained within the bounding box created by its control points.

2.2.2 De Casteljau's algorithm

In 1959 the french mathematician Paul de Casteljau constructed an algorithm for dividing a Bézier curve into two. The union of these Bézier segments is equivalent to the original curve.

2.2.3 Bézier Clipping

Finding the intersection points between two Bézier paths is not as straight forward as for something like two lines. To solve this, an algorithm called Bézier clipping explained in [2] can be used. It utilises the convex hull property of Bézier curves - and therefore Bézier paths - as well as de Casteljau's algorithm for splitting curves. An implementation for finding all intersection points between two Bézier paths using Bézier clipping is outlined below.

```
intersections ← Empty list
epsilon ← A value >0 small enough for the desired accuracy
procedure FindBezierPathIntersections(A, B)
    if A.BoundingBox does not intersect B.BoundingBox then
        return
    end if
    if A.BoundingBox.Size + B.BoundingBox.Size < epsilon then
        intersections ← Midpoint between A and B
        return
    end if
    A1, A2 ← SplitWithDeCasteljau(A, 0.5)
    B1, B2 ← SplitWithDeCasteljau(B, 0.5)
    FindBezierPathIntersections(A1, B1)
    FindBezierPathIntersections(A1, B2)
    FindBezierPathIntersections(A2, B1)
    FindBezierPathIntersections(A2, B2)
end procedure
```

Worth noting is the *Midpoint*, which is one of many possible approximations of the intersection point. For a small enough *epsilon*, the approximation used is trivial as the segments approaches points as *epsilon* approaches 0.

2.2.4 Composite Bézier curve

A composite Bézier curve is a spline made out of Bézier curves. The series of Bézier curves are joined together end to end with the start point of one curve coinciding with the end point of the other curve. This is used in the projects as it allows for chaining of cubic Bézier path segments creating a spline.

2.3 A* Algorithm

A* is an algorithm widely used for path finding and graph traversal [3]. Given a start, and end node in a weighted graph, the algorithm will find the shortest path between the nodes.

2.4 Procedural mesh generation

All physical objects in Unity have an associated mesh, i.e. their surfaces. A cube for example can be thought of as having a mesh consisting of 6 different surfaces. In computer graphics, a triangle mesh is a type of mesh where the surfaces are created through a set of points, called vertices. These vertices are then joined together by a set of triangles. Going back to the cube example, a cube in its simplest form would have 12 triangles and 8 vertices. The eight vertices are at the corners of the cube. Each face of the cube has the shape of a square, which can be created with two triangles, hence double the amount of triangles as square faces.

2.5 ABM

3
Methods

3.1 Tools

3.1.1 Unity

The traffic simulation tool is built in a well-known game-engine called Unity. There are a few reason why it was chosen as the development platform for the project instead of a similar game-engine like Unreal Engine. To begin with, C# is the main programming language supported by Unity, which some of the team members had previous experience with. Furthermore, C# is a higher level language compared to C++, the main language of Unreal Engine, making it easier for the team members without experience to learn. Because of this, the time it took to begin programming in the early stages of the project was most likely shorter, compared to if Unreal Engine was chosen as the platform.

Another reason would be that Unity comes with the Unity Asset Store, a marketplace for acquiring creator made assets. This feature is important because, for example, instead of having to create custom models for the vehicles, they could instead be purchased using the given budget. This saves a lot of time, that could be better spent on other parts of the project. One of the more notable purchased assets is Edy's Vehicle Physics that are used to rig vehicle models with realistic physics. Instead of having to develop custom vehicle physics for each model, the team could instead use the asset to quickly configure a model with physics.

The final reason why Unity was chosen, is because of its flexible developing structure. The level of customization available inside the engine is a lot greater when comparing to Unreal Engine. However, because of this, Unity ends up being more unstable whereas Unreal is far more stable and robust.

3.1.2 GitHub

A commonly used tool when developing software in larger groups is Git. Git is a free and open-source version control system that allows its users to collaborate in a efficient and easy way.

GitHub is an online software development platform that utilizes Git to store and track software projects. It allows for users to work in their own separate branches, and later merge those into the main repository. Before a team member could merge their new code to the main repository, the code would have to be reviewed by at least one other team member to ensure that the code was well commented, functional, and that it follow the C# coding standard.

3.1.3 Trello

It was decided early on that the projects work flow should follow the SCRUM and Agile software development practices. Trello is a website that hosts scrum-boards in an user-friendly way. This allowed the team to keep track of what needs to be worked on in the project during the sprints. A sprint is a set time period when new tickets are made, and completed.

3.1.4 Balsamiq Wireframes

During the first stage of creating a UI, its important to start with a simple mock-up design. This is what the tool, Balsamiq Wireframes, is used for. The user can quickly design wire frames depicting how the UI will appear during different times in the program. This includes everything from buttons to pop-up menu's that might appear in the simulation tool.

3.1.5 Third-Party Assets

Built into Unity is their asset store. Instead of creating everything from scratch, the team opted to purchase some assets. An asset can be anything from a 3D model to animation and scripts. The two main assets purchased for the project are Edy's Vehicle Physics and European Road Signs. Edy's Vehicle Physics is a package that includes a tool that allows its user to easily implement realistic vehicle physics into 3d models. This saves a substantial amount of time in the end because there would be no need to create custom physics attribute for each vehicle model.

As the name states, the European Road Signs assets include a plethora of street signs, as well as an editor to customize them. Without this asset, there would have been a need to create custom 3D models and texture, which no team member had previous experience with.

3.2 Simulation Design and Implementation

3.2.1 ABM

3.2.2 Road Generation

3.2.3 Intersection Generation

3.2.4 City Generation

3.2.5 Navigation

The basic navigational responsibility of each agent is the ability to follow the road lanes, avoid colliding into other agents, follow the traffic rules and being able to navigate to a given position.

In order to achieve a lane following agent, a navigation path is created for each road lane. The path is created as a double linked list. The nodes are insert along the lanes Bézier curve at a constant rate. Each node in the linked list store all information needed to navigate that lane. The position of the node, the agent that is currently on the node and special traffic rules the vehicles need to follow are stored on the node. Traffic signs such as stop sign are represented as a node and the traffic logic can be accessed by the agent when they encounter the node.

The agents steer towards a node that is a certain distance in front of the car. This distance is influenced by the current speed. By steering towards nodes that are in front of the agent, a smooth and reliable steering is achieved. Similarly to steering, breaking is accomplished by looking at nodes at a certain distance ahead. When a node with stop logic is found, the agent will break and stop before that node. This is done by looking for stop nodes at a distance ahead equal to the break distance of the agent. The agents also claim each node they are over so other agents can stop when the node at the break distance is claimed.

To enable the ability to navigate the roads, a weighted directed graph is created from the roads. The graph nodes are all the road endpoints and intersections. The edges between the nodes are weighted with a cost that is calculated as the distance * the speed limit. The agents navigate to a given end node by receiving a path of edges from the A* algorithm. When an agent drives up to a intersection, the intersections give a new road path to follow given the navigation node the agent is traveling to.

3.3 Graphics

3.3.1 Animations

3.3.2 Environment Materials and Textures

3.4 Performance

3.4.1 Quality vs Performance

3.4.2 Performance Benchmarks

3.5 Work flow

When developing any software larger than just a single use script, the amount of work and information can quickly grow beyond the level of ones own simultaneous comprehension. Therefore these kinds of projects require rigorous planning and strategizing to not get lost in all the different tasks and do them in a smooth and reasonable order, that allows for parallel continuous progress.

To achieve this a strict work flow framework was developed, where the first step was to analyze the work load and disposable time. This included drafting a time plan for the whole time scope of the project 3.1.

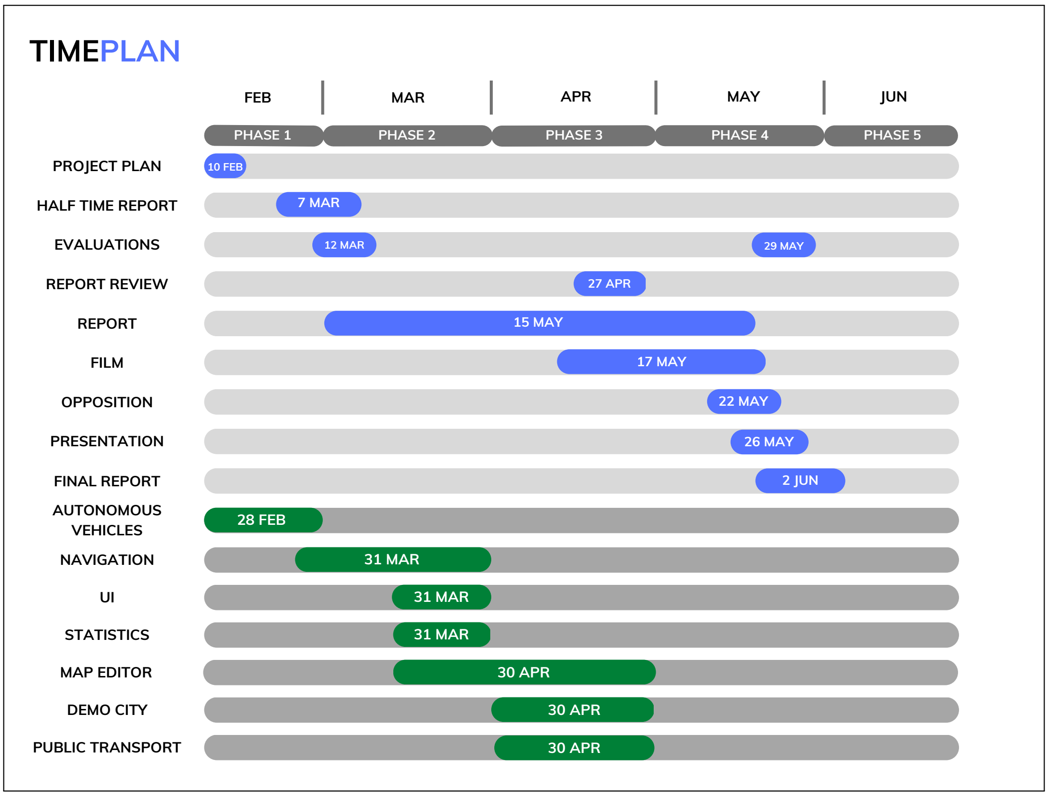

Figure 3.1: Project Time Plan

With this it's now much easier to keep track of the general progress of the project, as well as helping with planning short term goals. Coincidentally this is the next step of the work flow model. The short term goals where planned using a scrum framework with weekly sprints, explained in 3.5.1. These sprints were upheld for the duration of the project to keep a stead flow of progress, together with the time plan they create a very clear way of seeing the current state of completeness.

The third aspect of the work flow is the approval of progress. As mentioned earlier a large project requires a substantial amount of planning to not get lost. The approval of progress can be seen as just as important as the planning and execution itself. Without a popper method of approving new advancements/functionality, the project can quickly falter. If progress never goes through the process of approval many things can go wrong. Evidently, badly written code can cause issues that are easily preventable with a quick inspection. Code can even be considered good but with no input from the rest of the team, visions of how higher order elements will be implemented can differ. This can implicitly create more complex problems much further on, which can be very hard and time consuming to resolve. To solve this, code review's 3.5.2 for every change made are part of the work flow.

3.5.1 Weekly Sprints

The weekly sprint model stems from the scrum framework, which is a framework for developing and sustaining complex products. The sprint model follows 4 repeating stages of development: Planning, Implementation, Review and Retrospect.

Each sprint starts out in the planning stage, where a meeting is held to set up this sprints goals. This includes moving/creating stories for the backlog as well as the current sprint. The stories are mainly chosen by the project manager then developed in unison with the scrum master and input from the rest of the team.

The next stage of the sprint is the implementation itself. This is the time were the teams focus is solely on delivering good quality solutions to complete all of the current sprints stories, and eventually working on the backlog as time is presented.

Next up is the review stage, not to be confused with code reviewing 3.5.2. In this stage another meeting is held called a "Demo meeting", where all members get to do a small demonstration of all their progress during the sprint. This is an important step to onboard all members on new functionality and make sure that desired behaviour is achieved. When a story is regarded as fully complete it's archived to make room for new ones.

Lastly the retrospect stage, which is usually carried out following the review stage. In the retrospect stage the current sprints efficiency and quality is discussed. And plans/ways to increase these and the overall effectiveness are considered. When all is done the cycle begins anew until the project is done.

3.5.2 Code Reviewing

3.6 Testing

4

Results

Text ...

5
Conclusion

Text ...

Bibliography

[1] A. A. Shavez Kaleem, “Cubic bézier curves,” 2000. [Online]. Available: https://mse.redwoods.edu/darnold/math45/laproj/Fall2000/AlShav/bezier-dave.pdf
[2] T. Sederberg, “Computer aided geometric design,” 2012. [Online]. Available: https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=1000&context=facpub
[3] S. V. Konakalla, “A star algorithm,” 2014. [Online]. Available: http://cs.indstate.edu/~skonakalla/paper.pdf

A
Appendix 1

Figure A.1: Unity logo

B

Appendix 2

This is where we will place appendix 2