Universität Bremen

Department 3: Mathematics and Computer Sciences

BACHELOR THESIS (B.Sc.)

Spatial Manipulation of Ropes: Simulation and Visualisation

Henrik Axel Nickelmann
Digital Media (Media Computer Science)
Enrolment Number: 4240344

Angabin Rahman
Digital Media (Media Computer Science)
Enrolment Number: 4254053

Filed on January 6th, 2020

First Examiner: Dr. Thomas Barkowsky
Second Examiner: Prof. Dr. Ing. Udo Frese

Offizielle Erklärungen von

Nachname: Nickelmann Vorname: Henrik Axel

Matrikelnr.: 4240344

Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.
Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht.
Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein.
Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1. Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10% aller Abschlussarbeiten

2. Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach u. Jahr.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
☐ Ich bin <u>nicht</u> damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

____________________	____________________
Datum, Ort	Unterschrift

Offizielle Erklärungen von

Nachname: Rahman Vorname: Angabin

Matrikelnr.: 4254053

Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.
Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht.
Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein.
Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1. Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10% aller Abschlussarbeiten

2. Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach u. Jahr.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.
☐ Ich bin <u>nicht</u> damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

Datum, Ort	Unterschrift

Abstract

Reasoning about and acting in spatial environments is an important part of everyday life and also plays a key role in many scientific research areas. Spatial Cognition research deals with specifically these issues and contributes its findings to other research areas like cultural studies, study of medicine or linguistics.

The following thesis is making a contribution to spatial cognition research, with the aim to fill a niche. Examining three spatial-cognitive problems based on ropes, this work is able to provide a manipulable digital simulation of each problem. The problems addressed are *tying and untying knots*, *finding the shortest path between two waypoints on a route network map* and *transforming a circle into a square with the same surface area*. Within the frame of this thesis, these spatial problems are implemented without an extensive use of algorithms and complicated computations. The simulations focus on the direct manipulation of spatial objects in the spatial environment, imitating the human cognitive approach. This work contributes to the notion that cognitive, artificially intelligent agents can act in and reason about space and spatial objects similarly to humans and animals.

Table of Contents

1 Introduction [NICKELMANN & RAHMAN] 1

1.1 Background of Spatial Cognition Research [NICKELMANN] 4

1.1.1 Cognitive Agent [RAHMAN] 5

1.1.2 Spatial Problems [NICKELMANN] 6

Entangled Knot Problem [NICKELMANN + RAHMAN] 6

Shortest Path Problem [NICKELMANN + RAHMAN] 7

Squaring Circle Problem [NICKELMANN] 9

1.2 Methodical Approach [RAHMAN] 13

1.3 Chapter Overview [RAHMAN] 15

2 Current State of Spatial Cognition Research and Game Engines [NICKELMANN & RAHMAN] 17

2.1 Spatial Cognition Research and Transdisciplinary Domains [NICKELMANN] 17

2.1.1 Research Topics of Spatial Cognition Research [NICKELMANN] 18

2.1.2 Transdisciplinarity of Spatial Cognition Research [NICKELMANN] 19

2.2 Strong Spatial Cognition Research and the State of Rope-Based Simulations [NICKELMANN] 20

2.3 Introduction to Game Engines [RAHMAN] 22

3 Tool Selection: Game Engines [RAHMAN] 27

3.1 CryEngine [RAHMAN] 30

3.2 Unreal Engine 4 [RAHMAN] 31

3.3 Unity3D [RAHMAN] 33

3.4 Comparison: CryEngine, UnrealEngine 4 and Unity3D [RAHMAN] 34

4 Modeling and Implementation [NICKELMANN] 41

4.1 First Attempt at Modeling the Rope: Blender [NICKELMANN] 41

4.2 The Joint Component in Unity and Joint Types [NICKELMANN] 43

4.3 Constructing the Basic Rope and Implementing the Entangled Knot Simulation in Unity3D [NICKELMANN] 45

4.4 Building the Shortest Path Simulation [NICKELMANN] 51
4.5 Implementing the Squaring Circle Simulation [NICKELMANN] 54

5 Results [NICKELMANN + RAHMAN] 57
5.1 Entangled Knot Simulation [RAHMAN] 58
5.2 Shortest Path Simulation [RAHMAN] 63
5.3 Squaring Circle Simulation [RAHMAN + NICKELMANN] 66
5.4 Demo Video [RAHMAN] 70

6 Discussion and Conclusion [NICKELMANN + RAHMAN] 73
6.1 Entangled Knot Problem [RAHMAN] 74
6.2 Shortest Path Problem [RAHMAN] 76
6.3 Squaring Circle Problem [NICKELMANN] 78
6.4 Reached Goals & Outlook [NICKELMANN] 81

7 Literature 82

8 Appendix 87

1 Introduction [NICKELMANN & RAHMAN]

Humans and animals move, act and solve tasks in a spatial environment at all times. Largely intuitively, they do so by processing information about their environment, building a conceptual representation of this information and drawing conclusions for their actions from the representation. Autonomous robots also move and act in an environment and reason about spatial information, but mostly in a different way - they sense their environment and build an accurate representation of spatial information, calculating consequences for their actions via algorithmic reasoning. They only use information they can directly obtain from their environment.

In both cases, the nexus of spatial information processing, representation of spatial knowledge and spatial behaviour is referred to as *Spatial Cognition*, which is consolidated in a scientific discipline. The branch called *Strong Spatial Cognition* engages to convey the spatial behaviour of humans and animals to robots, aiming to evade complicated algorithms and to replace them with easier, more direct approaches oriented around the direct manipulation of physical objects. Initiated by the cognitive systems group (CoSy) at the University of Bremen[1], this subfield is novel and its objects of research not very common-place yet.

Inspired by the approaches from the work by CoSy and coming from a background of Digital Media, we were motivated to explore this particular field of research and introduce a digital viewpoint to the existing methods. The aim is to build a simulation to three particular spatial cognitive problems and, by that, provide existing approaches with more efficiency. Furthermore, the simulations are following the ideas of *Strong Spatial Cognition,* as described above, as well as further specifications that we set as criteria. The goal is to mimic a realistic cable behaviour, wherefore the requirements include options for the manipulation of the objects and an accurate transformation of the objects according to the manipulation by an agent.

[1] https://cosy.informatik.uni-bremen.de/index.html, retrieved December 22, 2019.

To ensure a realistic environment and intuitive visuals, the simulation is implemented in a 3D environment instead of a 2D environment. This is due to the fact that the implementation of a 3D-object allows a 2D usage, however, this is not possible the other way around. As we required a 3D-object for our first examined problem the entangled knot problem, we decided to use the resulting objects for the other problems. Moreover this work is focussed around the functionality of the simulation and therefore neglects any visual embellishments of the simulation.

In the beginning of our research, we focused on finding similar approaches in prior works in this research area. However, we were astonished to find that this was a niche and therefore not very well covered at that point. Motivated to contribute novel approaches to this area and inspire future applications, this thesis engages to work on an explorative approach. This approach is aiming to introduce a digital factor by implementing a simulation to the spatial cognitive problems and expand existing boundaries of this field of research.

During our primary examination for existing simulations, we found several simulations created with game engines (further explained in chapter 2.3), specifically with Unity3D[2] by Unity Technologies, which were not fulfilling our criteria. The simulations were able to fulfil most behavioural attributes, but were unable to manipulate to the extent of solving the spatial cognitive problems or did not consider ideas of *Strong Spatial Cognition.* Due to the fact that this thesis is notably inspired by the approaches of CoSy group, our criteria for the simulations are derived from their existing models. This thesis can be viewed as the explorative work to instantiate a digital version of the prior work. Therefore our criteria are quite preset and only needed further consideration in the digital implementation.The criteria we set at the beginning of our work were based on the specifications of the spatial problems we were examining.

This work intends to implement simulations that each depict a specific spatial problem, to offer novel and less algorithm-based ways of tackling these problems as

[2] https://unity.com/de, retrieved December 01, 2019.

a cognitive agent, and to outline why the simulations are of value for future spatial cognition research. Hence, the following chapters of this introduction will provide an overview of the scientific field of Spatial Cognition and outline the structure of the thesis. Key terms like *cognitive agent* and *spatial task/ spatial problem* are described so that the chapters on the theoretical background, the documentation and the conclusion can be followed and understood. Afterwards, the methods utilized to implement the simulations and the thesis as a whole are depicted. Closing the introduction is an overview of each following chapter.

1.1 Background of Spatial Cognition Research [NICKELMANN]

Spatial Cognition is a broad term referring to what information a cognitive agent can have about space, like distances or directions, and how to process that information (Vasilyeva and Lourenco 2012). The term *cognitive agent* includes humans, animals and autonomous robots, who form a full cognitive system together with their bodies and their environment. In classical information processing there is a division between the brain or the AI (Artificial Intelligence) system and the perception of a spatial problem, the agent's actions, its body and the environment. Information is abstracted from the real-world problem and translated into a knowledge representation, existing within the brain or AI system to perform spatial reasoning. However, in a *full cognitive system*, this distinction is dropped in favor of a contiguous system, of which the spatial problem itself is a part of. This has the advantage of bypassing the abstraction of spatial relations and being able to simulate them, and spatial interactions, through motion models (Freksa et al. 2017a).

Cognitive agents solve *spatial problems*, or *spatial tasks*, either by a combination of knowledge representation and algorithms (in the case of AI), or without conscious knowledge processing (in the case of humans or animals). The subfield *Strong Spatial Cognition* deals with problems regarding the direct use of spatial and temporal structures by AI, considering embodiment and preserving all properties of the environment, while neglecting abstracted knowledge representations and algorithmic reasoning. The goals are to (1) generate less CPU-intensive models for AI, so that it can act more efficiently, and to (2) more accurately model cognitive processes and their dynamics, complexity and scalability (Dreyfus 1978, Freksa 2015, Freksa et al. 2017a).

1.1.1 Cognitive Agent [RAHMAN]

The word ‘agent’ in correlation to the computer world, may be allocated to many entities. Here it is mainly referred to computational agents, specifically ‘cognitive agents’. In the following segment we are analysing and examining the role of an cognitive agent and its role in our thesis.

It might be advisable to recollect the term ‘cognitive concepts’ as well as the ‘intentional stance’ at this point. People often use cognitive concepts to understand how others behave. To understand the complex role of these agents we need to examine the challenges they are facing. Although computational agents outperform the human agents in many aspects, such as accuracy, iterating problems and gathering data, they are still behind in other aspects. Not only can a cognitive agent not take over all human processes, but it is vastly limited to a specific set of tasks. Yet as humans we tend to assign these agents with cognitive concepts, such as beliefs, knowledge, desires and intentions. Although it is commonly known that the agents are programmed. Most computational processes merely mimic the human capabilities. Even with the involvement of deep learning the agents are bound to have a programmed starting point, specifically implemented handling of situations and technical restrictions. The smarter the agents get, the higher we raise our expectations and the more we ascribe them intention to their action. It is a common phrase in our daily work to say ‘He doesn’t like me today.’ or ‘It always does the opposite of what I want’, referring to the computational assistant/agent. By claiming that it can act differently or chooses not to function when used by a specific person, we assign a human behaviour to a pre-programmed agent.

A cognitive agent has yet to learn to imitate the irrational human instinct; although, this being the most unstable and inexplicable feature of the human mind, it is indispensable. This feature allows us to take actions faster, omitting long and cumbersome algorithmic comparisons. In fact, it allows us to drive the car, without rationally examining the other drivers next steps. Furthermore, it enables us to estimate our surroundings, is the key in our problem-solving and continuously saves us from potential danger. In our thesis we are designing a simulation of different spatial problems. The goal is to benefit from the human approach to solve problems and enhance these in the processes of cognitive agents.Thus trying to ensure a faster and more intuitive operation.

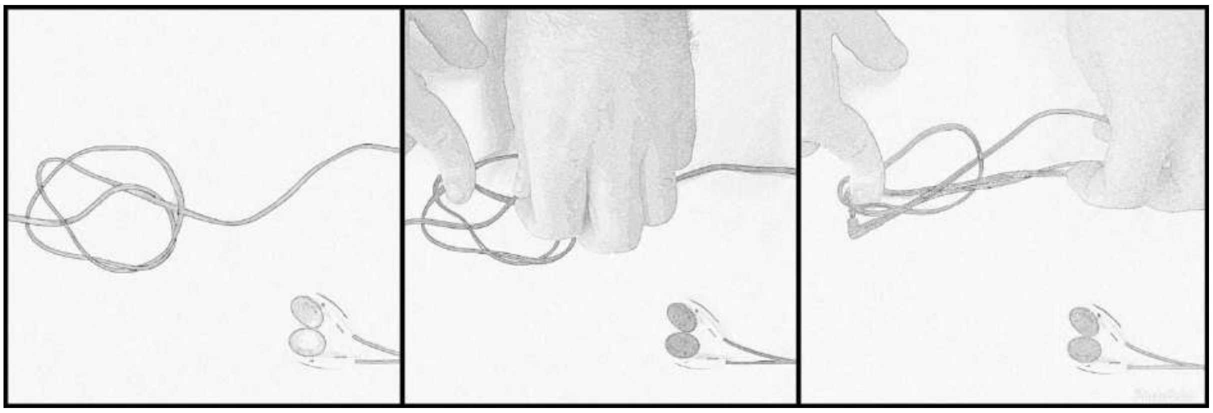
1.1.2 Spatial Problems [NICKELMANN]

Spatial problems, or spatial tasks, are specific configurations of a spatial environment or a spatial structure which necessitate a cognitive agent to act in certain ways to find a novel solution to that problem or to achieve a known goal. In this thesis, we are demonstrating three digital simulations, each corresponding to a different spatial problem a cognitive agent might encounter at one point.

Entangled Knot Problem [NICKELMANN + RAHMAN]

There are many domains in which tying or untying a knot can occur - from the daily lacing of shoes, to untangling earbuds, to the many forms of medical sutures. The necessity to apply knot-tying in many different areas attaches importance to the implementation of a digital simulation for knot-tying, considering the increasing incorporation of digital devices in home, agents and the overall notion of the Internet of Things (IoT). Apart from the importance of a knot and the ability to tie or untie one in our everyday lives, the most remarkable thing is the fact that this simple intuitive process, as seen in draft 1, is hard to recreate. What seems quite trivial and easily done - by toddlers even - strains the scientists to this date.

The formalisation of this process that combines cognitive and spatial understanding has been implemented by relying on the heavy use of algorithms (Phillips et al. 2002, Brown et al. 2004, Wang et al. 2005, Mayer et al. 2008), especially for the medical domain. There has not yet been an significant effort made to implement such a simulation with strong spatial cognition in mind, which we would like to change in the course of our thesis. The final outcome of this work is aiming to provide a simulation that is based upon the strong spatial cognition. Furthermore draft 1 is providing us the guideline to follow along in our digital approach.



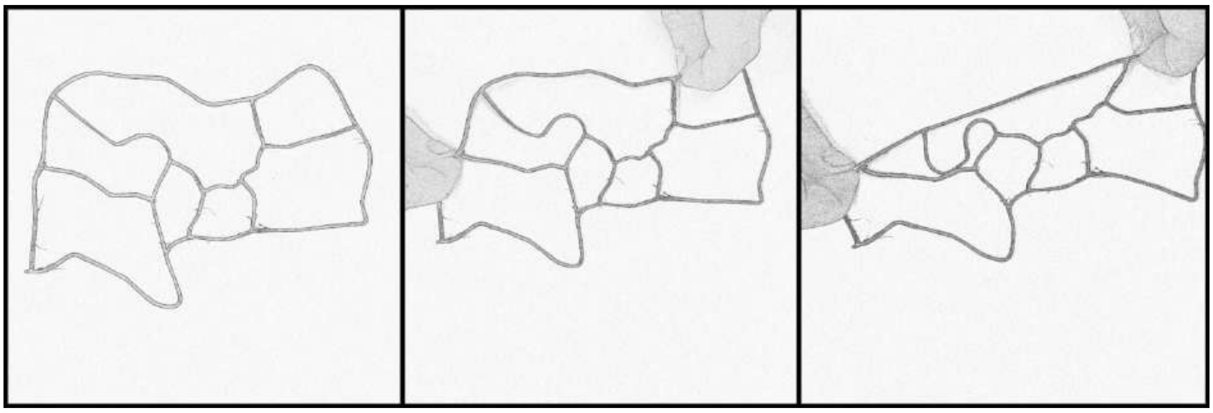
Draft 1: Untying of a headphone cable executed by a human agent displaying the intuitive process (that is omitting heavy algorithmic formulations). This draft has been used as inspiration and set the criteria to the destined digital simulation attributes and functionality

Currently cognitive AI agents are able to tie or untie knots by algorithmic processing, however, a direct manipulation of a spatial cognitive problem would be useful and increase the overall performance. Also, we argue that it would lead to more flexibility as the AI is acting in a more exploratory manner and can better react to a changing spatial environment. Thus, a digital simulation of knot (un)tying that is focussed on the direct manipulation of spatio-temporal configurations would prove very useful and would be a valuable contribution to strong spatial cognition research.

Shortest Path Problem [NICKELMANN + RAHMAN]

The second problem that is examined in this thesis belongs to the category of navigation. This field has experienced strong endorsement not only from the automobile industry but has been of general interest and continues to search for more efficient ways to improve the navigational process. It is also of interest for

Spatial Cognition research; for example, Uttal (2000) outlines how the use of maps influences the development of spatial cognition in children and vice versa.
In the frame of this work these facts are taken into consideration, leading us to the approach to find the shortest path, a problem that was worked on by the CoSy group. The approach is following the idea of graph theory, which is finding the shortest path between two nodes (Dewdney 1988). Building upon the non-algorithmic strings-and-pins approach, the CoSy group went on by printing out a 3D net that imitates a road network on a map (Freksa 2015, Freksa et al. 2016). This analog approach was designed to be manually pulled apart by an agent and detect the shortest path by visually distinguishing the most stretched path between two points (see draft 2).

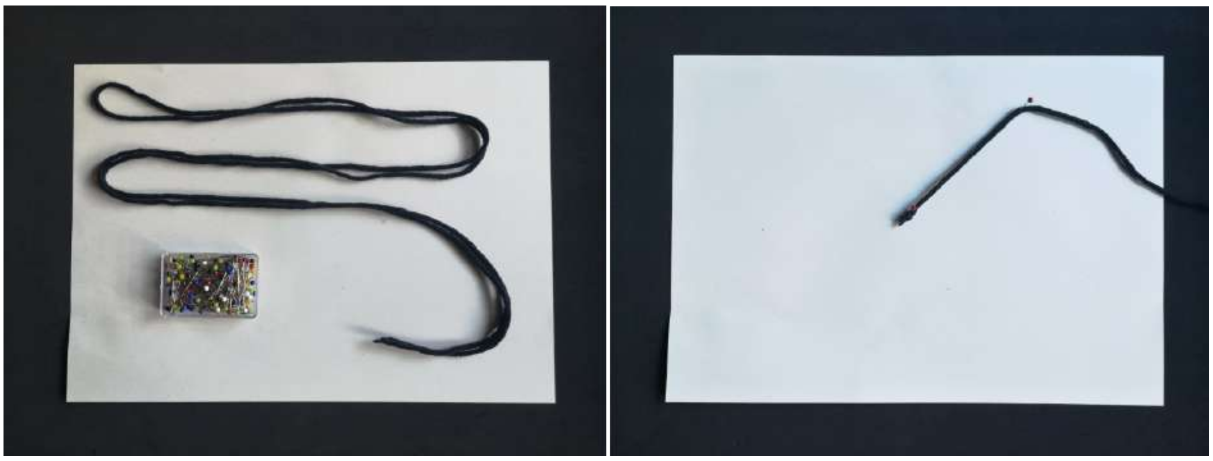

Draft 2: Pulling of a reduced map, printed out by a 3D printer. The displayed process is used to determine the shortest path between two points by cognitive AI or human agents. This draft has been used as inspiration and set the criteria to the destined digital simulation attributes and functionality

Pulling apart two points of a net to find the shortest path between two points is a very natural and overall simple solution to this cognitive spatial problem (Freksa 2015). This might not be a very intuitive solution if you confront a cognitive agent with a strings-and-pins route network and simply ask them to find the shortest path between two given points. However, once established, it takes minimal efforts and requires very little movement by the cognitive agents - they are now able to manipulate a spatial configuration to find out certain spatial information. If a map is large-scaled, with many different roads and waypoints, it would take a lot of computations to find the shortest path between two given points. However, if you can simply pull at those points, the effort is reduced drastically. Although the existing approach by CoSy is already covering many points, the simulation resulting from this thesis will further reduce the usage of resources as e.g. the net printout is no longer required. This next step to implement a simulation of a string-based map to be used by AI agents is only sensible, so that it can find the shortest path between two points way more efficiently. Thereby the simulation can pick up on the simplicity of the process and still withdraw the translation of this process into the real world.

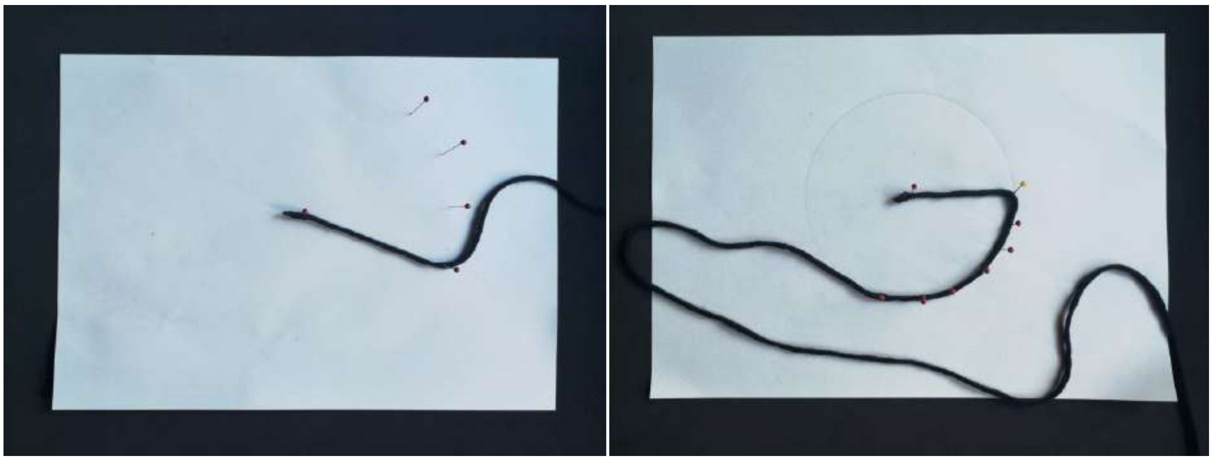
Squaring Circle Problem [NICKELMANN]

The squaring circle problem is a common problem in geometry which has kept mathematicians busy for centuries, and cannot be solved by classical, straightedge-and-compass approaches (Schubert 1891, Hobson 1913, Seidenberg 1962, Engels 1977, Saraswathi 1999, Dani 2012, Crabtree 2016). The task is to transform a circle into a square with the same surface area. Interestingly, the strings and pins approach was already used in Ancient Egypt and Ancient India, dating back to around 800 to 600 BC in India as described in the Sulvasutra records (Dani 2012, Crabtree 2016). Additionally, in both cultures, approximations for Pi were already pretty accurate, considering the time period in which they were calculated - in Ancient India, the approximation was the square root of ten, or 3,1622 (Dani 2012), whereas in Ancient Egypt, it was 3,1605 (Engels 1977). Due to the prevalence of the straightedge-and-compass approach ever since the days of Ancient Greek mathematics (Crabtree 2016), the task of squaring a circle was thought to be impossible, and in the late 1800s, it was pronounced as such (Schubert 1891). However, it turns out to be solvable if you change your utilized tools. With strings and pins, and without any mathematical computations, you can accurately transform a circle into a square with the same surface area (Crabtree 2016). This approach has several advantages: First, strings retain some mobility even after they are pinned down to indicate a line, whereas pencil marks do not, giving the option of floating pins to represent variables, for example. Second, with multiple pins, you can easily describe ellipses and other more complex geometric figures, as you can slide the pins along strings. Third, with one end of a string being fixed at the center of a circle, you can describe not only all points on the circumference line, but all points on the surface area of the circle. Most importantly though, with strings and pins, it is possible to directly transform the circle’s circumference to a straight line, as strings are deformable - this is not possible with straightedge and compass (Freksa et al. 2016).

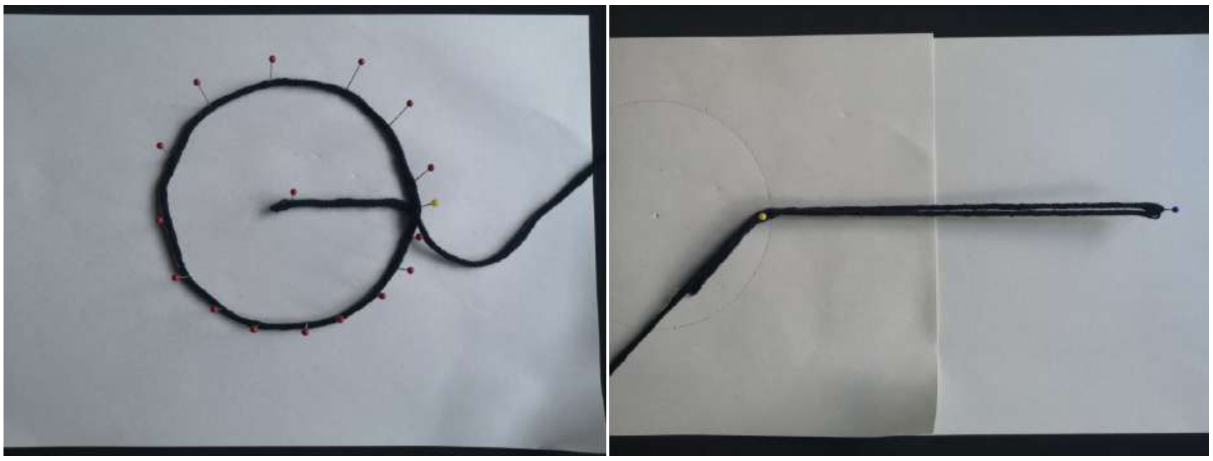
The simulation of this problem is most suitable for demonstrating that spatial problems can be solved without much algorithmic reasoning or complicated mathematical computations. It is implemented in such a way as to mimic a strings-and-pins approach to this problem (Freksa et al. 2016). Naturally, it is a great fit for strong spatial cognition research, as the strings-and-pins approach does not rely on any algorithms, but instead uses the properties of spatial objects (strings and pins) to transform the area of a circle into a square with the same surface area .


Draft 3: Ordinary strings and pins as the utilized tools to tackle the problem of squaring the circle (left); Setting an arbitrary radius for the circle (right)

The squaring circle problem can be solved conveniently by the strings-and-pins approach (Freksa et al. 2016). By using ordinary strings and pins (Draft 3, left), you can indicate any distance in a two-dimensional plane, and thus, indicate a radius (with arbitrary length) of a circle (Draft 3, right).


Draft 4: Marking the whole circumference of a circle (left); Pinning down the string on the circumference (right)

Next you can pin the string down at any point that is the same length away from the center as the previously established radius (Draft 4, left). By doing that, you can create the circumference of the corresponding circle, the marks left by the pinning indicating said circumference. Alternatively, you can tie a pencil to the floating end of the string and draw out the circumference line. Afterwards, you pull out the string from the center, pinning it down on the circumference. That point marks one end of the circumference’s length. After pinning down the string along the previously left marks (Draft 4, right) or the pencil-drawn circumference line, the entire circumference (Draft 5, left) can be discerned and further utilized.


Draft 5: The whole radius plus circumference marked and able to be transformed back into a straight line (left); The half circumference to be used for the following steps in the solution process (right)

At this point, you have solved the problem of extrapolating the circumference of any circle, which cannot be solved by the classical straightedge-and-compass approach (Freksa et al. 2016). This circumference can now be rolled out into a straight line, which can be halved afterwards (Draft 5, right) so that you can use the half circumference in the later steps of the solution process.

The strings-and-pins approach illustrates that geometric problems can be solved by using the properties of spatial objects instead of mathematical computations. With the more common straightedge-and-compass approach, you are not able to solve the squaring circle problem, as you cannot accurately extrapolate the circle's circumference. The goal of simulating this approach digitally is to show that digital solutions to spatial problems focusing on the direct manipulation of spatial objects can be more efficient than common, algorithm-heavy digital solutions. Furthermore, the underlying process of solving the problem is more similar to natural and human approaches. Thus, cognitive AI agents acting based on such a simulation will lead to them acting more like a human, and, in the end, to them interacting better with cognitive human agents.
The subsequent steps after ascertaining the circle's circumference will not be included in the simulation, as the main goal is to show that the strings-and-pins approach can solve problems classical approaches cannot solve. To use this simulation in a cognitive AI system would necessitate the implementation of the absent steps, however, implementing them is not too difficult, but it would go beyond the scope of this thesis.

1.2 Methodical Approach [RAHMAN]

This thesis is investigating a suitable approach to simulate three spatial cognitive problems, as mentioned above. The interesting and significant facts about these problems are that most of them are solved by humans intuitively. This intuitive behaviour, without algorithmic formalising, is what is aimed at in our work. In order to find an alternative to algorithmic-based solutions we are aiming to implement a spatio-cognitive method for a cognitive agent in AI. The overall aim is to explore this particular field of research and finally build a simulation to prove that there is a way to solve these cognitive problems by an cognitive agent more efficiently and less ressource-bound. Current methods and approaches in this field use 3D printouts, to implement the idea of *Strong Spatial Cognition*. For the simulation we set the goal to make it a manipulable demonstration of the three spatial cognitive problems. The visual appeal of the simulation is not in the focus of this work and is therefore mostly kept simple. This is done consciously to signal the importance of the functionality and the main focus of this simulation. Therefore we continue to directly explore the possibilities to simulate the spatial cognitive problems and evaluate possible tools to use for the simulation. Evidently, the character of the research, as well as the preparation, are significantly different from a hypothesis-based thesis.

We set the goal to model the components, assemble and visualise the rope, before we go on to simulate the rope with characteristic behaviour. The ultimate aim is to have a rope that is working according to our criteria and is able to be spatially manipulated by an agent. An explorative approach appeared to be the most suitable way to research this matter. In the beginning we started to collect information on this topic and find potential programs that would be able to simulate a rope. During this part of the research we also assembled our first design ideas. Furthermore, we undertook several paper-designed approaches to test out which attributes and specifications we need to implement in our simulation. Even though some programs were able to simulate a rope, they often did not match our criteria for fulfilling the spatial cognition tasks.

The idea to this thesis emerged during a discussion of prior approaches to research ways to solve spatial-cognitive tasks with computational cognitive agents. The research group CoSy of the Bremen University were already profoundly researching in the area. Apart from a manifold of studies to look deeper in this area of expertise, we got interested in a few rope-based approaches/experiments. These approaches significantly stood out, due to their simplified ways to solve the spatial problems. Besides the fact that they could replace lengthy algorithmic solutions, they demonstrated the intelligent and intuitive solving techniques of a human agent. The aim is to make an computational cognitive agent use and benefit from these solving techniques.

At that point, all these approaches were printed in a 3D printer model and then actually used in the real world. So to some extent this approach is fulfilling its task already. The cognitive agent is able to solve the spatial task with the human technique. Now the next step is to omit bringing the 2D into the real world. Introducing a simulation might be beneficial to that. Our idea is, as previously mentioned, to simulate and thereby visualise this whole process.Thus we are not only increasing the speed of this process, but also enhancing the overall efficiency, by skipping the entire 2D-3D translation procedure. The manipulations of the rope, such as undoing knots, forming a loop, finding the shortest paths or measuring the length of a circle (in order to transform it into a square with the same surface area), are all combined in the simulation. Likewise delivered in the real world, they are now all part of the simulation. As we are examining more than one rope-based manipulation, we are confident to research this as a team of two. Since all of these spatial problems are rope based, thus quite similar to approach, we decided to work on ‘all’ of them. It serves the purpose not to neglect one experiment in favour of the other, or entirely dispose of one subtopic. By forming a team, we hope to give sufficient attention to each subtopic and generate a competent visualisation of spatial cognitive tasks in a short amount of time.

1.3 Chapter Overview [RAHMAN]

The first chapter *Introduction* of this thesis is stating our intention and motivation to pursue our work. It further introduced the term *Cognitive Agent* and illustrates the *Spatial Problems*, before focussing on three particular spatial cognitive problems assessed in this work. The problems described are the *entangled knot problem*, the *shortest path* and the squaring circle. This chapter closes with the detailed description of the *Methodical Approach* that is used to examine the spatio-cognitive problems and the *Chapter Overview*, to provide the structural composition of this thesis.

The second chapter *Current State of Spatial Cognition research and Game Engines*, is a informing about the current state of the art and relevance of this thesis to the contemporary developments in this research domain. Furthermore, a rough overview as well as definitions that are relevant to our work, are provided. This is to allow an easy understanding and access to this work, even for readers that are entirely new to this topic. This chapter is introducing game engines as a tool of choice, thereby initiating the third chapter.

The third chapter *Game Engines* is introducing and comparing three game engines, by emphasizing on their qualities and attributes that are required for the solution of the spatial cognitive problems examined. The compared game engines are *CryEngine* by Crytek, *Unreal Engine 4* by Epic Games and finally *Unity3D* by Unity Technologies. By this comparison we intend to provide the technical facts and thought processes that helped us to choose our tool for our purpose.

In the following fourth chapter *Modelling and Implementation,* we are giving an overview of all methods used and functions implemented. We are giving an insight of the modelling process and the implementation process that were required to construct our simulations. Besides a detailed documentation of the functions used during the implementation, it further provides images to accompany the different stages of the simulations. The idea is to be transparent about our methods and thus enable or inspire further work in this field.

In chapter five we provide the results that we produced in the scope of this thesis. All three simulations, as well as the demovideo that we produced, are separately stated and illustrated with pictures from the simulations. This chapter will describe the entangled knot problem, the shortest path problem, as well as the squaring circle problem elaborately. The demovideo is described with a precise minutes disclosure to exactly describe the visual display to the audience.

Finally this thesis is closed by the last and sixth chapter *Discussion and Conclusion*. This last chapter of this thesis comprises a discussion of the final simulations and the conclusion of where this thesis can be placed. Subsequently, that chapter states the reached goals and limitations of the work. An outlook on possible future applications and encouragements for immediate research directions in this field is closing the thesis.

2 Current State of Spatial Cognition Research and Game Engines [NICKELMANN & RAHMAN]

This chapter gives an overview over the topics of spatial cognition research and transdisciplinary research. There are many research areas interested in aspects of spatial cognition and many scientists included findings of spatial cognition research in their studies. Afterwards, the subfield of strong spatial cognition is introduced. Strong spatial cognition is a novel research area birthed by the CoSy group at the University of Bremen. Many interesting ideas and approaches to topics of general spatial cognition research are advanced and refined. In the context of strong spatial cognition research, the CoSy group focuses on cognitive agents acting directly in a spatial environment and with spatial objects, detached from extensive algorithm use.
Afterwards, a general introduction to game engines is closing this chapter, including some criteria for choosing a game engine for the simulations. A description of the most commonly used game engines as well as a comparison and a justification for the game engine chosen for the implementation is given in the following Chapter 3.

2.1 Spatial Cognition Research and Transdisciplinary Domains [NICKELMANN]

Spatial cognition research deals with questions regarding the interaction of cognitive agents with spatial structures: How do cognitive agents reason about the spatial environment and spatial objects? How do they translate their knowledge about the spatial structures into actions? Or why do they behave differently than other cognitive agents? It is a very broad and manifold research field with a lot of possible research topics and questions. Those topics and questions can be applied in many situations and improve our understanding of the world around us. For example, spatial problem-solving skills are of great importance in many technological or engineering domains, in all forms of sport and, especially since the emergence of Augmented and Virtual Reality, in many digital domains as well.

2.1.1 Research Topics of Spatial Cognition Research [NICKELMANN]

A common research topic of spatial cognition research is the question of how cognitive agents reason about spatial structures and spatial relations (Byrne and Johnson-Laird 1989, Buckley et al. 2019). Researching this topic would not only involve which logical inferences cognitive agents make about spatial relations, but also on what basis. Are there general principles underlying the thought about spatial configurations? If so, is it possible to express them as formulas?

Another research topic related to spatial reasoning is spatial problem solving (Freksa et al. 2017a & 2017b, Buckley et al. 2019). Spatial problem solving also includes the actual real-world spatial structure of what is reasoned about (or a spatial representation of it), focusing on finding possible solutions to a spatial problem. Common spatial problems are finding a shortest path on a route map, detecting if an object is in front of or behind another object, or mundane activities like opening doors, driving a car or throwing a basketball. As spatial problems and tasks are around all of us all the time, applying the findings of research on spatial problem solving has a great potential for developing new approaches to these problems and finding novel, more efficient ones that can improve our daily lives.

Representations of spatial knowledge in a cognitive agent's brain (or network of sensors and actuators) are also of interest to spatial cognition researchers (Kuipers 1978, Uttal 2000, Tomai 2004). This subdomain deals with conceptual representations of perceived spatial information and how that representation is built within the agent's mind. For example, a *cognitive map* is an abstracted, but coherent model of the large-scale environment surrounding a cognitive agent, guiding them when navigating through said environment. It is a representation constructed via observing and filtering key aspects of the spatial environment. A mental step-by-step route description is also a representation of spatial knowledge, utilizing commemorated actions instead of a mental picture. There are many different representations of spatial knowledge - examining them can provide novel and practical approaches to spatial tasks.

These are only some topics spatial cognition research deals with. It is a very broad field that commonly thinks outside its own box, providing its findings to other research areas and importing knowledge from them. Thus, spatial cognition researchers often engage in transdisciplinary research, developing and expanding the field of spatial cognition research itself. To signify this, a sample of transdisciplinary research approaches is given in the following subchapter.

2.1.2 Transdisciplinarity of Spatial Cognition Research [NICKELMANN]

Spatial cognition research has a lot of interest in other related research fields. Researchers not only deal with questions regarding spatial cognition itself, but branch off into other research areas like biology, sociology or mathematics. What follows is an overview over related literature and research to give a better understanding of what spatial cognition research can contribute to other domains.

Burgess (2008) summarizes advances in the field of Cognitive Neuroscience and how they relate to Spatial Cognition, focusing on spatial memory and knowledge representation (for example egocentric versus allocentric representations). He also addresses how the activities of neural networks evoke corresponding spatial behaviour. Sinha and Jensen de López (2000) write about combining both cognitive linguistics and socio-cultural approaches to language and cognition to develop a broader picture of both research areas. They argue that this transdisciplinary approach can yield a better understanding of the embodiment of culture and spatial cognition and can extend it beyond the human body. Haun et al. (2011) also write on the influence of culture on spatial cognition, focusing on the expression of spatial relations via language. They found that both preference of spatial strategies and competence to apply them vary across cultures and that this variation stays consistent with increased complexity in spatial arrangement.

Other research investigates: The role of hand gestures in spatial cognition and how it influences expression of, communication and reasoning about spatial information (Alibali 2005); cognitive space as non-metric, topological space and how it can extend the findings of spatial configuration analysis on how people move in different locations (Penn 2003); or how spatial ability impacts the capacity for and the approach to spatial problem solving (Buckley et al. 2019).

We already mentioned the research of the CoSy group at the University of Bremen in the introduction to Chapter 2. They carry out research in a previously uncharted research territory, called *strong spatial cognition*. It deals with topics of spatial cognition and tries to expand the notion of cognitive AI agents being able to handle spatial information and derive actions from it in a manner akin to human or animal agents. This means that the aim of their research is to develop approaches which allow direct manipulation of spatial configurations and diminish use of algorithms and computations. As the three simulations we are working on are implemented with the same goal in mind, the following subchapter will provide an insight into the novel research area of strong spatial cognition. Furthermore, it transitions towards illustrating the current use of ropes for digital simulations and the rationale for why they do not fit the criteria for strong spatial cognition and our simulations as they are intended.

2.2 Strong Spatial Cognition Research and the State of Rope-Based Simulations [NICKELMANN]

Strong spatial cognition research reflects upon the role of AI in usual spatial cognition research and on how to use spatial cognition methods to develop better AI. Nebel and Freksa (2011) outline artificial cognitive systems and their strengths and weaknesses compared to natural cognitive systems. They argue that artificial cognitive systems are “excellent candidates to assist human cognizers and to complement their weaknesses” (Nebel and Freksa 2011). Freksa (2015) also illustrates some spatial tasks which are of use for AI agents when implemented as a computer simulation. These tasks include determining which object is in front of another object, performing spatial operations without detailed knowledge about the operation or the environment, or interpretation of road or street maps. He also writes about the idea of representing these maps via strings and pins which enables pulling at two points of the string map to find out the shortest path between those two points, which is the most stretched. This approach is an easy and reproducible solution to the shortest path problem and has not been tried out yet as a digital simulation to be used by artificial cognitive agents without conscious, algorithm-based knowledge processing. In general, there are many rope-based spatial tasks which offer different solutions to spatial problems than tasks not using ropes. For example, it is not possible to construct a square from a circle with the same surface area as the circle by using the common geometric tools like a straightedge and a compass. It is, however, possible to do this with strings and pins. Freksa et al. (2016) write about the advantages the strings-and-pins approach has over the straightedge-and-compass approach. It can be the basis for a digital simulation of the squaring circle problem.

We already mentioned that there has not yet been an attempt to implement a digital version of the shortest path problem or the squaring circle problem via the strings-and-pins approach. However, Brown et al. (2004) worked on a knot-tying simulator where users can “grasp and smoothly manipulate a virtual rope and [...] tie arbitrary knots” (Brown et al. 2004). They focus on how to overcome common problems in regards to contact detection and contact management, and illustrate how their simulator could be of use for domains like surgical suturing, sailing or rock climbing. Their work is of great value for these domains, but in our case, we want to use methods which are not reliant on an extensive use of algorithms - the main goal of our work is to develop simulations which work towards cognitive AI agents making direct use of spatial and temporal information while, more or less, neglecting algorithmic reasoning and artificial knowledge representations. The goal of Brown et al. was to develop a realistic application for said domains and to enhance a human agent’s ability to perform in these areas, regardless of whether or not many algorithms were used within the simulation.

There are other researchers working on these kinds of simulations, like Phillips et al. (2002), who also worked on simulated knot-tying with the goal of application in the medical field in mind, or Wang et al. (2005), who implemented a virtual reality training simulation for laparoscopic surgery. Mayer et al. (2008) even developed an

AI system using recurrent neural networks to perform knot-tying for heart surgeries. While most of these are novel ideas and are of great value for their respective field of research, the work of Mayer et al. even being a cognitive AI system, they all rely on heavy use of algorithms. There has not been a knot-tying simulation focusing on the direct use of spatial or temporal structures developed yet, which is why our work is a valuable contribution to the field of (strong) spatial cognition research.

We have so far outlined the core of spatial cognition research, given an overview of related research areas and what is researched at the intersection with spatial cognition. Furthermore, we have given a glimpse of the novel field of strong spatial cognition research. We purposefully aim to make a valuable contribution to spatial cognition research, particularly to strong spatial cognition, and to explore the borders we have previously illustrated. We argue that our rope-based simulations fill a niche and are an important approach to answer research questions and tackle problems not yet or fully explored. We intend to implement simulations that each depict a specific spatial problem, offer novel and less algorithm-based ways of tackling these problems as a cognitive agent, and to outline why they are of value for future spatial cognition research in the closing chapters of this thesis. What follows next is an overview over the current state of game engines, as they are digital environments that fit very well to our approach of implementing the simulations.

2.3 Introduction to Game Engines [RAHMAN]

A game engine is a software, designed to be a framework, in which a developer can build for different platforms, such as mobile devices, desktop computers or consoles. It typically consists of several essential parts, e.g. a rendering engine, a physics engine and collision detection. The benefits from game engines can basically be derived from the fact that it offers a much simpler approach to develop games rather than doing it from scratch. They contain presets of objects and functions, so that the developer doesn’t have to figure out each parameter for every pixel in the scene. This allows even less experienced developers to produce high quality software. Most game engines allow a cross platform performance, which lifts the constraints to any certain platform. This also allows the integration of objects that have been processed in other graphic software programs, e.g in blender[3], to the game engine. To give an overview of existing game engines and to understand our criteria and the resulting choice, it is necessary to give a brief introduction to current game engines.

The existing engine options are already very competent and accommodate a fair amount of presets to build a game. Despite that, the development of an engine is rather complex and laborious. Even the allure and the advantages of a customized game engine, don’t appeal in comparison to the extent of the complexity. Therefore, it is generally recommended to avoid writing the software for an engine from scratch. Although it is definitely one of many options.To find the most suitable tool, we took a closer look at three game engines. Apart from their technical aspects, we also looked for the support by the systems community and the most intuitive workspace that would allow us to quickly pick up it's functionality and learn about new options after updates. To find a program that was not only technically strong, but also supportive in their community was important to us. Thereby we ensured to have the opportunity to discuss any on-going problems along the way and to research any functions we were unsure of using.

Besides our own criteria, there are several suggestions of significant features to look out for, when choosing an engine, on gaming forums and websites. Not only do they suggest to take in account the capabilities of your computer’s hardware, but also to look out for the compatibility of the game engine and the operating system of the computer. Furthermore there are engines that are specifically designed for particular game genres, which often allow developers to create games without the use of code - which might come in handy when the developer is fairly new or entirely inexperienced. It is also crucial to consider the targeted platform, where the game or software is supposed to be run on, otherwise the game might end up having difficulties to port on different platforms. Subsequently the skills and preferences of the developer do determine the engine choice. If the developer’s own experience is limited or outdated, it is safe to research current features and reviews give by more

[3] Blender: https://www.blender.org/download/releases/2-80/

experienced game developers. It is most likely to find recent and accurate proposition and information online, rather than in books or outdated publications.

In some cases engines have an integrated visual editor, allowing you to manipulate any objects in the engine directly. Alternatively, the engines do not come with an editor, which means the developer has to import manipulated objects from other programs. The perspective plays another great role, as only some engines are able to do both 2D and 3D. In fact the engines that are able to provide 3D features are more likely to be more complicated and turn out to have more challenging learning curves.

Before we get into a direct comparison of thee engines, it might be beneficial to understand the context of where and when game engines came into existence or when they reached their height of significance. Prior to the concept of game engines, each game had to be hard-coded as an individual entity. Engineered mostly around the display and memory constraints that inhibited elaborated designs, due to its heavy data usage. Even when platforms increased their accomodation capabilities, the games were substantially detached - which made it infeasible to create an engine that facilitates more than one game. This resulted in games, that not only were limited in their graphics data, but also small in level numbers and bound to be hard-coded. In the 1980s games and their creation began to become exceedingly popular, which promoted the release of various 2D game developing systems. These game developing systems were referred to as ‘construction kits’. Such as ‘Pinball construction set’ (1983), Adventure construction set’ (1984) or ‘Arcade game construction kit’ (1988). The first true 3D game engine was *XnGine,* released by Bethesda in 1995. However this engine suffered from stability issues and bugs on Windows 95. In the scope of this thesis we are focussing on three game engines that were derived from the above mentioned evolution.

Game engines like *CryEngine*, *Unreal* or *Unity3D* are mostly derived from a preceding game. Developed in 2002, Crytek published *CryEngine* to facilitate their first-person shooter game FarCry. Likewise Epic Games, was released the *Unreal*

Engine to facilitate the first-person shooter game Unreal in 1988.[4] *Unity*, the game engine published by Unity Technologies, however, was released as a Mac OS exclusive game engine in 2005. However all engines mentioned support multiple platforms nowadays and thus can be categorized as cross platform game engines.[5]

This chapter outlined the scope of spatial cognition research and gave insight into transdisciplinary research areas. The objects of general spatial cognition research were discussed and the relevance of spatial cognition research for other research areas highlighted. The novel research area of strong spatial cognition, developed by the CoSy group at the University of Bremen, was explicated, while the value of our work within that novel research area was stated. Additionally, the characteristics and advantages of game engines were presented, leading into a more detailed analysis of specific game engines in the following chapter. Building upon the introduction of these game engines, a comparison of them and, ultimately, the reasoning for the game engine of choice conclude the next chapter.

[4] Kissner, Michael: Writing a Game Engine from Scratch (2015), last accessed on 21.10.19: https://www.gamasutra.com/blogs/MichaelKissner/20151027/257369/Writing_a_Game_Engine_from_Scratch__Part_1_Messaging.php

[5]How Unity3D become a game development beast (2013), last accessed on 21.10.19: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/

3 Tool Selection: Game Engines [RAHMAN]

To examine the cognitive spatial problems mentioned above, it is essential to use the appropriate tools. The right tool will not only allow us to simulate the spatial task in the most effective way, but also yield the best possible performance and outcome. This chapter introduces the most commonly used game engines and works towards a comparison of them. Concluding from that comparison, the choice of the game engine used for the simulations closes this chapter

Our simulation requires a 3D environment to ensure an intuitive real-world design to solve the cognitive spatial problem by the cognitive agents. The idea is to implement a manipulable rope that acts according to our defined physical and geometrical properties, to resemble the behaviour of a cable.
As the rope needs to be flexible, the individual objects that assemble the rope are crucial. After trying out different shapes of objects, we decided to stick to capsules and spheres. This is due to the fact, that these geometrical instances ensure the highest angular motion, out of all other geometrical shapes we had on hand. Although the goal is to mimic a realistic cable behaviour, we decided to omit implementing any effects that emerge gravity. This is because the main task is to solve the actual spatial problem, which is not dependent by the force of gravity.

Nevertheless, during some initial designing of the simulation we came to know, that the implementation of gravity was in fact hindering the simulation at this point. The rope's maneuvering was heavily affected by this force, making it extremely challenging and almost impossible to form or pull through a loop. However, this ability needed to be provided, given that we are aiming to resolve a knot.

During our research we found several attempts of rope designs that were based on rectangular objects. In our opinion this shape was not very suitable. For our work we intended to go for a shape, that not only provides flexibility, but also ensures stability in our rope design. The aim was to omit gaps and breakage, without giving in to the ropes default elasticity. This is one of the reasons why we chose to go for two types of objects to instantiate the rope. Initially we undertook endeavours to design a rope solely on one geometrical shape. As a result of a rope based on sphere-shaped objects only, we observed a severe lack of stability. Similarly unsuccessful was rope based on capsules-shaped objects only, as it showed a deficiency of flexibility, derived by its shape.

Furthermore some pre-made rope sets were purchasable in the game engines asset store. However, those rope designs failed to work on certain platforms or were unable to form a knot. Oftentimes the vendors of such rope sets declared many options to manipulate the rope behaviour, raising our hopes. Unfortunately many of them lacked crucial functions, which led to their negligence in their entirety in our further approach.

One of the most promising systems we found along the way, was an asset called *Obi Rope*. This asset was ready for purchase in the asset store of Unity and provided by Obi, a virtual methods studio. Obi[6] claims to be the first CPU-based realtime physics framework that facilitates unified particle physics for Unity. At this point we have not come across a software that specifically evolved its expertise around ropes. So, with the prospect of not having to do everything from scratch, we decided to take a deeper look into the offered features. As the process of constructing a physics simulation entirely on our own might excel our efficiency and time limit, thus predetermining the quality of the final outcome .

Before purchasing we decided to research more about the advertised features and compare them with the overall experience of the community. Besides the *Obi Rope,* this provider had other virtual methods with different functions and types of particle physics, which they unified in one framework. Namely the *Obi Cloth*, *Obi Fluid*, *Obi Softbody* and lastly *Obi Rope*, which was particularly our matter of interest. This asset claims to create realistic ropes and chains, giving the developer the absolute

[6] http://obi.virtualmethodstudio.com/

control over their look. Furthermore it offers advanced editor tools, a two-way interaction with rigidbodies and claims to support all collider types. Apart from that, it features a multithreaded solver as well as extensible modular architecture and frequent updates/support. So all in all it sounded quite promising and the overall reviews from the community were positive and mostly high-rated.

The visual method *Obi Rope* creates rope geometry by generating procedural smooth mesh using splines. Furthermore it uses tangent space updating and normal map support. This comes in handy as the developer is not busy thinking about creating these physics but can concentrate on the actual design of the objects purpose. It also allows a change of rope length at runtime, as well as the design of tearable or cuttable ropes and closed loops. The integrated modular solver helps to keep performance issues at bay, as it can be specifically tuned down to suit the constraints that your rope needs. For instance, constraints for bending and per particle pin manipulation, allow an individual rope behaviour. *Obi Rope* further features an in-editor simulation preview, as well as other advanced tools such as a particle editor. This particle editor enables an easy application of tools like paintbrush, property smoothing and effortless brush selection. To further simplify the design process it allows the developer to save ropes mid-simulation and instantiate them warm-started. Although it supports all standard Unity colliders, it more importantly provides automatic camera culling, which means that non-visible ropes do not update their simulation.

Upon further examination, however, minor but essential problems started to reveal themselves. First of all, there were the problems of system crashes on mobile devices and iOS. Technically, these could have been easily neglected for our further approach. Unfortunately, it was secondly also lacking content support. Most importantly it was unable to tie a knot with the ropes provided in the framework. Although Obi mentioned updates to ensure the elimination of bugs like that, the inability to tie a knot was not solved to this date. Therefore we stopped further research in this direction, discarded the idea to purchase this asset and decided to concentrate on building a rope directly. This way we hoped to ensure that the rope met all our criteria to the best way possible.

The aim is to tackle the spatial problems with the intuitive problem solving of humans rather than the lengthy algorithmic approach used by AI robots. Therefore the object needs to be partially agent-based, so that the agents can move and select the individual objects during a simulation process. In order to guarantee a smooth simulation we chose to allow an agent-based selection only to certain objects, leaving the elements that form the actual rope not-selectable (see *Shortest Path Problem*). This allows us to give the agent a regulated control, without unnecessary and misleading selections of other objects.

To sum up, this thesis is using a game engine to create the suitable environment and the specific objects for the simulation of three spatial cognition problems.There are several game engines that are competent and notably potent for this task.

3.1 CryEngine [RAHMAN]

The first engine we want to introduce is Crytek's game engine CryEngine. This German brand published modified versions of CryEngine successfully to the game development world. The present version is CryEngine V. This version was released in 2016 and simultaneously inaugurated a licence model to access full functions. The editor in CryEngine is one of the most striking features of this engine. In comparison to editors of other companies like Unreal, it adds objects to the world space rather than eliminating existing objects from a equipped world space. Enabling the development of scenic landscapes and leaving the impressive real-world feel to each player. A impeccable graphics system featuring detailed objects of nature and weather scenarios, is what made this engine, build by Crytek, outperform other engines. Equally distinct is their licence system, that does not come with a royalty fee but a reasonable monthly fee.

CryEngine was mainly designed to make games for consoles.[7] Far Cry was the first game developed by Crytek and fostered with CryEngine1, while it was published by Ubisoft on the Microsoft Windows platform. Besides further extensions of Far Cry, they managed to make more popular games like Crysis (2011) and SNOW (2017). CryEngine also released their source code, making it a source-available commercial software.

They state on their website “With CRYENGINE, we have a simple goal: to create the most powerful game engine in the industry, and to give creators all across the globe the tools to harness this power to create world-class gaming experiences, no matter their budget or team size.”[8] Early on CryEngine has been licenced by Ubisoft, only to develop an in-house version called Dunia Engine, later in 2015 Amazon licenced CryEngine as well and modified it with several extensions running it under the name Amazon Lumberyard - free of charge.

3.2 Unreal Engine 4 [RAHMAN]

Equally considerable was the game engine provided by Epic games. This American brand, founded by Tim Sweeney, released their engine in 1998. Sweeney wrote 90 percent of the first Unreal Engine[9] and thereby introduced a powerful engine to gaming industry, almost single-handedly. In the beginning it ran off of UnrealScript. It is now written in C++ and uses Blueprint Visual Scripting technology, which allows the developer to interact within the editor in a node-based interface to create gameplay elements.[10] This enables the full intuitive potential of arranging concepts virtually, a feature that is usually reserved for programmers only. The most current

[7] Unreal Engine vs CryENGINE: Best Game Engines (2015), last accessed 14.09.19: https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose

[8]https://www.cryengine.com/features, last accessed 14.09.19

[9] An Epic Interview With Tim Sweeney (2012), last accessed 14.09.19: https://www.gamesindustry.biz/articles/2012-03-13-an-epic-interview-with-tim-sweeney

[10] https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html , last accessed 14.09.19

version, Unreal Engine 4, was released in September 2019. The engine is licenced as free, the developers, however, pay royalties as soon as the commercial revenue amount crosses a certain threshold. This licence system allows a large number of new developers to interact and develop within this engine, without having to commit or heavily invest at first. In 2014 more than 400 games were created with this engine, which led to its crowning as the most successful game engine by Guinness world records in the same year. Not only has it managed to keep its high position in the gaming industry over the years, but it also created another poster child: Fortnite. Epic is perking the cross-platform trend with Unreal Engine using Fortnite.[11] This aspect naturally engaged our interest, as we were considering to develop a cross-platform software eventually. However, it is unable to develop games for the past generation hardware, which might lead to problems in an overall coverage or a stronger focus on mobile and PC platforms. Moreover the engine allows texture and material artists to create effects from the beginning, which is useful depending on the importance of the texture in the project. As this engine is incredibly powerful visually, it usually requires equally powerful hardware to utilize the engine's full potential. So that the overall capacity and capability of the hardware used during the thesis needed to be considered, in order to avoid an overload. Another noticeable fact is that Unreal Engine 4 is limited in its compatibility with git, which is very unfortunate if it is the tool of choice as version control system.

Besides the fact that UnrealEngine has a built-in beginner solution with Blueprint Visuals, it is able to export cross-platform including consoles. Moreover it possesses outstanding next-gen graphics, good online resources and is literally free to use until the game makes profit. Yet it is heavy and demanding on performance, has a more challenging learning curve compared to Unity and also obtains a less plenitude of assets in its marketplace.

[11] Tim Sweeney wants Unreal to power the cross-platform revolution (2018), last accessed on 20.09.19:https://www.engadget.com/2018/03/21/tim-sweeney-cross-platform-fortnite-epic-unreal-engi ne-real-time-gdc/

3.3 Unity3D [RAHMAN]

Lastly, the third engine option we were to consider was Unity3D developed by Unity Technologies. Although this game engine was initially published as a Mac OS exclusive in 2005, one of its most prominent features today is its ability to work cross-platform. As a result of this, games can be easily ported onto Android, iOS and other platforms, making it a great engine for the development of mobile games. Unity3D is considered as one of the best game engines out there as the engine offers its users a wide range of tools and features that are easily accessible even if you are not as tech-savvy.[12] Unity's simplicity may be one of its greatest selling points.

In addition tools can be acquired via the asset store, a platform which provides assets and environments that are generated by the community. Furthermore Unity is approaching spatial computing. Competitors like Epic Games and their Unreal engine are focusing more heavily on desktop gaming, but an area where Unity is increasingly focusing its efforts is in the AR/VR space.[13] When it comes to making virtual or augmented reality content, there are no companies with technology as far-reaching as Unity.[14] Moreover Unity has no real modeling or building features, besides some basic shapes, inside the editor. The lack of this feature can be easily bypassed by acquiring assets from the asset library.

Despite this, it supports various file formats that are common among 3D games, thereby enabling and simplifying crucial integration. Unity3D predominantly uses C# or JavaScript, which are considered to be the preferred languages for new developers, as they are very similar. Thus, this engine preserves the hardware capacity with its light and quick interface, omitting major pressure on the CPU. Similar to the other engines, Unity3D is free of charge for personal usage. However, they introduced a license subscription model in 2016. Its current version, Unity 5,

[12] CryEngine vs Unreal vs Unity: Select the Best Game Engine (2018): https://medium.com/@thinkwik/cryengine-vs-unreal-vs-unity-select-the-best-game-engine-eaca64c60e3e

[13] With new realities to build, Unity positioned to become tech giant (2017) , last accessed on 26.019.19:https://techcrunch.com/2017/05/25/with-new-realities-to-build-unity-positioned-to-become-tech-giant/

[14] Unity Technologies: Most Innovative Company (w.D) , last accessed 27.09.19: https://www.fastcompany.com/company/unity-technologies

was released with a high number of graphical improvements, like Physically Based Shading and real-time Global Illumination besides many more. “It's evident they are joining the next-gen game engine war between UE4 and CryENGINE and with 64-bit support and WebGL Unity 5 offers some excellent features that make it a strong contender among the game engines[15]”

To sum it up, Unity has one of the richest asset stores in the gaming world. There are plenty of resources and tutorials, besides online platforms to facilitate the developer and grant a connection to the online community. This engine is also free to use until a certain revenue is reached. Nevertheless, it can be quite overwhelming for beginners and visual coding can only be instantiated by paid plugins. Although this engine is fairly popular with indie developers it’s said to be self-centred, meaning that Unity users are said to be limited with their acquired skills to this program only.

3.4 Comparison: CryEngine, UnrealEngine 4 and Unity3D [RAHMAN]

In order to compare the game engines with each other and examine how well they meet our criteria, we are forming the following categories. These categories are then ranked in ascending order of priority. Therefore the categories are divided in: learning curve, graphics, language, community, licence model and special features.
The general data and technical information are gathered objectively . However, the ranking of the criteria does depend on and states solely the personal preferences of the developer. Equally, the hardware capabilities, the goals of the project as well as the existing skills of programming are determined here.

[15] Unity Technologies: Most Innovative Company (w.D) , last accessed 27.09.19: https://www.fastcompany.com/company/unity-technologies

As it is commonly known,game engines tend to take up quite some space and can get heavy on your graphic card. Subsequently, it was crucial to undertake trial runs with our existing hardware, to make sure the game engine was not exceeding our technical competences. Followed by a rough testing of the software and its competence to execute the ideas we planned for the simulation. Fortunately all three engines provided a trial version to test on. Moreover, our team is not coming from a typical gaming background. Besides general knowledge of playing games and thus using game engines, there is no further expertise in using game engines. So we considered it to be inadequate to go for a game engine with a steep learning curve, given that we are limited in our time and resources. These factors determine the prioritising and should therefore be considered, in order to generate a positive outcome and to understand the following comparison.

Despite the fact that UnrealEngine was the first to be published in 1998, it was quickly followed by CryEngine in 2002 and Unity3D in 2005. Nevertheless, the chronological gap was quickly closed. Today all three are head to head, commonly known and popular choices in the gaming community. In order to examine them closer, we are extracting their features in the following categories mentioned before.

Learning curve

As time is a determining and limiting factor in this thesis, we are looking for an engine that allows us to adapt quickly to its tools and functions. The ideal engine would allow us to use our existing programming skills and integrate them in our new project. Unreal is discussed to have a longer learning curve – especially when there is no prior exposure or experience to game engines. Furthermore, its coding language can be another challenging factor to get into the swing of things. On the other hand CryEngine is designed as a add-on editor, which may also need more time to get used to. So, these specific points are definitely worth thinking about while looking for a tool for the thesis.

Graphics

While the Unreal Engine 4 is a game engine that provides auspicious graphics, -which allows the scenes to be realistic and detailed to each particle - ,the CryEngine has graphics similar to the capacity of Unreal Engine, but uses an add on editor.
As this thesis mainly revolves around ropes, the graphical aspect is not of particular significance and can thus be neglected. The focus is mainly on the functionality and attributes of the rope. This fact also leads to the exclusion of the other two game engines, although they were graphically advanced. Another notable factor is the cross platform integration of the Unity, that supports more than 25 platforms. Putting it way ahead of its competitors, Unreal Engine and CryEngine.

Language

There is no 'best' programming language, however, there are various qualities that differentiate a language from another. CryEngine and Unreal Engine 4 are both written and coded in C++. Unlike Unity, which uses C# and JavaScript, C++ is typically used for console applications, while C# is used for mobile, windows and console applications. Also C++ can be run on any platform, however, C# is windows specific.[16] Besides the fact that C# was derived from C++, it is also said to be easier for beginners, especially if there had been a prior exposure to Java. As this is the case in our team, it might be a pivotal point to think about.

Licence model

Unity3D has different license models, that are divided in individual and business packages. First, there is the 'free' model, which allows you to use the platform without any further payment. This model applies as long as the revenue is below $100.000/year. The 'learn premium' model requires you to subscribe and pay a monthly fee. Thereby you are mainly entitled to learn from unity certified instructors, besides other extra features. On the other hand business models run on higher expenses, while assigning more functions. 'Unity Plus' and 'Unity Pro' are both annual plans. Whereupon the plus model is limited to a revenue of $200.000/year,

[16] C++ vs C# (w.D), last accessed on 30.09.19: https://www.geeksforgeeks.org/c-vs-c-sharp/

which the pro model is entirely freed from. However, there are changes planned for the business models in the upcoming year 2020.

CryEngine announced a new 5% royalty-based model.The 5% threshold, which only kicks in after $5,000 of revenue is raised when working with the latest version of the engine, offers developer studios of all sizes the opportunity to achieve their vision and means success stories will contribute directly to making the engine better for everyone [16]. Until then you gain access by paying a monthly fee. Depending on the project size and its targeted usage it may be a more suitable option to consider.

The licence model of Unreal is based on 5% royalties as soon as the revenue is above 3.000$/quarter, however, anything below this amount will not be assessed.

Unreal Engine 4, the latest version of the game engine developed by Epic Games, is now completely free for anyone to use, the company announced today.
Epic originally launched Unreal Engine 4 in March 2014 for "early adopters" with a subscription model, charging $19 per month plus a royalty fee of 5 percent on sales. Previous iterations of the Unreal Engine had been directed at large development teams making big-budget games, with costly licensing fees associated with the technology. Epic said at the time that it wanted to open up Unreal Engine 4 to a wider audience. Unreal Engine 4 is now free to download, and all future updates will also be free. Developers of commercially released games or applications will pay Epic a 5 percent royalty on gross revenue above $3,000 per product, per quarter. [17]

Community
The community makes a great difference when tackling a new project. Especially new developers can find help and solutions to problems that occur along the way. Oftentimes the game engine offers help on their website or they provide a platform to interact with users in an open space. Besides helping beginners to overcome minor hiccups, the community has the power to actively change and influence the engine’s

[17] Epic makes Unreal Engine 4 free (2015):
https://www.polygon.com/2015/3/2/8134425/unreal-engine-4-free-epic-games

development. The companies have learned to follow trends and give solutions to bugs that the community has claimed as a problem, over the years. A quick and helpful response to questions by the community does not only show integrity of the companies, but they also benefit from these ongoing suggestions of improvement. In a way it adds to a company's reputation to pay attention and solicit discussions about current problems and question. An active community platform can be interpreted as a sign for a relevant engine and subsequently a responsible host. Out of the three engines mentioned above, Unity3D is said to have a very well-established community with great documentation.

Special features

Unlike Unity, which not only has an asset store, but also targets 27 platforms. Furthermore the store allows the developer to get more tools, sound and materials.
A characteristic feature of Unreal is its Blueprint Visuals. Its node-based interface allows an easy handling of object-oriented classes. It was introduced with the publishing of UnrealEngine 4. Cry Engine, however, is one of the few game engines that works with a add on editor, which can be viewed as a plus or a problem, depending on the developer preferences.

Considering these points of all three engines there are pros and cons for each. Nevertheless, Unity3D appears to be most adequate for this thesis. This particular game engine allows us to work in 2D and 3D. It has a wide range of tools and a large online community to discuss problems and stay uptodate, which was one of our prioritised features in an engine. The fact that it uses and runs on C# and JavaScript is another plus point to us, as we are already familiar with these coding languages.
We are not planning to exceed the yearly revenue, which requires the developer to pay a fee, with the current project. So that the factor of licensing can be neglected. Furthermore the licence free version should be able to engage to the fullest with the features featured by Unity.[18]

[18] Ranking (w.D), last accessed 06.10.19 :
https://www.gamedesigning.org/career/video-game-engines/

While it's not completely free like UE4 or Unity 5, it does not require any royalty fee, so $9.90 is all you ever have to pay to Crytek. Consequently, there is no best game engine. Choosing the right one really depends on the game being developed, the developer’s preference and requirements.

In this chapter, an overview over specific, customary game engines was given - CryEngine, Unreal Engine 4 and Unity3D. Afterwards, a detailed description of each of them followed the comparison, considering aspects like the learning curve, graphics and, most importantly, which game engine best realizes the criteria for the three simulations. The conclusion is that Unity3D is the best fit for the simulations as they are intended. In the following chapter, the whole modeling and implementation process in Unity3D is described. It includes the overall process, why we chose certain methods and ways of implementation, the criteria we met and problems that arose during the implementation.

4 Modeling and Implementation [NICKELMANN]

This chapter outlines the modeling and implementation process of the three simulations - the entangled knot, the detection of a shortest path on a route map and the strings-and-pins approach to the squaring circle problem. The main tool we used for the simulations, Unity3D, is described in Chapter 3 of this thesis. In this chapter, the modeling tool we first wanted to use for constructing the basic rope model, *Blender*, is explicated. Subsequently, the vital component of all three simulations, the Joint component of Unity3D, is elucidated, as well as how it serves to model the basic ropes. Proceeding from this foundation, the three simulations are illustrated in three consecutive subchapters. For each simulation, the implementation process, the final implemented behaviour, the goals achieved and the problems encountered are described.

4.1 First Attempt at Modeling the Rope: Blender [NICKELMANN]

Blender [19] [20] is a 3D modeling program providing the functionality for all steps of the 3D pipeline - modeling, animation, rendering et cetera. The key animation feature of Blender is the bone structure. A skeleton is constructed for each 3D model which defines the object's movement via weight matrices. They control the influence of each joint on certain mesh vertices. This process is called *Rigging and Skinning*. A model can then be imported into Unity3D, having its movement behaviour already implemented. In Unity3D, only small adjustments are then necessary, besides the implementation of the specific simulation setups. The reason for using Blender to model the rope is that the positions of the mesh vertices are calculated at run-time, so less memory is needed for the rope movement. Additionally, there are more possibilities for deforming the rope, which is crucial for the simulation[21]. It also allows

[19] blender.org
[20] Editing Bones - Blender Manual. (w.D.). Accessed on October 3rd 2019, at: https://docs.blender.org/manual/en/latest/animation/armatures/bones/editing/bones.html
[21] Blender 3D: Noob to Pro/Bones. (w.D.). Accessed on October 3rd 2019, at: https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Bones

us to use any model we want for the rope, as the underlying bone structure turns invisible during rendering. Admittedly, the run-time movement evaluation necessitates more calculations being made - as we want to implement simulations in line with the goals of strong spatial cognition research (see Chapter 2.2), this is a poor tradeoff. Higher memory consumption is less of a contradiction to these goals than more computations are, which is the main factor to be avoided. Besides, the import into Unity3D did not work properly, as the rope deformed in ways it should not have - the part above each bone rolled up and formed a half-loop (see Image 1).

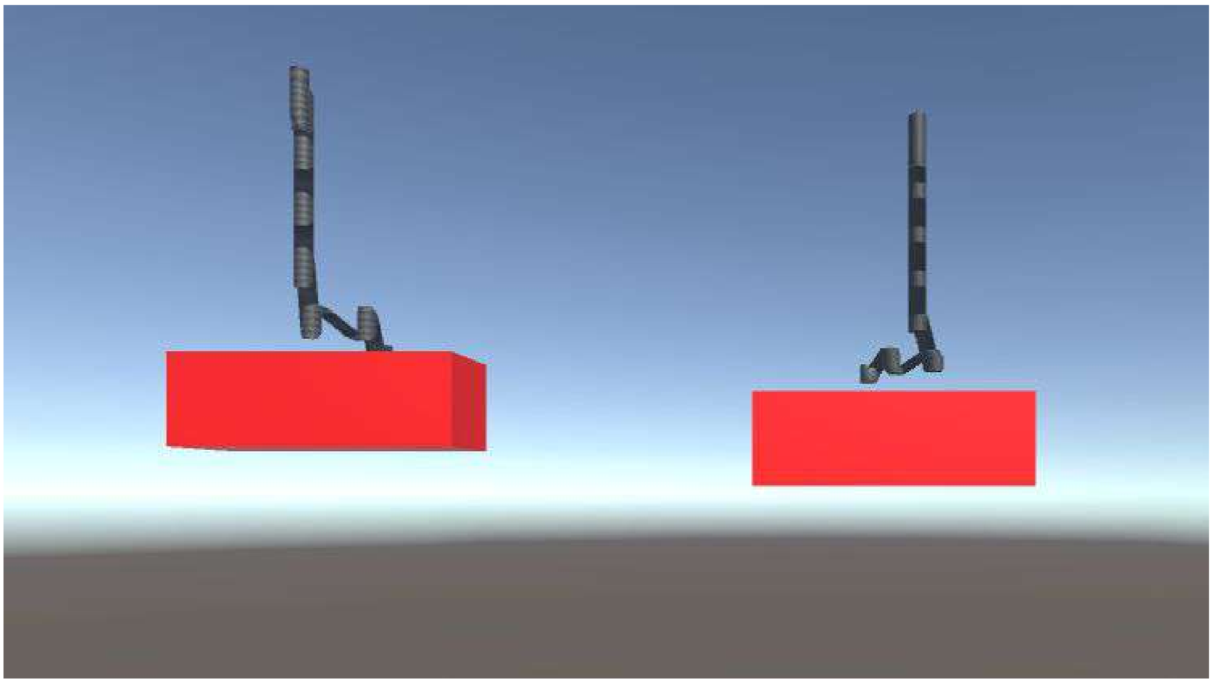


Image 1: On collision, the rope models imported from Blender behaved strangely, deforming the mesh to look like a half-loop.

As we encountered the *Joint component* in Unity3D around that time and had not fixed the import problems yet, we decided to abandon this approach and model the rope as well as implement its behaviour in Unity3D. The game engine provides all the features we need to fulfil the criteria, the Joint component being the key to the implementation of the three simulations. Using the Joint component instead of the bone structure of Blender also reduces the amount of computations necessary, as the object meshes need not be deformed for the object's motion.

4.2 The Joint Component in Unity and Joint Types [NICKELMANN]

"[The] Joint component connects a Rigidbody to another Rigidbody or a fixed point in space"[22]. They apply forces to the Rigidbodies that move them along, or rotate them around, each of the three axes in 3D space, as well as limit their movement. Their properties are adjusted easily via the Inspector window of Unity3D, and can also be accessed in a script via *GameObject.GetComponent<Joint>().property* (where *GameObject* is the game object the Joint component is attached to, and *property* is a placeholder for the Joint property to be changed, like *connectedBody* or *breakForce*).

In Unity3D, there are five different types of Joints. They cause different object motion behaviour by exerting different forces and limiting the movement more or less than other Joints. What follows is a short description of the different Joints accessible in Unity3D to explain why we used the *ConfigurableJoint* in the end. Note that the evaluation of each Joint type is based on its usefulness for the Entangled Knot Simulation, as it was the first simulation we implemented, intending to use the underlying rope model for the other two simulations.

The **Fixed Joint**[23] is used to make the connected Rigidbody directly follow the movement of the Rigidbody the Joint is attached to, akin to parenting in an object hierarchy. This is not useful for the intended simulation, as, for example, moving any link of the rope upwards would lead to each other link following the upwards movement - there would be no rotation possible. Likewise, rotating a link of the rope would lead to each other link rotating accordingly. The rope would always be in a straight line if the links were connected via Fixed Joints.

[22] Unity - Manual: Joints. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/Joints.html
[23] Unity - Manual: Fixed Joint. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/class-FixedJoint.html

The **Spring Joint**[24] is useful if you want the connecting Joint between two Rigidbodies to be elastic. The game object the Spring Joint is attached to oscillates if a force is applied to the connected Rigidbody. A damper can be used to slow down this oscillation. This is also not useful for the intended simulations, as we do not want any link of the rope to oscillate at all; we want the link to be flexible, but also always connected to each other gapless.

The **Hinge Joint**[25] rotates the object it is attached to at a predefined anchor point in local space around a predefined axis, like an actual hinge. As you can only choose one axis to rotate around, the Hinge Joint is not fit for the intended simulations. For entangling and detangling a knot in 3D space, rotation around each of the three axes is necessary. The rope should be able to be pulled and pushed in any direction, at any part of it, which naturally leads to it being rotated around more than a single axis.

The **Character Joint**[26] combines the Hinge and the Spring Joint - there is a swing axis around which the joint swings when a force is applied to the connected Rigidbody. Similarly, there is a twist axis, which the joint is rotated around when a force is applied. The Character Joint is not used in the simulations as it does not allow for the velocity of the Joints to be changed. This is necessary as the links of the rope should have the maximum velocity possible to reach the target position as fast as possible. In other words, the links need to follow each other immediately. However, when the rope is not being manipulated, the velocity should immediately be set to zero. This is only possible with the Configurable Joint.

[24] Unity - Manual: Spring Joint. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/class-SpringJoint.html
[25] Unity - Manual: Hinge Joint. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/class-HingeJoint.html
[26] Unity - Manual: Character Joint. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/class-CharacterJoint.html

The **Configurable Joint**[27] combines the functionalities of all other Joints and thus is the most customizable of the Joints. It is used in the simulations as you can precisely define or lock movement along and rotation around all axes, and also change the velocity of the rope links. The velocity definitely needed to be changeable as movement and rotation need to stop immediately after pulling or pushing the rope, while being as high as possible during the rope manipulation. Combined with its very high customizability, we decided to use Configurable Joints for the rope model.

4.3 Constructing the Basic Rope and Implementing the Entangled Knot Simulation in Unity3D [NICKELMANN]

The goal for constructing the rope is to create the different rope parts from a script while setting as many properties as possible via prefabs. Creating the parts from a script saves a lot of time because you do not have to place each object, adjust its properties and link its Joint to the corresponding Rigidbody. This also provides the option to adjust the length of the rope easily.

First, we tested the behaviour of the Configurable Joints on a rope we manually created. As this was our first time using the Joint component of Unity3D, we had some problems initially. In order to best describe the explorative process of creating the rope and to help people avoiding these mistakes if they want to work on these or similar simulations, we depict the problems we encountered as detailed as possible. Furthermore, we used two different object types for the rope - spheres and capsules - as we thought mimicking the bone structure used by Blender would be beneficial for the implementation. In the end, the choice of objects makes no difference as long as the colliders are set up properly. For this reason, we recommend using objects with round ends connected to each other, as to not collide with neighbouring objects during motion or rotation.

[27] Unity - Manual: Configurable Joint. (w.D.). Accessed on October 4th 2019, at: https://docs.unity3d.com/Manual/class-ConfigurableJoint.html

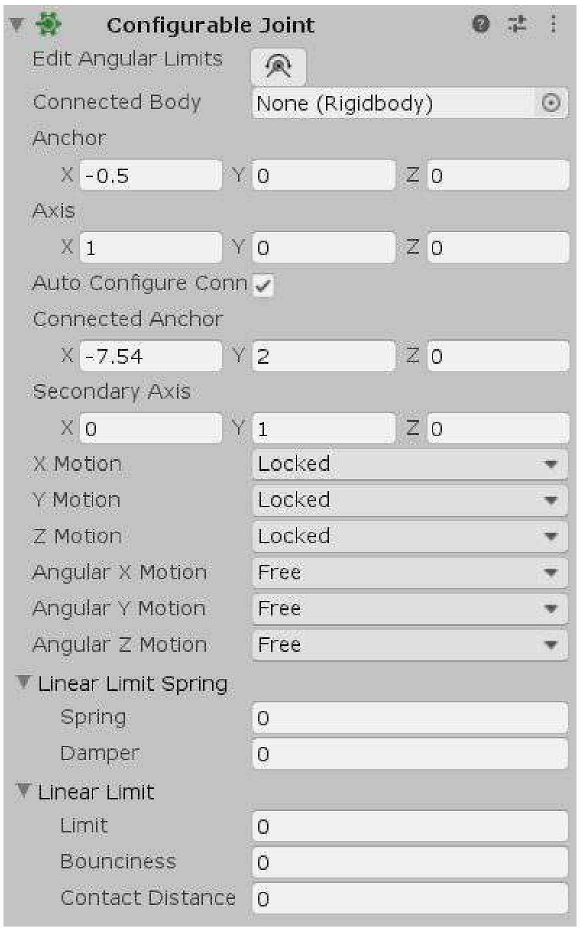
Furthermore we decided to omit implementing any effects that emerge gravity. This goes by the explanation that the main task is to solve the actual spatial problem, which is not dependent on the force of gravity. During the initial designing of the rope, we came to know that the implementation of gravity was in fact hindering the simulation at this point. The rope’s maneuvering was heavily affected by this force, making it extremely challenging/ impossible to form or pull through a loop, and thus, testing the rope behaviour. However, this ability needed to be provided (which it is in Unity3D), given that we are aiming to resolve a knot in 3D space in a realistic manner.

We started by creating multiple spheres and capsules seamlessly connected to each other, with the first and last object being a sphere. Each object was given a Collider adjusted to its size, a Rigidbody and a Configurable Joint. After scaling, moving and rotating the objects into a line, we began testing how the different Joint properties worked in order to create the base rope model. As Unity3D provides you with the option to move objects along or rotate them around one of the three axes in the Scene view, which is enough to fit our criteria, we decided not to implement any additional control options for users. The first mistake we made was giving each rope object two Joints instead of one, except for the first and last sphere, which we both gave one Joint. We thought that a Joint only affects one of the links it connects, and only affects it if the corresponding connected body is moved or rotated (the connection between two objects, thus, needing two joints). We quickly discovered this to be a problem as there was heavy jittering when colliders were enabled (which was necessary so the rope would not move through itself). There was additional unwanted behaviour of the rope, some parts were moving when they should not have, and vice versa. At that point, we realized that we needed to remove duplicate joints.

The next problem was that the movement was very slow and still caused jittering. This was because we did not set any link to be kinematic, so that it would affect the physics of each connected object and likewise would not be affected by them. As we were still in the testing phase and did not write any scripts, we decided to make the leftmost sphere kinematic and test the behaviour of the rope by manipulating that sphere (if we wanted, we could quickly make other spheres kinematic). It is important, however, that only one link is kinematic at one time. Otherwise, the rope would be stretched between two kinematic objects as those objects only move when they are directly pulled or pushed by a user. In the end, we want to write a script that always switches the kinematic object to be the one currently manipulated by the user.

At that point, jittering was still a problem, but at least the rope moved as fast as it should. However, the movement did not stop after pushing or pulling has been completed. We decided to write our first script, called *Movement* (see *Appendix, Image A1* for the complete final script). Its only purpose is to set the velocity and angularVelocity of the rigidbodies to zero every frame. This does not inhibit the user from moving or rotating the rope. The movement resulting from the user manipulating objects in the Scene view directly via the given tools is caused by continuous force application; resetting it merely prevents it from going above a certain limit and the object from moving after the user has stopped moving it. This is necessary to do every frame as the user might stop moving or rotating the rope at any time, and the rope should not continue moving or rotating afterwards.
The rope finally behaved as we wanted it to in regards to movement, but the jittering was still conspicuous.

We thought that this would be a good time to start scripting in general, as we could not stop the jittering with just the Joint properties displayed in the inspector window. We created our main script *CreateLinks* (see *Appendix, Image A2* to *A5* for the complete final script). Its purpose is to seamlessly create the sphere and capsule links, connect them properly via Configurable Joints, and attach other scripts to them. We also converted the spheres and capsules into prefabs, with an additional prefab for the first sphere as it has no Joint. With that, we only need to adjust two Configurable Joints, one for all the spheres except the first one and one for all capsules.


Configurable Joint

Edit Angular Limits	
Connected Body	None (Rigidbody)
Anchor	X -0.5 Y 0 Z 0
Axis	X 1 Y 0 Z 0
Auto Configure Conn	✓
Connected Anchor	X -7.54 Y 2 Z 0
Secondary Axis	X 0 Y 1 Z 0
X Motion	Locked
Y Motion	Locked
Z Motion	Locked
Angular X Motion	Free
Angular Y Motion	Free
Angular Z Motion	Free
Linear Limit Spring	
Spring	0
Damper	0
Linear Limit	
Limit	0
Bounciness	0
Contact Distance	0

Image 2: The Inspector window for the Configurable Joint component.

In order to attain the desirable motion behaviour for the rope, you first have to lock the X, Y and Z motion, as otherwise, moving a single object would not affect any other object at all. The Angular X, Y and Z motion need to stay free as the rope would remain completely stiff otherwise. After that, you have to reset the anchor points around each Joint is oriented. For the sphere's Joints, this is x = -0.5 to place it on its left side, for the capsules it is y = 1 as they are rotated by 90 degrees around the Z axis. By changing these Joint attributes on the prefabs, the motion behaviour is already very close to fulfil the criteria for the simulation.

The script CreateLinks instantiates prefabs of the spheres and capsules alternatingly, the first sphere not being connected to any rigidbody but the other objects being connected to the rigidbody of the previously instantiated object via Joints. Afterwards, each object is assigned the Movement script and is set to not being kinematic (except for the first sphere, again). The last object to be instantiated is again a sphere. As the instantiation happens as a loop, users can easily change the number of loops in the script to change the number of links, and by that the overall length, of the rope.

At this point, the only problem left was the jittering of the rope. After some research on the internet[28] [29] [30] [31], we discovered the property *inertiaTensor*. In Unity3D, it is a

[28] What is inertia Tensor for Dummies? (2014). Accessed on October 27th at:

Vector3 that describes how much torque is needed to rotate an object around an axis (torque being equivalent to the force you need to move an object, but for rotation). The higher the values, the harder it gets to rotate an object around a certain axis. If the inertiaTensor is not set manually, it is “calculated automatically from all colliders attached to the rigidbody” [32]. After some experimentation on whether this could stop the jittering, we discovered that we needed to set the values way higher than those that were automatically calculated. This stopped the jittering completely, which in hindsight was probably caused by the links rotating too quickly but being stopped each frame. This was mitigated by setting the inertiaTensor via script to values that make it harder for objects to rotate (the value was set in the CreateLinks script during the instantiation).This does not inhibit the user from rotating any object. If anything, the overall manipulation of the rope is working smooth and as intended to fit the criteria for the simulation.

Next, we wanted to enable pulling and pushing the rope at any part of it, not just the first sphere. To achieve this, you have to make any object that the user currently manipulates kinematic and each other object not kinematic. In the end, we created the *KinematicController* script (see *Appendix, Image A6* to *A7* for the complete final script), which sets the first sphere instantiated as the active (kinematic) sphere immediately after instantiation. In the next step, the object currently being manipulated by the user is assigned as the active object and made kinematic via the assignment of *Selection.activeGameObject* to a placeholder. The previously active object (which was saved in a second placeholder after it was made active) is made non-kinematic. The newly activated object is likewise saved in the placeholder for the previously activated object. The script now works and enables users to pull at and push every part of the rope along the three axes with the motion tools contained in

https://forum.unity.com/threads/what-is-inertia-tensor-for-dummies.235919/
[29] Calculus 3: Tensors (13 of 45) What is the Inertia Tensor? (2018). Accessed on October 27th at: https://www.youtube.com/watch?v=Ch-VTxTIt0E
[30] Moment of Inertia Tensor (2011). Accessed on October 27th at: http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node64.html
[31] Unity - Joint tearing under larger force (2015). Accessed on October 27th at: https://gamedev.stackexchange.com/questions/99245/unity-joint-tearing-under-larger-force
[32] Unity - Scripting API: Rigidbody.inertiaTensor. (w.D.). Accessed on October 27th, at: https://docs.unity3d.com/ScriptReference/Rigidbody-inertiaTensor.html

Unity3D. The rope is now behaving exactly as we wanted it to, meeting every criteria we set in the beginning of our work - almost.

There is one problem we have not been able to fix, and that no other programmer has seemed to fix yet, likely because the situation it appears in is not of interest to many people. The problem is that the individual rope objects pull slightly apart from one another. In most simulations, this is not an issue as no other object should be exactly at that gap and be small enough to squeeze through it with enough force. Unfortunately, this is the case when you try to tie a knot. If you pull a knot too tight, one object will eventually slip through a gap where the knot is tied, breaking it. The elasticity of a Spring Joint, which is also inherent to the Configurable Joint, cannot be fully disabled. Intuitively, we tried to change the values for the Spring and Damper values of the Joint. However, setting the Damper value very high and the Spring value to zero did not solve the problem. We also tested other velocities for the objects or changing their force but still, the rope objects slipped through the gaps, breaking the knot that way. We also tried other solutions like increasing the solver iteration count in the project settings, changing the time step or increasing the rigidbodies' drag, but despite the knot being more stable, none worked out. The problem is, regrettably, not solvable to our knowledge, inhibiting the simulation to fulfil its intended purposes[33] [34].

As time has become a concerning factor at that point, we decided to move on and implement the other two simulations. The rope objects pulling slightly apart from each other is not a problem for the shortest path simulation or the squaring circle simulation as the objects will not be put tightly together in one place with great force. We would have liked to solve the problem and have a fully functioning entangled knot simulation, however, the other two simulations became our priority considering the time frame left. They will be explicated in the following two subchapters.

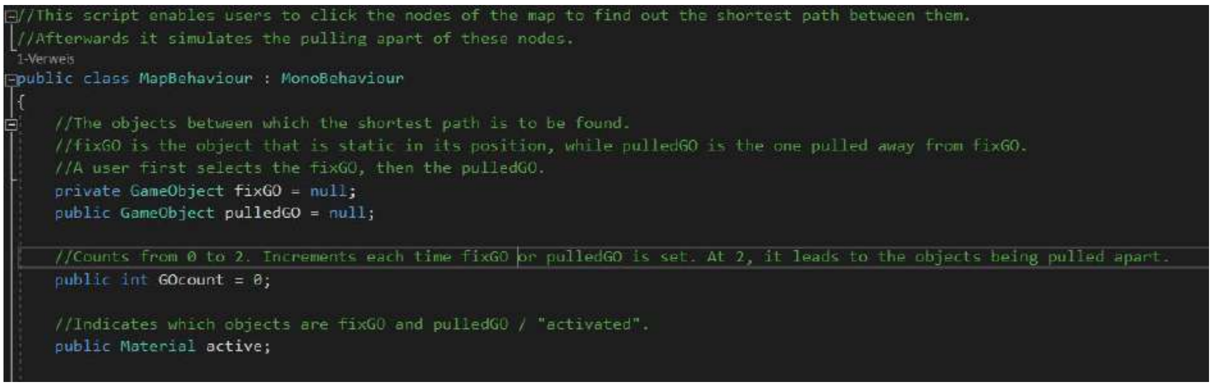
[33] Non-Springy Configurable Joint (2010). Accessed on November 30th at: https://answers.unity.com/questions/14358/non-springy-configurable-joint.html
[34] Joints sag with mass? What is the cause? Can it be eliminated? (2019). Accessed on November 30th at: https://www.reddit.com/r/Unity3D/comments/dv3byh/joints_sag_with_mass_what_is_the_cause_can_it_be/

4.4 Building the Shortest Path Simulation [NICKELMANN]

Besides the entangled knot simulation, we also wanted to implement a map simulation based on the shortest path problem (Freksa, 2015).

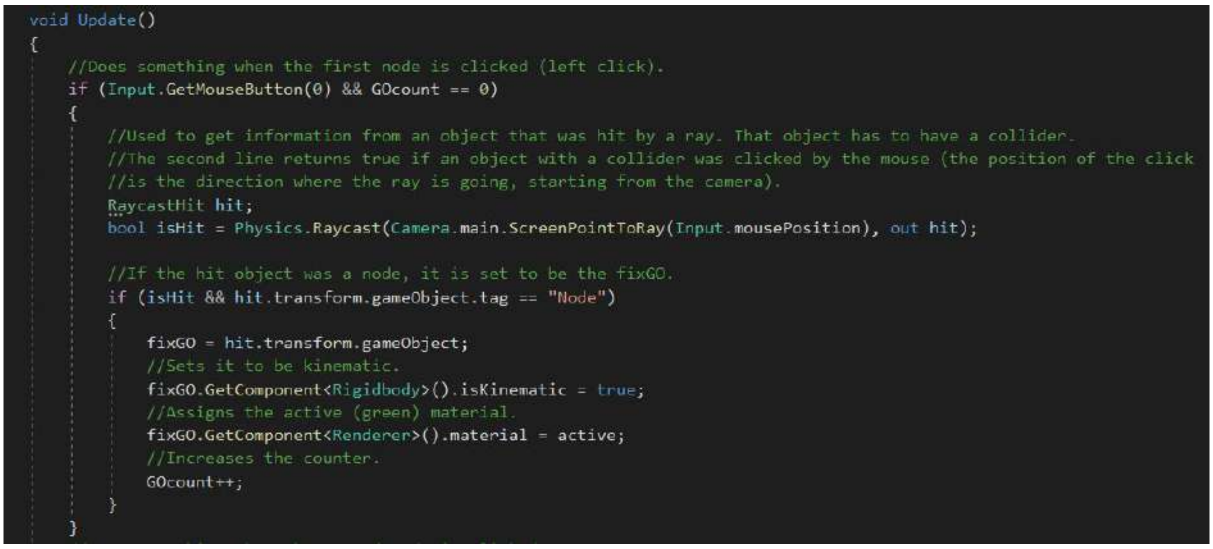
First, we planned how to construct the route network. We used bigger, red spheres as the waypoints/nodes between which a shortest path can be detected. The routes are made of the spheres and capsules we used for the entangled knot simulation. As spheres and capsules are alternating in this simulation as well, but the node spheres are connected to multiple capsules, we decided to give each capsule two joints to connect them to each adjacent sphere.

The only script from the entangled knot simulation we could use was the Movement script which limits the velocity of the links. We could not use the CreateLinks script as we created the route network manually, and the kinematic property is handled differently in this simulation, so we also could not use the KinematicController script.



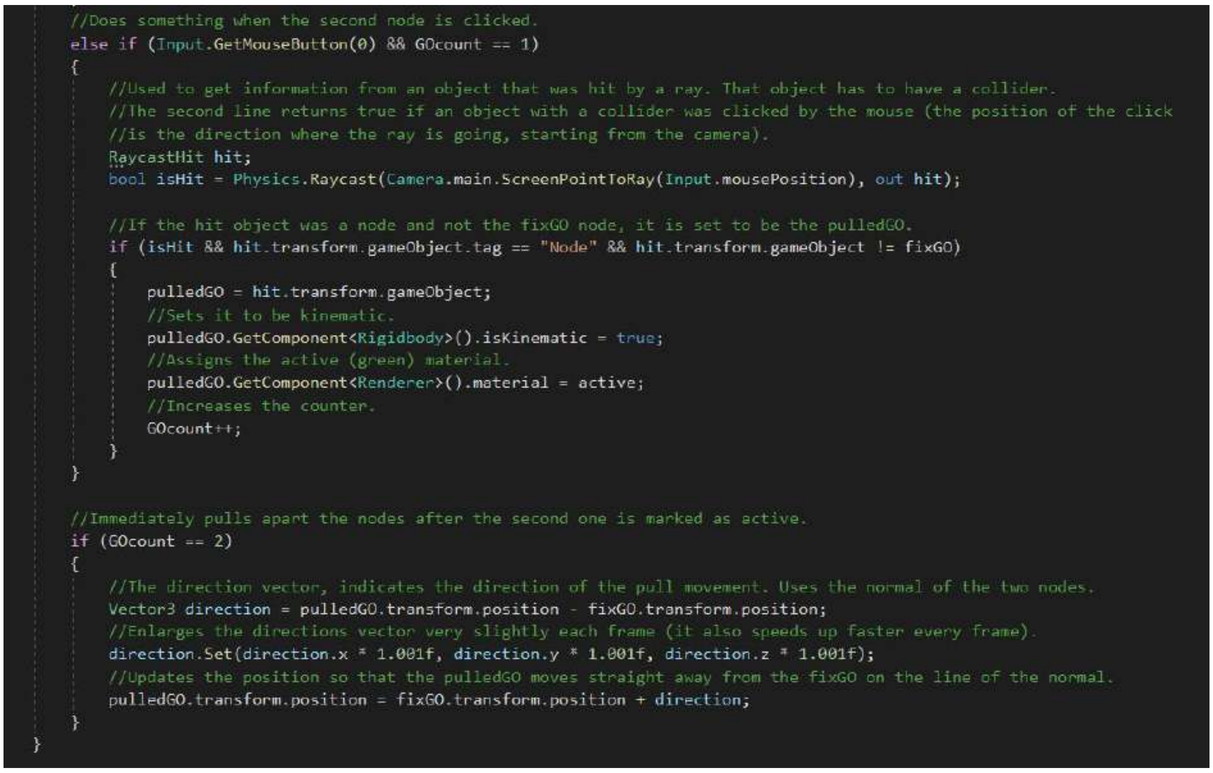
```
//This script enables users to click the nodes of the map to find out the shortest path between them.
//Afterwards it simulates the pulling apart of these nodes.
1-Verweis
public class MapBehaviour : MonoBehaviour
{
    //The objects between which the shortest path is to be found.
    //fixGO is the object that is static in its position, while pulledGO is the one pulled away from fixGO.
    //A user first selects the fixGO, then the pulledGO.
    private GameObject fixGO = null;
    public GameObject pulledGO = null;

    //Counts from 0 to 2. Increments each time fixGO or pulledGO is set. At 2, it leads to the objects being pulled apart.
    public int GOcount = 0;

    //Indicates which objects are fixGO and pulledGO / "activated".
    public Material active;
```

Code 1: The variables of the MapBehaviour script (the starting and ending point game objects of the path, a counter, a material to indicate the selected starting and ending point).



```
void Update()
{
    //Does something when the first node is clicked (left click).
    if (Input.GetMouseButton(0) && GOcount == 0)
    {
        //Used to get information from an object that was hit by a ray. That object has to have a collider.
        //The second line returns true if an object with a collider was clicked by the mouse (the position of the click
        //is the direction where the ray is going, starting from the camera).
        RaycastHit hit;
        bool isHit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit);

        //If the hit object was a node, it is set to be the fixGO.
        if (isHit && hit.transform.gameObject.tag == "Node")
        {
            fixGO = hit.transform.gameObject;
            //Sets it to be kinematic.
            fixGO.GetComponent<Rigidbody>().isKinematic = true;
            //Assigns the active (green) material.
            fixGO.GetComponent<Renderer>().material = active;
            //Increases the counter.
            GOcount++;
        }
    }
```

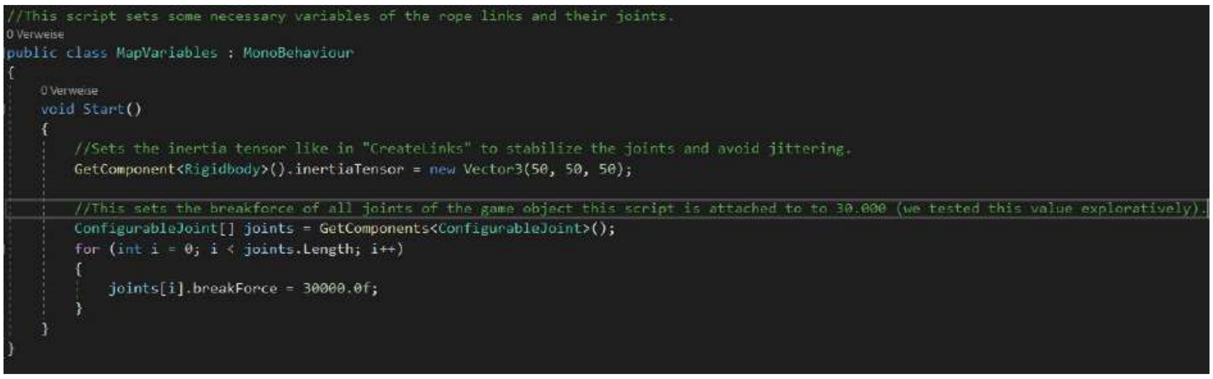
Code 2: Start of the Update() method. This allows for selecting the starting point of the path, which is the object that is fixed in position.



```
    //Does something when the second node is clicked.
    else if (Input.GetMouseButton(0) && GOcount == 1)
    {
        //Used to get information from an object that was hit by a ray. That object has to have a collider.
        //The second line returns true if an object with a collider was clicked by the mouse (the position of the click
        //is the direction where the ray is going, starting from the camera).
        RaycastHit hit;
        bool isHit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out hit);

        //If the hit object was a node and not the fixGO node, it is set to be the pulledGO.
        if (isHit && hit.transform.gameObject.tag == "Node" && hit.transform.gameObject != fixGO)
        {
            pulledGO = hit.transform.gameObject;
            //Sets it to be kinematic.
            pulledGO.GetComponent<Rigidbody>().isKinematic = true;
            //Assigns the active (green) material.
            pulledGO.GetComponent<Renderer>().material = active;
            //Increases the counter.
            GOcount++;
        }
    }

    //Immediately pulls apart the nodes after the second one is marked as active.
    if (GOcount == 2)
    {
        //The direction vector, indicates the direction of the pull movement. Uses the normal of the two nodes.
        Vector3 direction = pulledGO.transform.position - fixGO.transform.position;
        //Enlarges the directions vector very slightly each frame (it also speeds up faster every frame).
        direction.Set(direction.x * 1.001f, direction.y * 1.001f, direction.z * 1.001f);
        //Updates the position so that the pulledGO moves straight away from the fixGO on the line of the normal.
        pulledGO.transform.position = fixGO.transform.position + direction;
    }
}
```

Code 3: The rest of the Update() method, which allows for selecting the ending point of the path and simulates the pulling apart of the map.

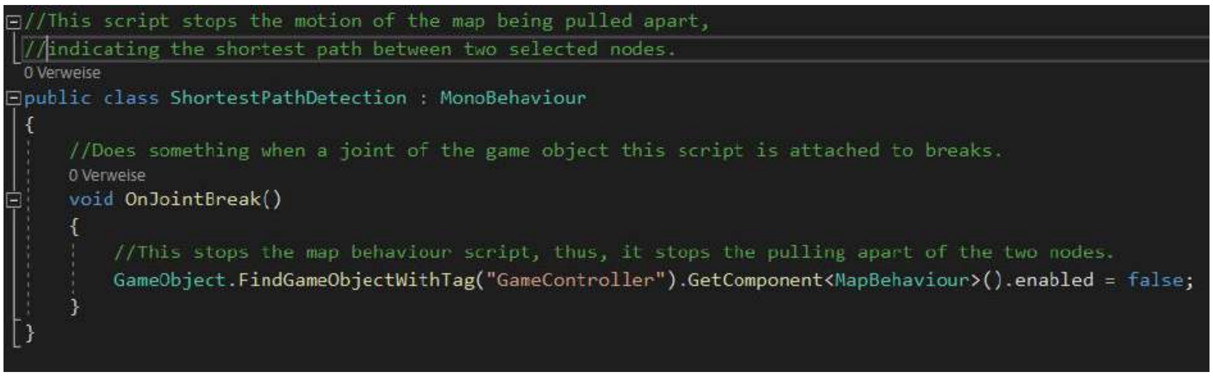
For the kinematic property, we wrote a script called *MapBehaviour* which is attached to a game controller object. Its purpose is to pin down one of the node spheres (see Code 2) and pull a different one straight away from it (see Code 3), which is done by simply clicking the nodes (the first one being fixed in its position, the second one being pulled away from it in a straight line). Between those nodes, the shortest path will be detected automatically. The two selected nodes are made kinematic as they should not be influenced by the movement of other links of the route network. They are marked by being assigned a green material.



```
//This script sets some necessary variables of the rope links and their joints.
0 Verweise
public class MapVariables : MonoBehaviour
{
    0 Verweise
    void Start()
    {
        //Sets the inertia tensor like in "CreateLinks" to stabilize the joints and avoid jittering.
        GetComponent<Rigidbody>().inertiaTensor = new Vector3(50, 50, 50);

        //This sets the breakforce of all joints of the game object this script is attached to to 30.000 (we tested this value exploratively).
        ConfigurableJoint[] joints = GetComponents<ConfigurableJoint>();
        for (int i = 0; i < joints.Length; i++)
        {
            joints[i].breakForce = 30000.0f;
        }
    }
}
```

Code 4: The MapVariables script for game object and joint properties.

In addition to the MapBehaviour script, we wrote the *MapVariables* script attached to each object which sets each Joint's break force and each object's inertia. We had to experiment a little bit to find out the optimal value for the break force, in the end, we settled at 30.000. If this value is exceeded, or in other words, the Joints are pulled apart by a force greater than 30.000, the Joint breaks, indicating the shortest path between the two selected nodes.



```
//This script stops the motion of the map being pulled apart,
//indicating the shortest path between two selected nodes.
0 Verweise
public class ShortestPathDetection : MonoBehaviour
{
    //Does something when a joint of the game object this script is attached to breaks.
    0 Verweise
    void OnJointBreak()
    {
        //This stops the map behaviour script, thus, it stops the pulling apart of the two nodes.
        GameObject.FindGameObjectWithTag("GameController").GetComponent<MapBehaviour>().enabled = false;
    }
}
```

Code 5: The ShortestPathDetection script for stopping the motion after a joint breaks.

The last script we wrote was the *ShortestPathDetection* script attached to each capsule, as they hold the Joints. When a Joint breaks, the function *OnJointBreak()* is called, disabling the MapBehaviour script and stopping the traction of the previously pulled node.

As we had already figured out the most difficult steps for implementing a simulation based on the specific rope properties and rope behaviour we wanted to use, implementing the shortest path simulation was not that difficult. What made it even easier for us was the fact that this simulation is basically a 2D simulation, and besides user input for selecting the two waypoints, each step is carried out by the scripts and not the user.

4.5 Implementing the Squaring Circle Simulation [NICKELMANN]

In addition to the entangled knot simulation and the shortest path simulation, we want to implement a simulation based on the strings-and-pins approach to the squaring circle problem explicated in Chapter 1.1.2.

We have implemented the simulation up to the point where the circumference of the circle is marked and the string should be unrolled, as this is the part which cannot be solved by a straightedge-and-compass approach. A central script, called *SquaringCircles* (see *Appendix, Image A8* to *A15* for the complete final script), directs each step of the squaring circle process. There are different materials for the cylinder, the spheres, pinned spheres and the spheres that are part of the circumference. This is done in order to discern the pins and the different parts of the string - the radius, the circumference et cetera. One array lists all spheres, another one all pinned spheres; this is important for counting the pinned spheres as a certain number of pinned spheres starts a certain part of the process. Two additional arrays hold all spheres beyond those inside the cylinder and all spheres of the circumference respectively.

The spheres are instantiated like they are in the entangled knot simulation. The only difference, besides different scripts being attached, is that the method *CheckBounds()* is called during instantiation. It checks whether or not a sphere is created inside the cylinder (the first one is always instantiated at the center of the cylinder) and if it is, assigns it to a layer which ignores collision with the sphere.

A user has to follow certain steps during the simulation of the squaring circle process, or else it will not work properly. The first step is to pin the sphere at the center of the cylinder (pinning works like in the shortest path simulation, besides the object's positions being frozen), and afterwards, they have to pin the first sphere directly outside the cylinder. The script checks the number of pinned spheres every frame. If that number is two, the part of the rope behind the sphere pinned directly outside the cylinder is rotated counterclockwise around the cylinder. This is done to measure the cylinder's circumference.

At this point, a script called *SCCollision* (see *Appendix, Image A16* for the complete final script) comes into play, which is attached to each sphere. It checks for collision with the sphere pinned directly outside the cylinder, for as long as there are exactly two pinned spheres. A collision with that sphere happens when the circulation is finished, and when detected, the colliding sphere gets pinned as well (thus, increasing the number of pinned spheres to three). All spheres that are part of the circumference are indicated via an orange material.

As mentioned before, obtaining the circle's/cylinder's circumference and being able to transform it into a straight line afterwards is the vital component of the simulation, as it cannot be achieved by the straightedge-and-compass approach. This has been implemented and it works as intended. We tried to implement further steps in order to show the complete task of squaring the circle, but due to time constraints, we did not get the next to work (which would be rolling back the string and pulling it out into a straight line). We definitely would have liked to fully implement the simulation, but within the frame of this thesis, it was neither possible nor greatly necessary. Nevertheless, we hope that this simulation is of value for further research of strong spatial cognition topics and will be continued to be worked on by other programmers.

In this chapter, we have presented the whole modeling and implementation process for the three simulations. We have illustrated each separate step of the process, starting with the endeavour to model the rope in Blender and importing it to Unity3D. After explaining why this approach was abandoned, we explicated the key to the implementation via Unity3D: The Joint component. There are five different types of Joints available, which we all briefly described, and we reasoned why the Configurable Joint is most suited for the intended simulations. Thereafter, we started the report on the implementation process. We discussed how we implemented the basic rope, which problems we encountered and which criteria the rope met. After configuring the basic rope, the most difficult work was done and we could focus on the setup of the simulations. Unfortunately, due to the individual rope parts pulling apart, tying a knot is not possible and we could not get a satisfying end result of the entangled knot simulation. However, the rope parts pulling apart is not a big problem for the other two simulations. We illustrated how we set up the map simulation and which behaviour needed to be implemented in order for it to work properly. In the end, the map simulation is working as intended and its functionalities can be applied to any map constructed equally to our example map. The squaring circle simulation and its methods were delineated in the end of this chapter. The criteria we wanted to meet and the most important step in the squaring circle process could be implemented, and thus, the end result of that simulation is satisfying. The end results of the three simulations will be depicted and discussed in the following chapter.

5 Results [NICKELMANN + RAHMAN]

In this chapter we are presenting our outcomes of the thesis. The examination of the spatial cognitive problems is following the order in which we proceeded to implement them. The results to the entangled knot problem will be displayed first, followed by the shortest path problem and ceased by the squaring circle problem. This is because the problems are considerably equal in their relevance, so that we do not need to distinguish the results based on that. Despite this fact, the simulations we would like to present have similar substructures. Thereby, some outcomes of prior examined problems were already incorporated in the simulation of the next problem. In this case, the entangled knot problem and the shortest path problem share their rope substructure, the latter problem inherits the rope design from the former problem.

Due to the fact that this thesis is following an explorative viewpoint, the results will not only be presented as the final simulations. Additionally, we are documenting the different stages of the simulations and the design process. These can be found in the documentation chapter, as well as visualised in a demovideo and are of equal importance as the end results.The length of the demovideo is one minute and fifty seconds and shows a shortened screenshot video, which was directly captured from the screen. Furthermore the figures in this chapter are mainly taken from this video to illustrate our work in this setting. Moreover, that chapter will incorporate all important solutions, that were considered or dismissed. The description of the design process might be of interest for further works. Thereby presenting the thought-process as well as the milestones that lead up to the final outcome.

The spatial cognition problems we are featuring are from different levels of difficulty. We like to note at this point, that in this thesis, the difficulty level is measured by the total of the different components. For example, a simulation that requires more and complex components is the squaring circle problem, whereas the simulation of the shortest path problem is depending on fewer components to be solved. Thus we are presenting different states of simulations, according to the needs of the spatial problem. Some states are solving the entire spatial problem requiring a complete simulation, while others only need partial simulation to be solved, before they e.g. receive further geometrical treatment.

5.1 Entangled Knot Simulation [RAHMAN]

As this spatial cognitive problem evolves around a knot, the aim of the simulation was the detangling of a knot. The process of detangling required several movement concepts and features, that needed to be implemented. For this reason we focussed on designing a rope that can be manipulated by a cognitive agent. Main mobility features to detangle the knot, like *pulling*, *pushing* or *dragging* the rope, were our highest priorities and are further described in our methods. This simulation consists of two alternating objects, one shaped as a capsule and the other shaped as sphere (see figure 1). The objects, except for the first sphere, are connected by configurable joints to form the manipulable rope.

In this simulation we designed the first sphere to be slightly different from the other spheres, which is now designed to be followed. As seen from figure 1, in the beginning of the simulation a rope was created and arranged horizontally. Furthermore a set amount of objects were instantiated, to resemble the characteristics of a middle length cable. Thereby referring to the experiment of detangling headphone cables, which inspired this simulation.

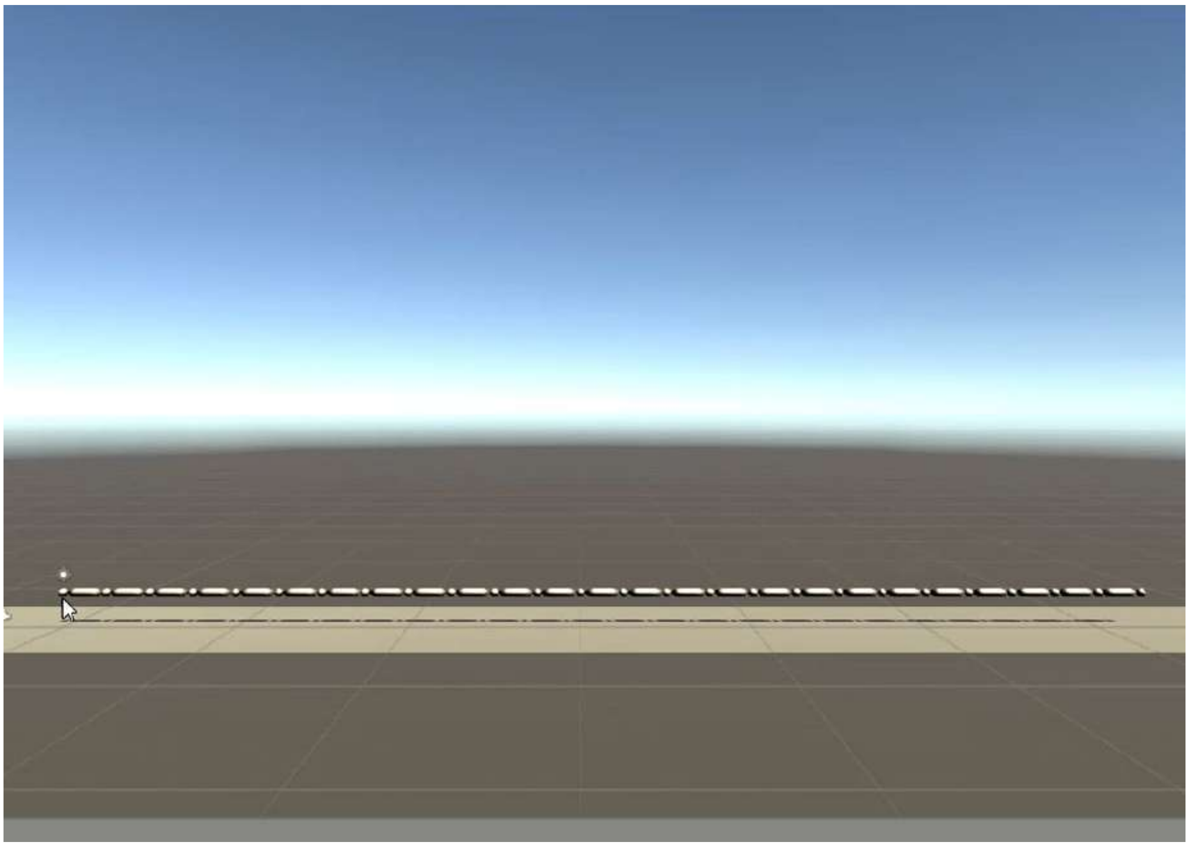


Figure 1: Simulated rope, formed by capsules and spheres (grey). The simulation is shown in its resting and untangled state.

The original idea behind this was to reproduce a multi-knot, consisting of random and different types of knots. This reproduction was the first step to engage a cognitive agent to use the human approach of solving the knot. In order to understand the human approach, we tried to pre-tie the knot, which is supposed to be solved. As soon as we were able to form a loop (see figure 2), without jittering or breaking effects, we continued our pursue to form a knot.

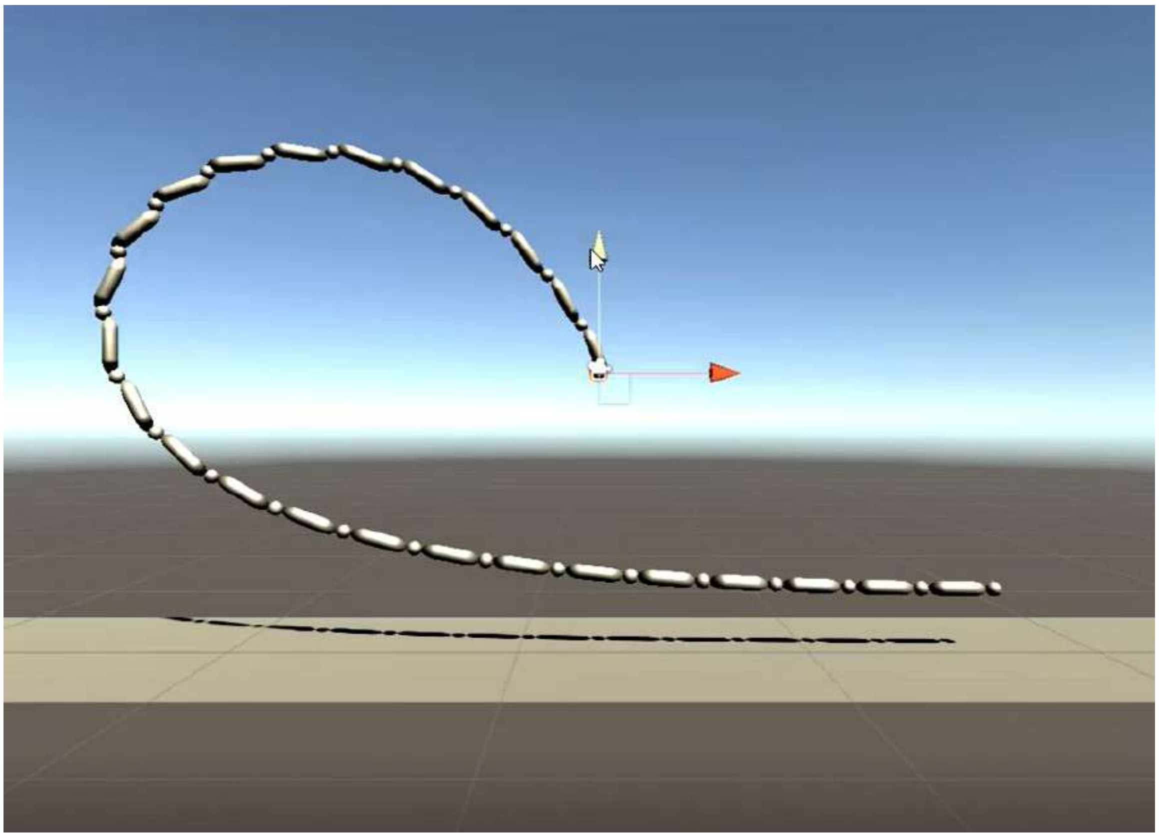


Figure 2: Forming of a loop with the rope, using the provided navigation by Unity3D. The simulation is shown in its active state.

The first sphere was initially *kinematic*, which lead to breakage, when paired with a *kinematic* last sphere. Also, the general prefab of sphere was equipped with two joints and was programmed to orientate itself to the previous sphere. However, the first sphere was not assigned to any sphere, therefore it automatically orientated towards the middle sphere. This conflict was omitted by designing a new prefab to instantiate the first sphere. Furthermore, this prefab was designed without a joint, which previously caused the jittering.

However we faced more difficulties to maintain a stable rope without losing its ability to be flexible. Although the rope was able to be manipulated into a loose knot (see figure 3), it was not able to form any tighter knots. At this point all manipulations were executed by Unity’s controller, which allowed a movement in all three axis directions.

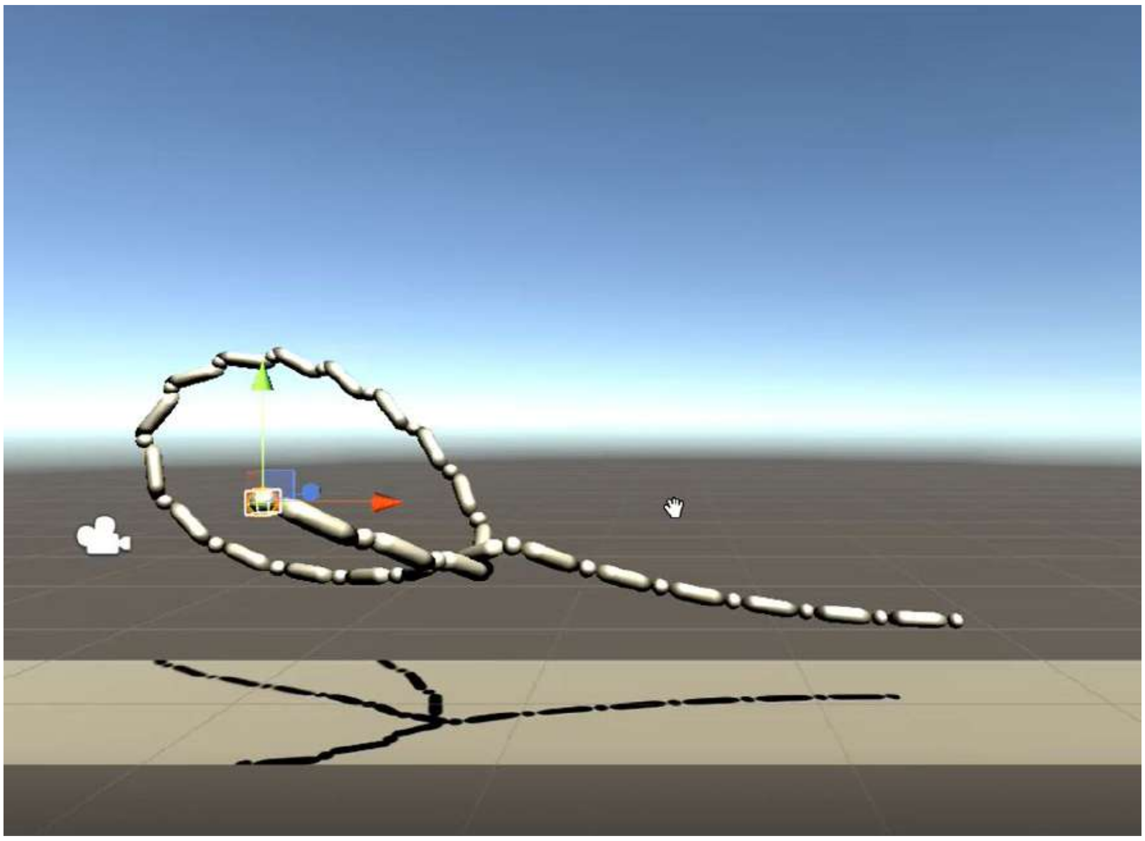


Figure 3: Forming of a loose knot with the rope, using the provided navigation by Unity3D. The simulation is shown in its active state

.

As soon as we attempted to form multiple or tighter knots, the rope would break. This behaviour was caused by the gaps between the capsules and spheres. These gaps ensured the flexibility of the rope, however, when strongly pulled they would let these objects slip through. Efforts to make the gaps smaller, thus tighter, were unsuccessful. Tightening the gaps caused by the joints would make the rope stiff and less manipulable. To the extent that a loop was not formable at one point. Figure 4 shows the tightest knot that could be formed with this simulation, without causing the prior mentioned disturbances in the simulation.

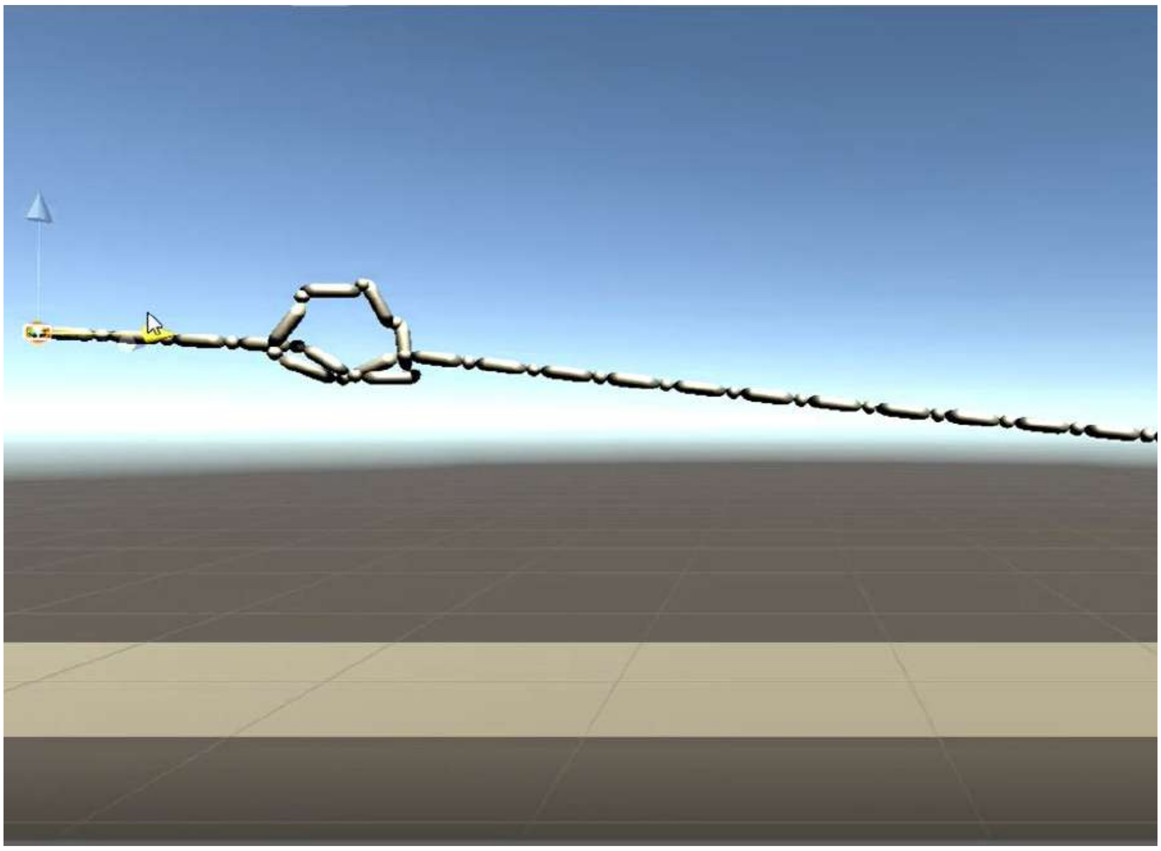


Figure 4: Tying a knot with the rope, by pulling at the opposing two ends of the rope.Tightest knot constellation possible to achieve, before it breaks. The simulation is shown in its active state.

Further attempts to evade the gaps, besides a decreasing distance between the objects, was to use a mesh combiner. The theory was to combine the objects into one mesh and thus mimic a cable, omitting any breakage caused by gaps. This idea is yet to be effectively implemented. Due to time limitations, we were unable to find further approaches, after our first attempts did not succeed.

The result of the entangled knot problem, in the context of the thesis, is a simulation of a flexible rope, that is able to be manipulated by a cognitive agent. All objects that are a component of the rope, are designed to mimic the character traits, that allows the simulation to behave accordingly, when pushed or pulled.

5.2 Shortest Path Simulation [RAHMAN]

The aim of this simulation was the detection of the shortest path between two points. By stretching at two waypoints, the most stretched edge indicates the shortest path. The waypoints are representing potential starting and target points on a map, thus imitating a navigational route (see figure 5). The construct is designed similar to a graph, with varied weighted nodes and different length edges.

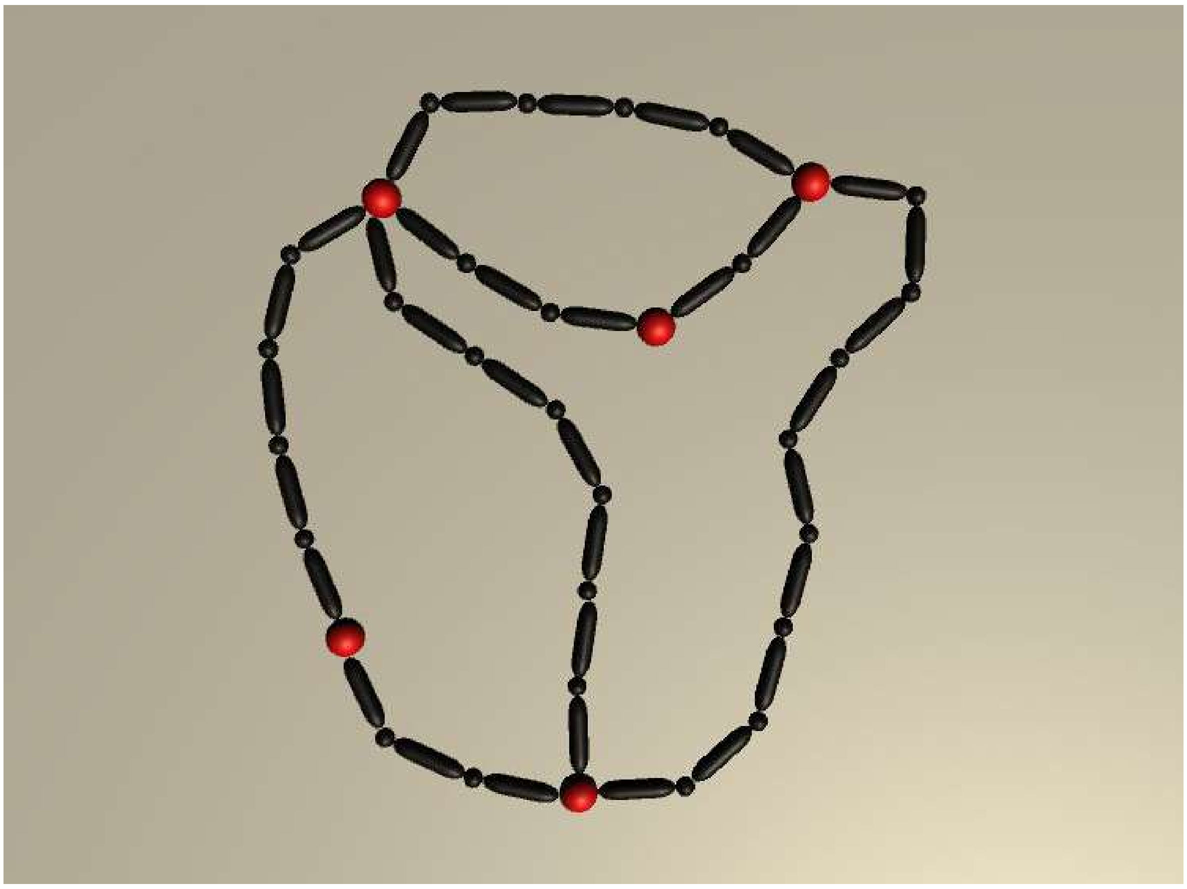


Figure 5: Simulated rope forming a net. Components of the rope are waypoints (red), as well as capsules and spheres (black). The simulation is shown in its resting state.

The rope, that we designed previously to simulate the knot function, was added to this simulation. In addition to the existing rope, another object category was added. We called these objects *waypoints*. This object category is introduced to connect multiple ropes. Since they obtain more specific attributes for movement, the cognitive- agent is able to have more control during the simulation. In contrast to these manipuble waypoints, the rope cannot be directly manipulated by the cognitive agent. To clearly differentiate between objects of the rope and the actual waypoints, different colours were used. While the capsules and spheres of the rope were black, the waypoints were hued red. A colour change from red to green indicates the selection of a point (see figure 6).

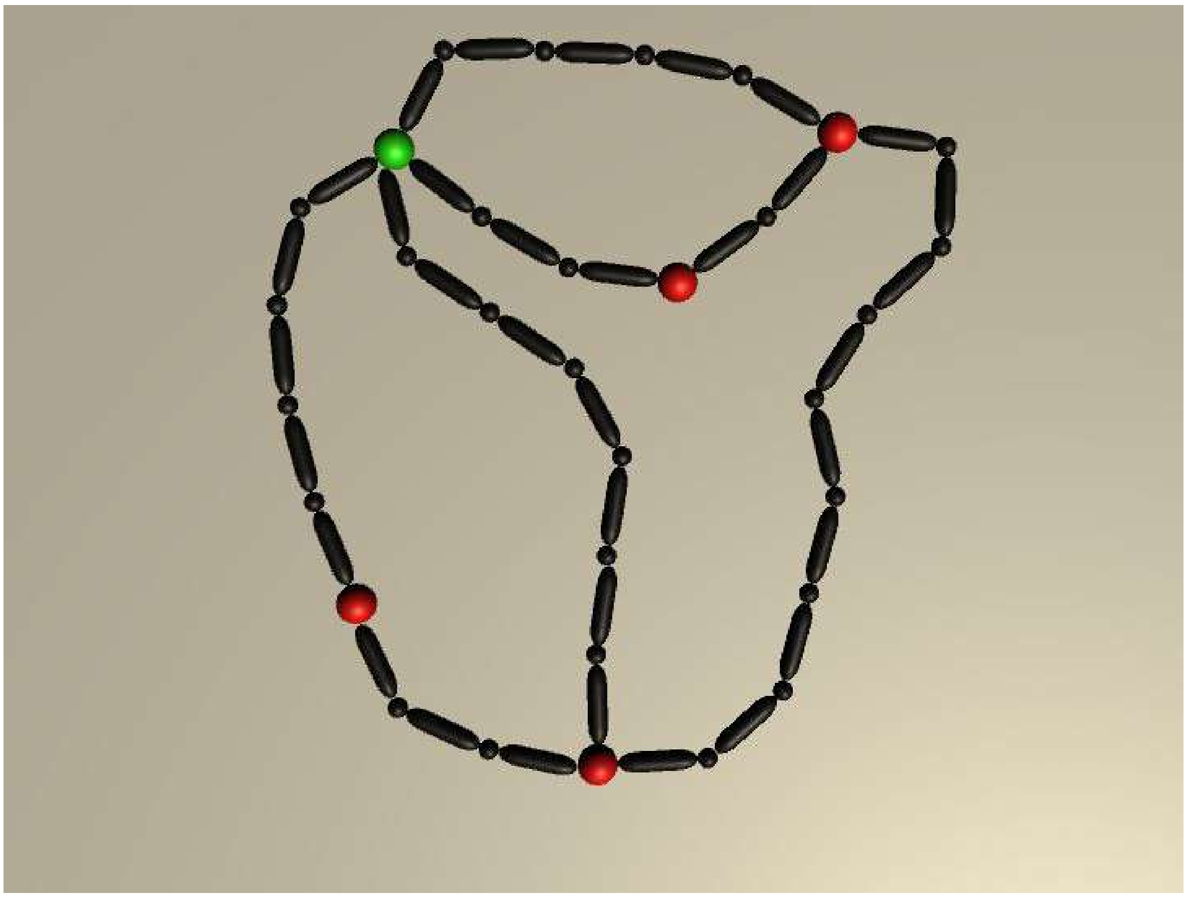


Figure 6: The selected waypoint is green, while selected ones still red. The simulation is shown in its alert state; stretching will start as soon as the second waypoint is selected.

A selection can be carried out by mouse click. Once two waypoints are selected, there is no other opportunity to select another point. This is what we call the *alert phase* of the simulation, which is occupied with the stretching. While the first waypoint is selected, it is simultaneously pinned down to its initial position. The stretching is then triggered by the selection of the second point.

The mechanism of the second point, as described in the methods chapter, allows it to move in the opposite direction. Once it reached the ultimate position of utmost stretching, meaning the function *OnJointBreak()* would reach a certain threshold (see figure 7), it is stopped.

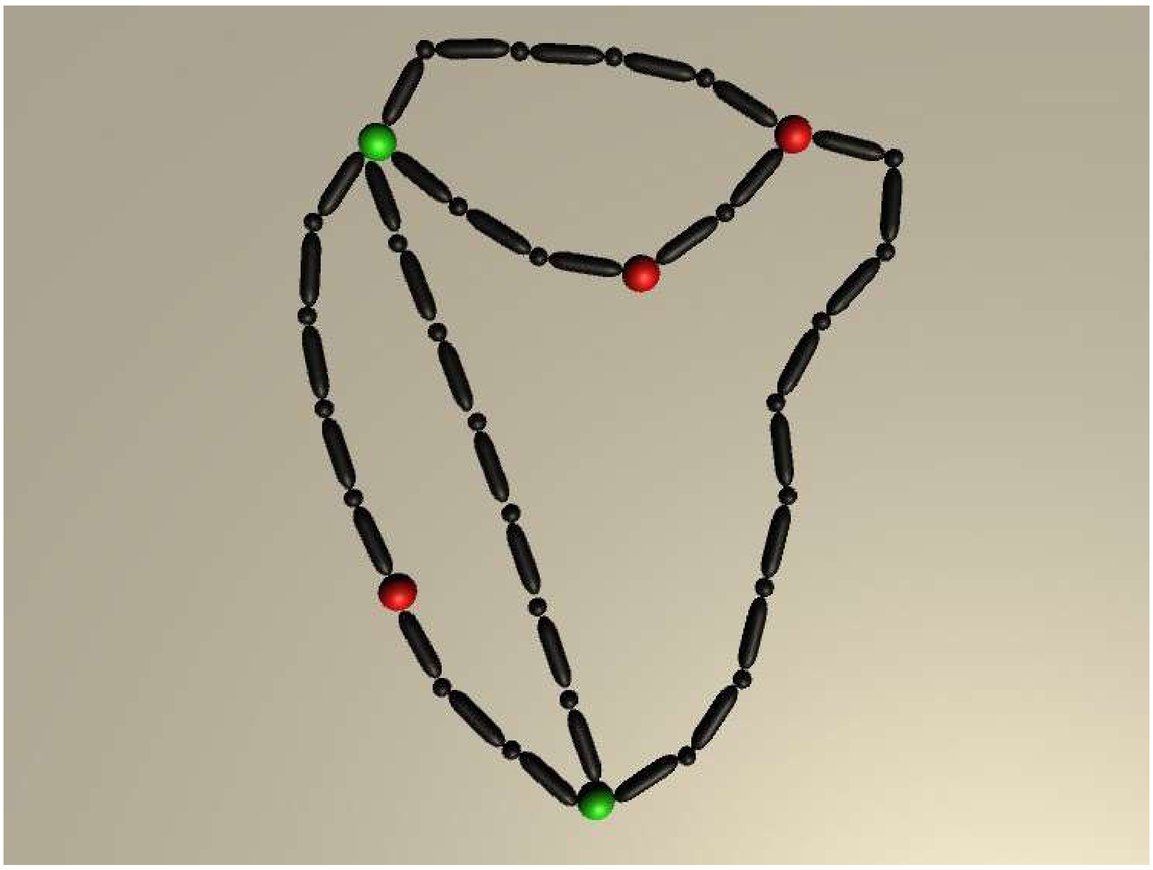


Figure 7: Two selected waypoints (green), while the first selected point is pinned, the second moves opposite, thus triggering a stretch. The simulation is shown in its active state.

At this current state of the simulation it is not possible to induce a route with multiple stops. However, this simulation is able to stretch at an two waypoints, regardless of how many edges are attached to it. Furthermore it is able to perform a full stretch by itself, without causing a deformation of the rope or damaging breakage. The detection of the shortest way, in the context of the thesis, is possible with this simulation.

5.3 Squaring Circle Simulation [RAHMAN + NICKELMANN]

In order to simulate this spatial cognitive problem entirely, it had to be broken down into several minor components. Therefore we started off by simulating the most interesting component, which measures the circumference of the circle/cylinder with a rope. While this part of the problem requires a simulation, the other components could be carried out by geometrical operations and are thus neglected in our examination. The function of this rope stems from the rope design implemented in the entangled knot problem. However, we chose to create a rope made of spheres only this time.

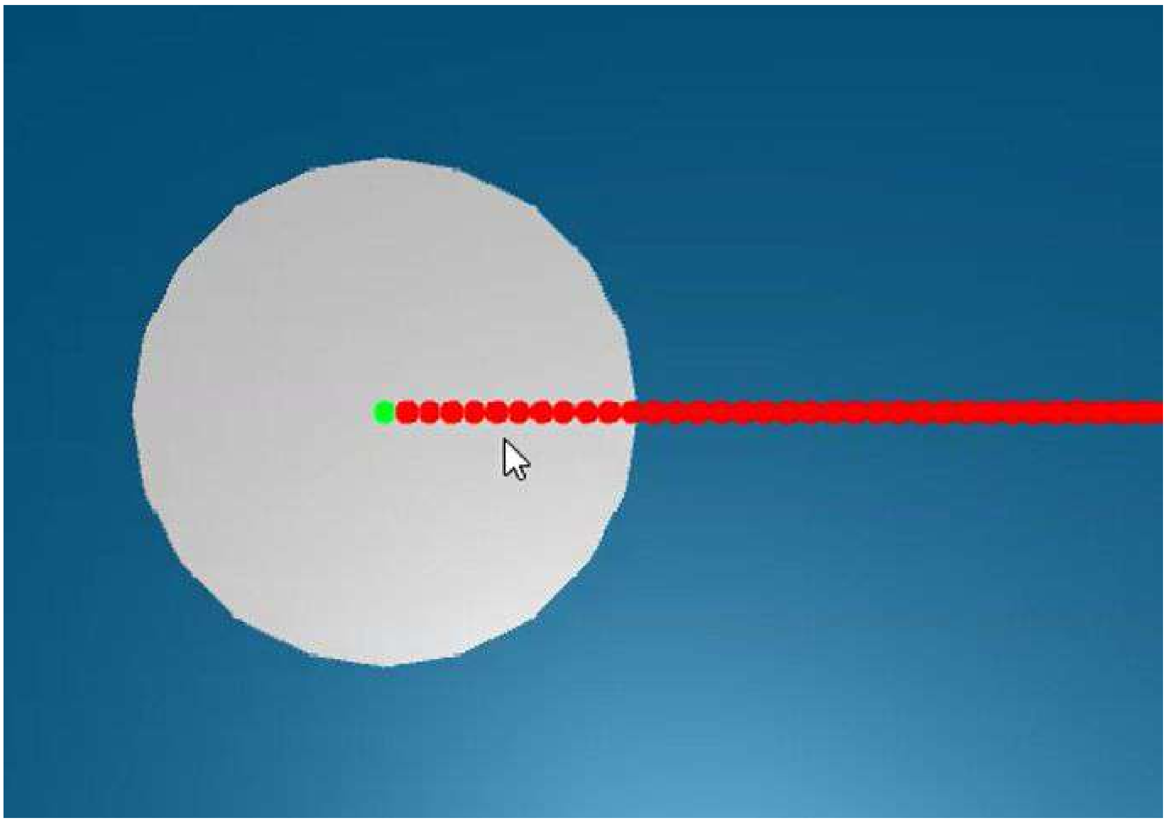


Figure 8: The grey cylinder is the measuring object; red spheres are the rope; here the first sphere is marked green by a cognitive agent. The simulation is shown in its active state.

As a result of the objects form, which was a cylinder in this case, we decided that this rope type was more accurate and more suitable in this circumference process. To make sure the rope stopped after once performing the circumference, we worked with collision detection (see *SCCollision*, in our methods chapter). As seen in figure 8, the first object of the rope is instantiated in the center of the cylinder.

At this point of the simulation, the cognitive agent/user is required to select the immediate sphere that is situated outside the cylinder, as well as the first sphere. This is not only to make sure we capture the radius, but also the start of the automated measuring (see figure 9).

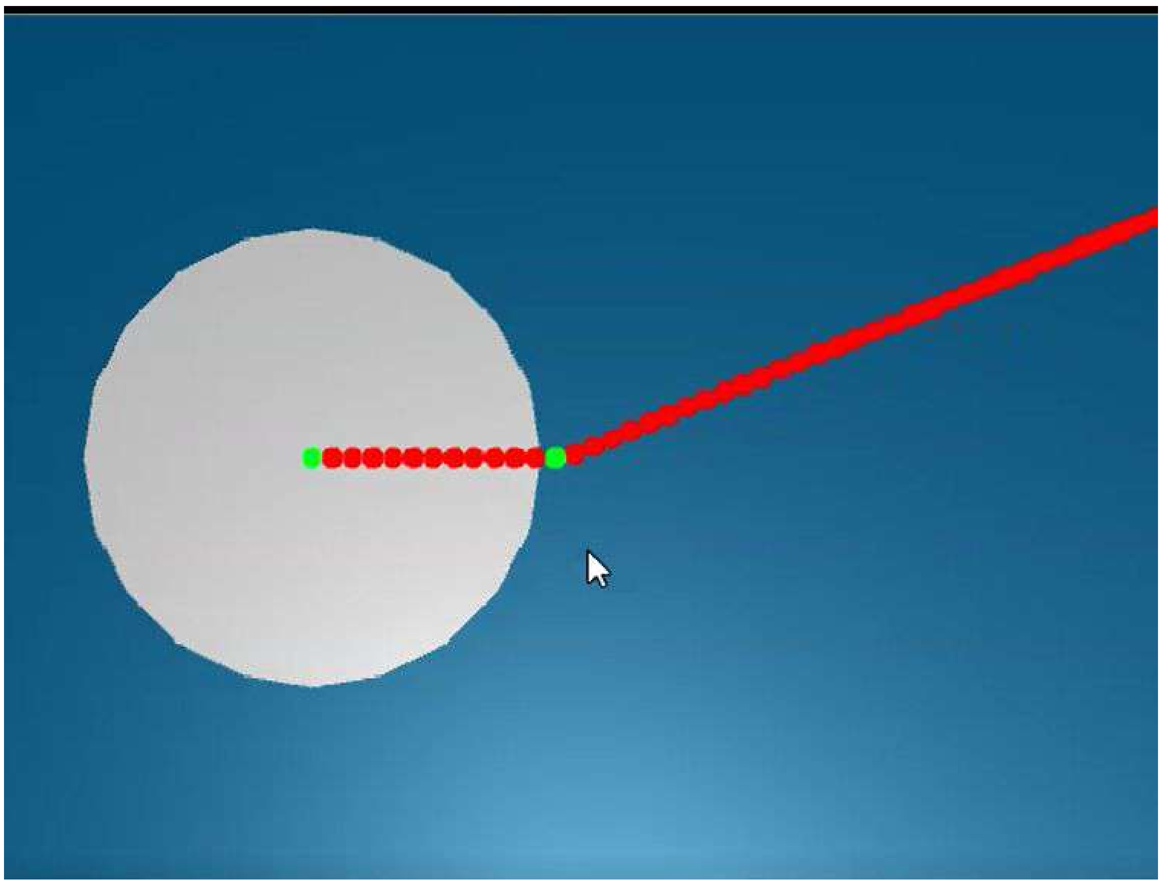


Figure 9: The grey cylinder is the measuring object; red spheres are the rope; green spheres mark the radius separately.The simulation is shown in its active state, while rope is wrapping around the circle.

To separately mark the selected and thus pinned spheres, they are coloured green (see figure 9). This selection and marking is meant to be performed by a cognitive agent. The selection of the second sphere, acts as trigger to start a counterclockwise rotation of the rope. Once the rope is wrapped around the cylinder, it is resulting in a full circumference (see figure 10).

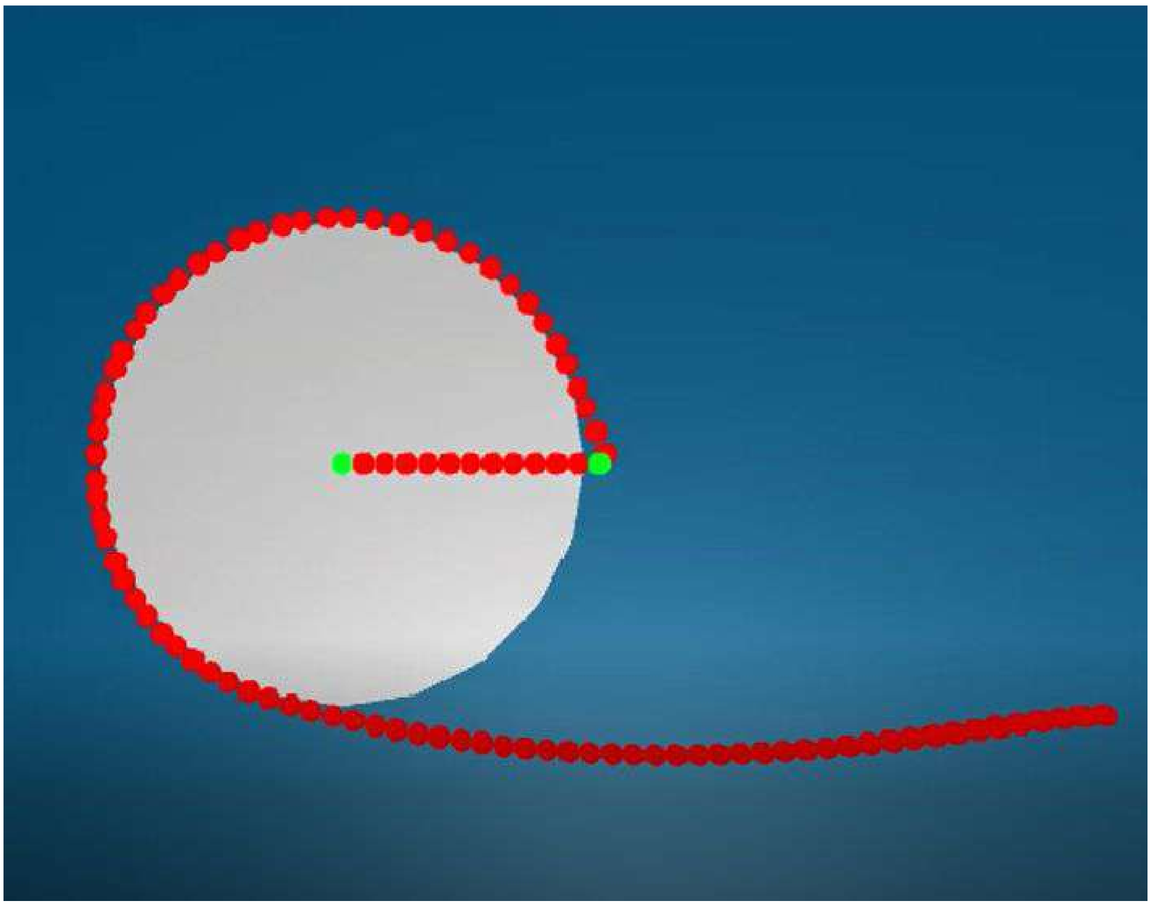


Figure 10: The simulation is shown in its active state, while rope is wrapping around the circle. This is automatically triggered by the second green sphere selection.

The rope stops upon the second collision with the first sphere of the rope, that is situated immediately outside the cylinder. To indicate this full circumference to the cognitive agent, we coloured this rope section orange (see figure 11).

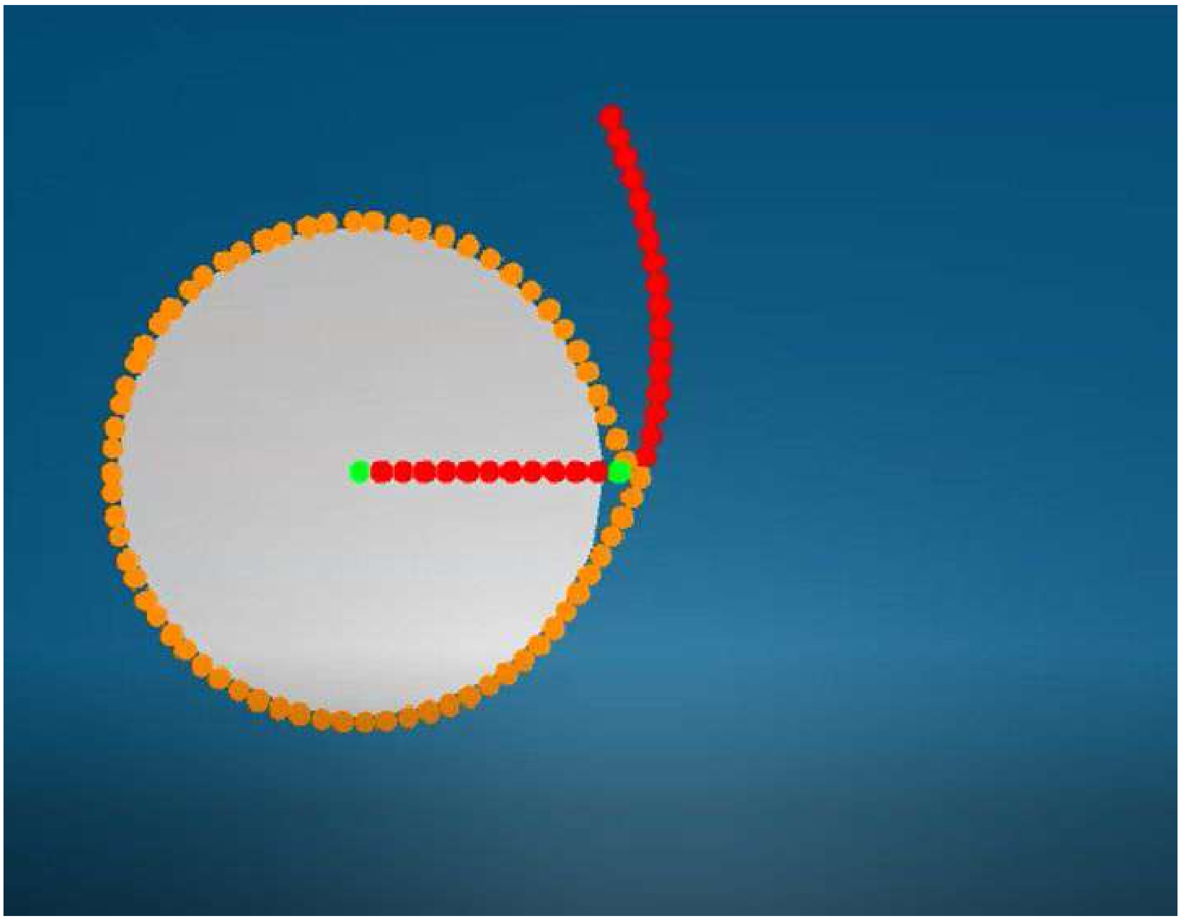


Figure 11: The orange coloured section of the rope is the measurement of the circumference of the circle.

This component of the simulation of the squaring circle problem is able to perform the circumference of the given cylinder and visually indicate the used rope section. Further steps can be carried out by geometrical operations.

In this chapter, the end results of the three implemented simulations were demonstrated. The implemented functionalities were explained, as well as how to interact with the simulations. We depicted the outcome simulations by screenshots and roughly explained how they work (visit the *Appendix* to see the full code of the three simulations).

In the following, concluding chapter, we reflect on the implementation process as a whole and discuss which of the previously set criteria (in regards to spatial cognition) we met. We also state which problems arose during the implementation process and which ones we could not fix within the time frame of this thesis. Additionally, we comment on why these simulations are of value for (strong) spatial cognition research and give an outlook on future research, hoping that our work will help other researchers and programmers.

5.4 Demo Video [RAHMAN]

Finally we would like to point to our link[35], which provides the full demovideo of the resulting simulations from this thesis. This demovideo was especially designed for the 14th International Conference on Spatial Information Theory (COSIT), which was held in Regensburg, Germany from the 9-13 of September 2019. Prof. Dr. Christian Freksa, from the CoSy group of the University of Bremen and participant in that conference, decided to present this new viewpoint on the examined problems in Strong Spatial Cognition in this thesis.

This demovideo is backed with descriptive keypoints. Thereby we aim to illustrate the process to the viewer, while providing a narrative to the visual input. In this video each examined spatial cognitive problem separately demonstrates the active process to find a solution via an cognitive agent. The naming of the problems is slightly different, as they are introduced by their working title. The length of the video is one minute and fifty seconds, illustrating a shortened and partially sped up video, which was directly captured from the screen. Moreover, the order of the shown problems is not of any particular order, however, can be reasoned by the argument of the sufficiency of the simulation and the dramaturgy.

The introductory screen (0:00 - 0:04) contains the name of the thesis and the name of the thesis composers. It is followed by the capture of the *Shortest Path Problem* (0:04 - 0:15), which contains the route network and the information comprised in bullet points. The information provided describes the aim, the visual objects, the problem that is examined and possible future applications. Moreover, the bullet

[35] link to the demovideo: https://drive.google.com/open?id=1UhN5Jr77k_qMSCEvs2RIbLGdF6TGoNj-

points are fading away after a short time, so that the viewer is not distracted and the observation of the simulation is not confined by the writings.

The actual demonstration of the *Shortest Path Problem* simulation (0:16 - 0:37) shows three possible options to manipulate the route net. In the first part (0:16 - 0:20), the selection of the waypoints is specified by the colour change from red to green. The stretching, in order to allocate the shortest path, is demonstrated in option 1 (0:20 - 0:22), option 2 (0:27 - .0:30) and option 3 (0:32 - 0:35). This is to use multiple examples to make an impression of this quick action and instill the fact that it is possible to freely choose any two waypoints in this simulation.

Furthermore, the entangled knot problem is introduced (0:40 - 1:13) and is supplied with bullet points that are explaining key elements, current problems and aims of the simulation (0:40 - 0:48). Meanwhile, the laid out rope is getting manipulated into a knot. In this next frame (0:51 - 1:05) we chose to incorporate another angle, bringing the viewer closer to the rope and thus be able to observe the manipulation. The first knot is created (1:05 - 1:10) by pulling at the end of the rope. Within this simulation the rope is only able to uphold a certain level of tightness, before it’s objects slip through the gaps can be viewed (1:11 - 1:13).

On the first screen of the simulation to the *Squaring Circle Problem*, a round object (grey) and a rope (red) is visible (1:14 - 1:16), accompanied by bullet point of information. The information states the aims, functions and other specific attributes, as well as problems that we were facing at that point of our work on this spatial cognitive problem. Considering to implement the visual from the last simulation, the selection of the points is marked by a change of colours (1:18 - 1:28). The selection of the second object of the rope is further triggering the circumference. It also enables orange-colouring of the section of the rope that is involved in this process (1:28 - 1:43), when the rope collides with it. The next step is only hinted in this simulation and implies the step of the unwrapping of the rope (1:43 - 1:50).

We aim to illustrate the process to the viewer with the video of all three simulations, while providing a narrative to the visual input. In this video each examined spatial cognitive problem separately demonstrates the active process to find a solution via an cognitive agent. An active process like a simulation, is more detailed and accurate in its depiction as a video and the provided instruction to the video. Therefore we encourage to follow the provided link to gain a deeper understanding of the simulation process.

6 Discussion and Conclusion [NICKELMANN + RAHMAN]

The findings of this thesis demonstrate that the simulation of spatial cognitive problems like an entangled knot, the shortest path and the squaring circle problem, are not only possible but harbour great potential. In the context of our work we are able to provide three working simulations of the above mentioned spatial cognitive problems. In contrast to existing approaches, it offers a solution without having to use any secondary tools or methods using 3D printouts, but is aimed to be directly integrated. Furthermore, this simulation incorporates the ideas of *Strong Spatial Cognition*, thereby mimicking the intuitive methods of a human to approach the problems.

Not only are these simulations faster, but they are also more moderate with resource usage. The simplistic design of the simulation was purposefully chosen to foster a decrease of the CPU[36] usage as well as the resources used for secondary tools[37] and memory capacity. One of the key aims of this work was to find a way to establish a method that combines a higher efficiency with a solution-orientated simulation to each problem. Besides the factor of efficiency, the visual aspect granted by a simulation, was crucial. In earlier approaches this part was analog, but nevertheless the key element to an alternative solution-finding process to a algorithmic rationalisation. It is this visual implementation that allows manipulation by cognitive agents, that differs this work from others and enables a direct integration of the internal system of a cognitive agent used in AI.

The translation of a spatial cognitive problem by an analog-intermediate step, which is then used by a cognitive agent in AI, only to be translated into a digital instruction, causes avoidable complexity to the solution-finding process. The simulation developed within this thesis offers a direct solution in a virtual environment, while

[36]CPU = central processing unit

[37]The secondary tool in this particular case is the 3D-printer, used for the printout of the net of the shortest path problem.

considering the analog nature of human methods. These qualities make them overall more efficient to solve cognitive spatial problems in future applications.

As previously mentioned, the visual appeal of the simulation was not the focus of this work, therefore the resulting simulation is kept in simple appearance. Additionally, to the relieve of the resource usage, this aids to our goal to keep the recreation and understanding of this work as a priority. The implementation of these simulations in a system of a cognitive agent used in AI, offers a great potential to evade complicated and lengthy algorithmic processes.

In consideration of the significant difference of the examined spatial cognitive problems, each problem is separately addressed the upcoming section of the discussion and conclusion. Beginning with the entangled knot problem, followed by the shortest path problem and ultimately followed by the squaring circle problem.

6.1 Entangled Knot Problem [RAHMAN]

The first spatial problem we assessed was the entangled knot problem. The examination was initiated by the first-hand determination of the features of the entangled knot problem, in order to appoint attributes that we needed. For this purpose we used our primary model of a knot, shown in draft 1. The idea was to create a rope that was not only flexible and manipulable but also able to uphold knots. With our simulation we overcame the limitations that we found in other simulations like e.g. in the above mentioned Obi-Ropes. Even though their product had excellent visual appeal and impressive fluidity in portrayed collisions, their purchasable rope- preset did not provide the ability to recreate a manipulable knot.

Although we intended to create a simulation that was close to the existing solution, we did undertake some changes. We decided to ultimately ignore the aspect of gravity in the simulation. In the real world approach by CoSy, a cable of a headphone was used to form a multiple knot. This was to imitate the everyday scenario of a tangled cable, regardless of how it ended up in the formation. The idea was to hand this knot to a cognitive agent and let that agent come up with a solution. Thereby conducting a human approach to solve the knot, rather than a algorithmic based way. We were closely following the experiment and implemented attributes, that allowed a manipulation of the rope in the simulation in a similar way to how it was handled in the real world approach. However the introduction of gravity caused problems in our digital version. Due to its own power, introducing gravity would cause a tear down of any loop that we formed beforehand. Therefore the rope would not withhold its shape, which made any further manipulation impossible. Simple yet essential operations like pulling or pushing the rope were hindered, which is why we took a step back and continued our simulation without adding gravity. Subsequently the simulation allowed a formation of a loop, followed by a manipulation of the rope to a knot. At this point we decided that the simulation of this spatial cognitive problem does not depend on gravity in order to work properly. Instead, we focused on the functionality of the simulation to imitate a tangled cable and the attributes for manipulation, as these were the key elements to this examination.

As mentioned in the results chapter, the current simulation of the knot allows to create a knot to a certain tightness. Further tightening of the knot, beyond that certain threshold, leads to a slip through of the individual objects. During our development phase we considered two other concepts to overcome this problem. As previously mentioned, the wrapping concept as well as decreasing distance between the individual objects did not bring any satisfying solutions. With hindsight the approach to combine the meshes of the objects and thereby creating a wrap around the cable still seems to be the most possible concept. The current simulation suggests that once the ‘slip through’ problem gets solved in future works, it is possible for a cognitive agent to manipulate and thus work on this cognitive spatial problem. With the help of a simulation, such as the one developed within the scope of our thesis, a cognitive agent is already able to manipulate the rope. So the next step is the forming of a multiple knot, which further leads to the study of how the human approach differs from an AI algorithm-based path.

As the work of the CoSy group suggests, the efficiency and most importantly the cut down of the load on the computing power is the overall aim. To achieve adequate results in digital approaches, further studies should take into account the differences in scale and functionality when it comes to convert a real world problem into a digital simulation - that again is used by an cognitive agent. We see great potential in the usage and implementation in further research about the improvement of AI cognitive agents and introducing them to use these human approaches. Using a digital simulation allows a seamless course of the process of solving a knot configuration, as it omits the extra translation and thereby reduces the resources that are required in this.

6.2 Shortest Path Problem [RAHMAN]

Since the rope, designed during the simulation of the entangled knot problem, turned out to serve all our criteria, we decided to use it as a base in our next simulation: the shortest path problem. In the context of this cognitive spatial problem, the fact that the rope objects slip through, is irrelevant. Decidedly we let this simulation inherit the previous rope design, when we were certain about the fact it would not hinder the performance of this simulation. This is because the shortest path problem requires different specifications in a simulation that do not depend on that level of flexibility. The operation that causes the rope to break is not part of this simulation. Even though the rope objects that are integrated in the map are designed as 3D objects, most of its manipulation is displayed in a 2D view to resemble a map from the birdsview. Due to the fact that the shortest path is revealed by stretching, we decided to design the simulation on a plane to preclude unnecessary rotations and deformation that would falsify the end result. All in all this approach is different from the current mechanical handling, which is using a 3D print-out of a net that resembles the paths on a map, without involving a plane. However the digital approach required a base to enable the stretching and provide a clear background to the scene to work with.

This simulation was more complex and thus challenging to create than the previous one. Not only did it require flexibility and stability while being manipulated, but we also wanted to introduce seemingly automated operations. By automated operations we mean the stretch that is triggered by the selection of the second waypoint. We preferred this way of designing the stretch, because of two reasons. First, we didn't want to involve a cognitive agent in this particular part to avoid any error e.g. over stretching. Secondly we decided to discharge and relieve the agent at this point, by opting for the automation of this part of the simulation.

The newly introduced object of this simulation was the *waypoint*. As mentioned in the previous chapter, this object is designed to have more functions implemented in its code. This is due to the fact that, after several attempts to find other solutions, this object type made it possible to generate a stretch. This stretch is performed without causing breakage. Furthermore our simulation allows the cognitive agent to select whatever point they want to manipulate. It was important to us to generate a object that has a simple and effective code-design; since the overall goal was to simplify the process and omit major strain on the computing power.

In addition, we decided to illustrate the selection of the waypoints by a change of colour, this is to allow the cognitive agent to receive an instant feedback. Due to the fact that currently only two points can be selected, we believe that an introduction of a visual feedback encourages a higher accuracy in the selection. The design of the current simulation would otherwise not be as intelligible to a cognitive user. Given the fact that the selection of the second waypoint endorses the stretch performance and simultaneously pins the first waypoint, it also restricts any further selection. Therefore we were considering to adopt any sort of feedback to make the selection stand out and avoid any confusion.

As previously mentioned, the current simulation grants the selection of two waypoints, imitating one point of departure and one point of arrival on a navigational route. Considering the aspect of navigation, which is the most suitable field of research to place this simulation in, we see more potential. The addition to multiple stops, indicating a stopover e.g like buying flowers before visiting a friend, might be of interest and provides further detail to the problem of the shortest path.

Running such a simulation in the background of an AI navigational application should increase its performance significantly. Some refinements to factor in like, for example, traffic congestion would still be necessary, but this could be achieved by weighing some route parts differently, making them more or less stretchable et cetera. In short, a string map functions as an easy and intuitive solution to the shortest path problem and is very useful in the broad domain of wayfinding. It offers opportunities for adaptations, and thus, a digital simulation of the string map is a meaningful contribution to the field of strong spatial cognition.

In our understanding, there are several further implementations and projects that can be continued from this work.

6.3 Squaring Circle Problem [NICKELMANN]

For the simulation of this cognitive spatial problem, we also decided to use the rope from the entangled knot simulation as the basis, as it fulfills all the criteria we set. The problem of the rope objects slipping through small gaps in the rope is not of relevance here as the rope will never collide with itself in a way that such a situation could occur. Similarly to the shortest path simulation, despite the objects being 3D objects, the simulation is basically a 2D simulation as the underlying squaring circle problem is a two-dimensional spatial problem. To reduce the workload, we decided to implement it with Unity3D; doing so does not inhibit the simulation in any way. Implementing the simulation in 2D might have even been harder as the cylinder (representing the circle) being in 3D helps to implement the step where an agent would pin the rope on the circle’s circumference. It is a bit tricky, but when you put all spheres inside the cylinder in a layer ignoring the collision and all spheres outside the cylinder in a layer not ignoring the collision with the cylinder, you can easily both simulate pinning the rope at the center of the circle and also wrap it around the cylinder, mimicking pinning the rope on the circle’s circumference. In this way, the simulation differs from the real-world problem. It is not possible to exactly pin down the rope on the outer line of the cylinder, you have to do it by letting the rope objects collide with the cylinder’s envelope. However, this is a very small inaccuracy. Such small inaccuracies might also emerge from pinning down the rope on a circle’s circumference, regarding the real-world task.

We implemented the simulation in such a way that users do not have to perform translation or rotation of objects themselves. The only thing they should know before starting the simulation is which spheres to click - the one at the cylinder’s center and the first sphere directly outside the cylinder. Having clicked the spheres in that order, the rope will wrap itself around the cylinder. The indication of the two clicked spheres and of the spheres making up the circumference also happens automatically. We tried implementing the option for the users to manually wrap the rope around the cylinder but it would not wrap around it very tightly, also, it takes way longer than the automatic approach. The C# method we used, *RotateAround*(), was more suitable to achieve this. It is an easy method that does not rely on many additional computations, being in line with the criteria we set for the simulations.

One more complicated aspect of the implementation is how the spheres are selected. As the cylinder is technically in front of the spheres, we had to utilize shaders for the red and green material that always render them on top. Additionally, when casting a ray from the camera to get the hit object, which should be a sphere, we had to ignore the cylinder as well. We achieved this by using the bitwise negator “~”, making the cylinder the only object not being selectable by the user. All in all, the simulation was complicated to set up in the beginning, but it now works as intended and is in line with the goals regarding strong spatial cognition.

We should also mention that, when it came to examine the squaring circle problem, we decided to change the previous rope design slightly. We introduced a rope that is consisting of spheres only. As described in the methods chapter, we aimed to produce a rope that performs a full circumference on the cylinder and to indicate the used rope length. Furthermore, we favoured the concept of using colour as an indicator as to when a full circumference is produced, as we wanted to get a universal feedback of the rope length. Therefore we used colour to indicate the rope section that was involved in the circumference. We preferred this procedure, as it allows to give an instant feedback to the cognitive agent. In addition to the visual feedback we were able to avoid a direct measurement, which would result in another number that becomes part of another algorithmic operation.

Further work on this simulation should start by implementing the following steps of the squaring circle problem. We have stated that the detection of the circumference is the most important step as it cannot be solved by the common straightedge-and-compass approach. However, only having implemented that step does not let a cognitive agent transform the circle into a square. We would have liked to implement the corresponding steps as well, but due to the time frame of the thesis, this was unfortunately not possible. We believe this simulation to be a great starting point for other programmers to implement a full squaring circles simulation based on the strings-and-pins approach and hope that it will be utilized by cognitive AI systems in the future.

Closing the gaps between the rope objects might also increase the accuracy of the circumference. We hope that this problem will be fixed in the future and that a non-springy Configurable Joint will be enabled by Unity3D, as that would also fix the problems for the entangled knot simulation. Additionally, as previously hinted at, finding a way to pin the spheres exactly on the circumference line of the cylinder would further improve the accuracy of the simulation. At that point, the accuracy of a digital simulation of the squaring circle problem would be higher than the accuracy of the real-world solution via the strings-and-pins approach.

Our hope is for this simulation to be utilized by cognitive AI agents. The basic problem of squaring a circle is a common problem in geometry, and it being easily solvable by an AI agent without having to use the tools of strings and pins yourself would be very efficient. Furthermore, such a simulation would provide a great opportunity for future research of the still novel research area of strong spatial cognition. Closing this chapter and the thesis overall will be a résumé of the goals we achieved with the work of this thesis and a compact outlook on future research and possible application areas.

6.4 Reached Goals & Outlook [NICKELMANN]

We have worked on three digital simulations of tasks associated with the scientific discipline of spatial cognition. Those three tasks are knot-tying and untying, finding the shortest path in a route network and transforming a circle into a square with the same surface area. Those tasks are each implemented in a digital simulation, showcasing specific approaches to these problems based on manipulable ropes.

As outlined in Chapter 2.2, these kinds of simulations put the focus on the direct use of spatial information instead of extensive use of algorithms and highlight a current gap in spatial cognition research and the application of its findings in cognitive AI systems. All other simulations dealing with knot-tying and untying put their focus on the domain they are to be applied in (which is most frequently a medical domain) and rely on many algorithms. Regarding the other two spatial problems, there has not even been an attempt to produce a digital simulation of them.

Thus, our work is of value especially for research of the novel field of strong spatial cognition. We have implemented simulations for problems that have either not been digitally tackled by a non-algorithmic approach or have not been implemented at all. The goal in mind - reducing the number of computations and handling spatial structures directly - is not commonly found within the realm of software computing. We hope that our contribution can be a starting point for future research and that researchers and programmers alike can use these simulations and continue to work on them. The simulations meet every criteria we have set, all of them using few algorithms and encompassing the direct manipulation of spatial configurations. There are problems that need to be fixed if the entangled knot simulation is to be applied in a cognitive AI system, the gaps between the rope objects inhibiting agents from tying the knots tightly. For the squaring circle simulation, further steps need to be implemented to actually transform the circle into a square with the same surface area. The shortest path simulation can already be applied, however, writing a script that can discern the routes and waypoints and automatically produce a road map would drastically reduce the needed workload. The simulations work reliably and contain valid methods of tackling the spatial problems. All in all, we believe our work to entice further research and application of these simulations and hope we can raise awareness for the novel research area of strong spatial cognition and its possibilities.

Pondering further research, we found that it is possible to implement simulations of spatial problems without using many algorithms. We hope to inspire programmers to try implementing such simulations as well, utilizing the game engines we have presented as they provide an excellent digital programming environment. They include many tools that are intuitive to use and directly show the output of the code, supporting the programming. Future research should be guided by the findings of spatial cognition and, hopefully, strong spatial cognition as well, as implementing simulations with those research areas in mind can produce intuitive, straightforward results that can be easily applied to cognitive AI systems.

We hope that we have inspired researchers and programmers alike to delve into strong spatial cognition research and implement simulations with the aforementioned goals in mind. We are happy to contribute the three simulations of common spatial problems and believe our work to be of value. We are eager to observe the progress made in this domain and to revisit it in the future.

7 Literature

Alibali, Martha W. (2005): Gesture in Spatial Cognition: Expressing, Communicating, and Thinking About Spatial Information. In: *Spatial Cognition & Computation*, 05 (5), 307-331.

Brown, Joel/Latombe, Jean-Claude/Montgomery, Kevin (2004): Real-time Knot-tying Simulation. In: *The Visual Computer*, 04 (20), 165-179.

Buckley, Jeffrey/Seery, Niall/Canty, Donal (2019): Investigating the use of spatial reasoning strategies in geometric problem solving. In: *International Journal of Technology and Design Education*, 19 (29), 341-362.

Burgess, Neil (2008): Spatial cognition and the brain. In: *Annals of the New York Academy of Sciences*, 08 (1124), 77-97.

Burgess, Neil (2006): Spatial memory: How egocentric and allocentric combine. In: *Trends in Cognitive Sciences*, 06 (12), 551-557.

Byrne, Ruth M. J./Johnson-Laird, P. N. (1989): Spatial Reasoning. In: *Journal of Memory and Language*, 89 (28), 564-575.

Crabtree, Jonathan (2016): Squaring the Circle - A Practical Approach. In: *Vinculum*, 16 (53), 7-9.

Dani, Shrikrishna Gopalrao (2012): Ancient Indian Mathematics - A Conspectus. In: *Resonance*, 12 (17), 236-246.

Dewdney, Alexander K. (1988): The Armchair Universe. An Exploration of Computer Worlds. New York: W. H. Freeman and Company.

Dreyfus, Hubert L. (1978): What Computers can’t do: The Limits of Artificial Intelligence (Revised Edition). New York: HarperCollins.

Engels, Hermann (1977): Quadrature of the Circle in Ancient Egypt. In: *Historia Mathematica*, 77 (4), 137-140.

Falomir, Zoe/Olteteanu, Ana-Maria (2015): Logics based on qualitative descriptors for scene understanding. In: *Neurocomputing*, 15 (161), 3-16.

Freksa, Christian (2015): Strong Spatial Cognition. In: Fabrikant, Sara Irina/Raubal, Martin/Bertolotto, Michela/Davies, Clare/Freundschuh, Scott/Bell, Scott (eds.) *COSIT 2015. Lecture Notes in Computer Science*, 15 (9368), 65-86.

Freksa, Christian/Olteteanu, Ana-Maria/Barkowsky, Thomas/van de Ven, Jasper/Schultheis, Holger (2017): Spatial Problem Solving in Spatial Structures. In: *Lecture Notes in Computer Science*, 17 (10607), 18-29. (Cited as *Freksa et al. 2017a*)

Freksa, Christian/Barkowsky, Thomas/Dylla, Frank/Falomir, Zoe/Olteteanu, Ana-Maria/van de Ven, Jasper (2017): Spatial Problem Solving and Cognition. In: Zacks, Jeffrey M./Taylor, Holly A. (eds.) *Representations in Mind and World. Essays Inspired by Barbara Tversky (1st Edition)*. New York: Routledge. (Cited as *Freksa et al. 2017b*)

Freksa, Christian/Olteteanu, Ana-Maria/Ali, Ahmed Loai/Barkowsky, Thomas/van de Ven, Jasper/Dylla, Frank/Falomir, Zoe (2016): Towards Spatial Reasoning with Strings and Pins. In: *Advances in Cognitive Systems, Poster Collection*, 16 (4), 1-15.

Harmon, Brendan Alexander (2016): Embodied Spatial Thinking in Tangible Computing. In: *Tenth International Conference on Tangible Embedded and Embodied Interaction 2016*, 16 (10), 693-696.

Haun, Daniel B.M./Rapold, Christian J./Janzen, Gabriele/Levinson, Stephen C. (2011): Plasticity of Human Spatial Cognition: Spatial Language and Cognition Covary Across Cultures. In: *Cognition*, 11 (119), 70-80.

Hobson, Ernest William (1913): “Squaring the Circle” - A History of the Problem. Cambridge: University Press.

Kuipers, Benjamin (1978): Modeling Spatial knowledge. In: *Cognitive Science*, 78 (2), 129-153.

Mayer, Hermann/Gomez, Faustino/Wierstra, Daan/Nagy, Istvan/Knoll, Alois/Schmidhuber, Jürgen (2008): A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks. In: *Advanced Robotics*, 08 (22), 1521-1537.

Nebel, Bernhard/Freksa, Christian (2011): Ai Approaches to Cognitive Systems - The Example of Spatial Cognition. In: *Informatik-Spektrum*, 11 (34), 462-468.

Penn, Alan (2003): Space Syntax and Spatial Cognition: Or why the Axial Line? In: *Environment and Behaviour*, 03 (35), 30-65.

Phillips, Jeff/Ladd, Andrew/Kavraki, Lydia E. (2002): Simulated Knot Tying. In: *International Conference on Robotics and Automation*, 02 (1), 841-846.

Saraswathi Amma, T. A. (1999): Geometry in Ancient and Medieval India. Delhi: Motilal Banarsidass Publishers.

Schubert, Hermann (1891): The Squaring of the Circle: An Historical Sketch of the Problem from the Earliest Times to the Present Day. In: *The Monist*, 1891 (1), 197-228.

Seidenberg, Abraham (1962): The Ritual Origin of Geometry. In: *Archive for History of Exact Sciences*, 61 (1), 488-527.

Sinha, Chris/Jensen De López, Kristine (2000): Language, Culture and the Embodiment of Spatial Cognition. In: *Cognitive Linguistics*, 00 (11), 17-41.

Thomas, Laura E./Lleras, Alejandro (2007): Moving eyes and moving thought: On the spatial compatibility between eye movements and cognition. In: *Psychonomic Bulletin & Review*, 07 (14), 663-668.

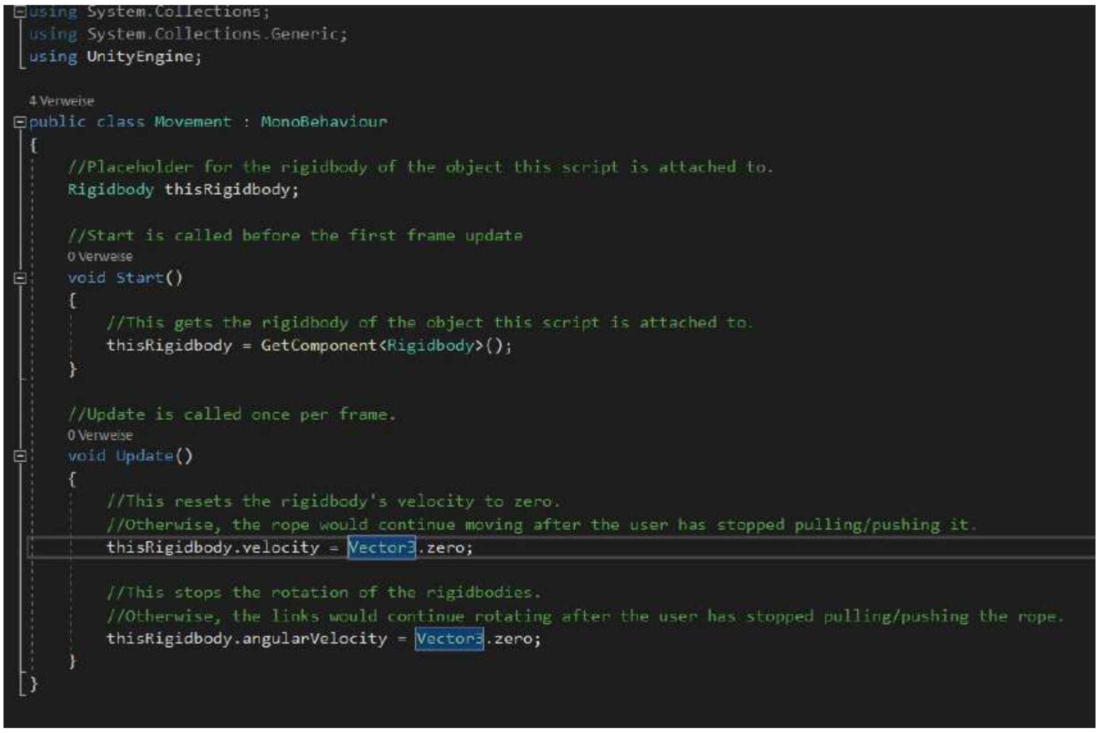
Tomai, Emmett/Forbus, Kenneth D./Usher, Jeffery (2004): Qualitative Spatial Reasoning for Geometric Analogies. In: *Proceedings of the 18th International Qualitative Reasoning Workshop, Evanston, Illinois, USA*, 04 (18).

Uttal, David H. (2000): Seeing the big picture: Map use and the development of spatial cognition. In: *Developmental Science*, 00 (3), 247-264.

Vasilyeva, Marina/Lourenco, Stella F. (2012): Development of Spatial Cognition. In: *Wiley Interdisciplinary Reviews: Cognitive Science*, 12 (3), 349-362.

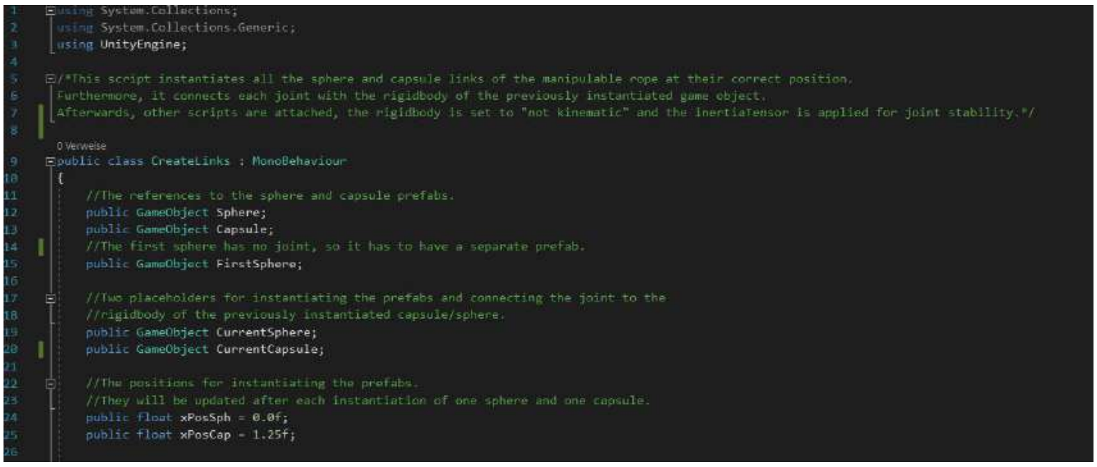
Wang, F./Burdet, E./Vuillemin, R./Bleuler, H. (2005): Knot-tying with Visual and Force Feedback for VR Laparoscopic Training. In: *Annual International Conference of the IEEE Engineering in Medicine and Biology*, 05 (7), 5778-5781.

8 Appendix



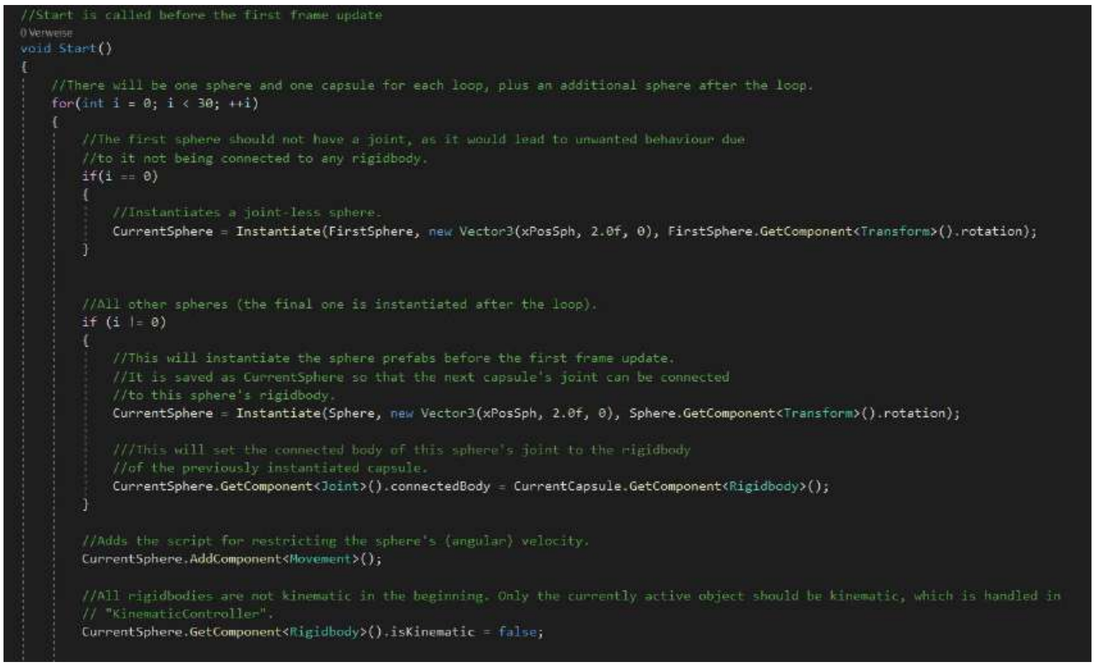
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

4 Verweise
public class Movement : MonoBehaviour
{
    //Placeholder for the rigidbody of the object this script is attached to.
    Rigidbody thisRigidbody;

    //Start is called before the first frame update
    0 Verweise
    void Start()
    {
        //This gets the rigidbody of the object this script is attached to.
        thisRigidbody = GetComponent<Rigidbody>();
    }

    //Update is called once per frame.
    0 Verweise
    void Update()
    {
        //This resets the rigidbody's velocity to zero.
        //Otherwise, the rope would continue moving after the user has stopped pulling/pushing it.
        thisRigidbody.velocity = Vector3.zero;

        //This stops the rotation of the rigidbodies.
        //Otherwise, the links would continue rotating after the user has stopped pulling/pushing the rope.
        thisRigidbody.angularVelocity = Vector3.zero;
    }
}
```

Image A1: The complete Movement script.



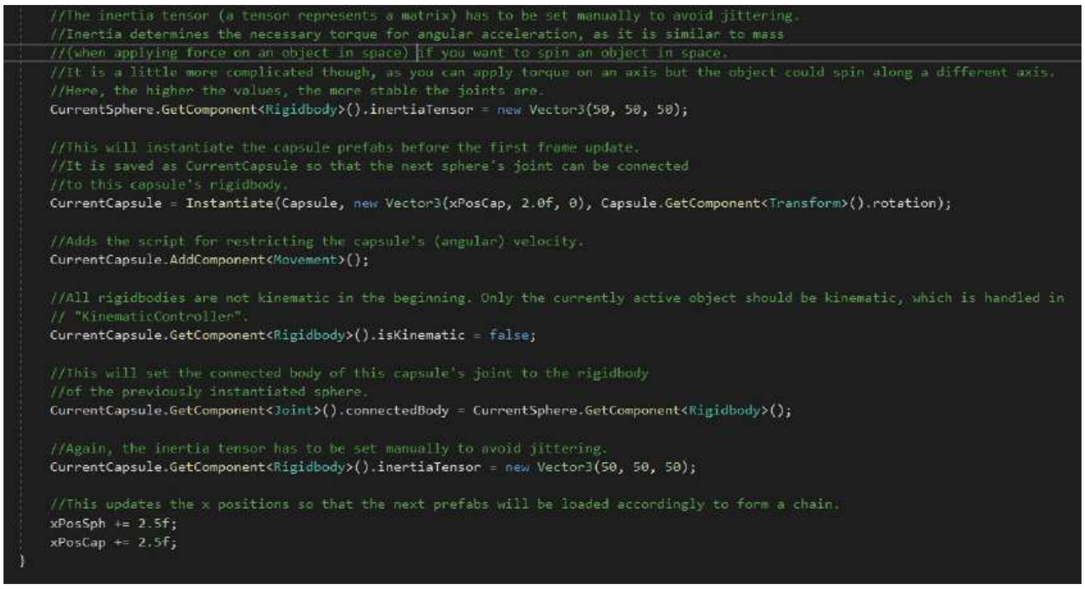
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/*This script instantiates all the sphere and capsule links of the manipulable rope at their correct position.
Furthermore, it connects each joint with the rigidbody of the previously instantiated game object.
Afterwards, other scripts are attached, the rigidbody is set to "not kinematic" and the inertiaTensor is applied for joint stability.*/

0 Verweise
public class CreateLinks : MonoBehaviour
{
    //The references to the sphere and capsule prefabs.
    public GameObject Sphere;
    public GameObject Capsule;
    //The first sphere has no joint, so it has to have a separate prefab.
    public GameObject FirstSphere;

    //Two placeholders for instantiating the prefabs and connecting the joint to the
    //rigidbody of the previously instantiated capsule/sphere.
    public GameObject CurrentSphere;
    public GameObject CurrentCapsule;

    //The positions for instantiating the prefabs.
    //They will be updated after each instantiation of one sphere and one capsule.
    public float xPosSph = 0.0f;
    public float xPosCap = 1.25f;
```

Image A2: The first part of the CreateLinks script, showing the variables.



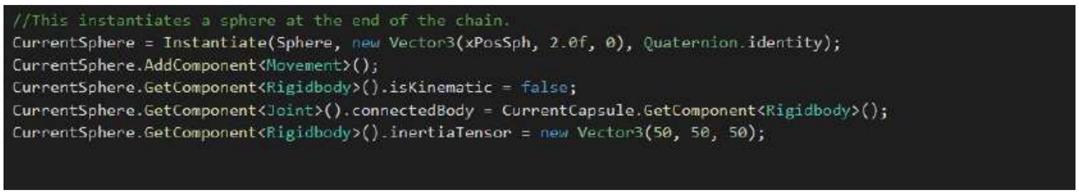
```
//Start is called before the first frame update
0 Verweise
void Start()
{
    //There will be one sphere and one capsule for each loop, plus an additional sphere after the loop.
    for(int i = 0; i < 30; ++i)
    {
        //The first sphere should not have a joint, as it would lead to unwanted behaviour due
        //to it not being connected to any rigidbody.
        if(i == 0)
        {
            //Instantiates a joint-less sphere.
            CurrentSphere = Instantiate(FirstSphere, new Vector3(xPosSph, 2.0f, 0), FirstSphere.GetComponent<Transform>().rotation);
        }


        //All other spheres (the final one is instantiated after the loop).
        if (i != 0)
        {
            //This will instantiate the sphere prefabs before the first frame update.
            //It is saved as CurrentSphere so that the next capsule's joint can be connected
            //to this sphere's rigidbody.
            CurrentSphere = Instantiate(Sphere, new Vector3(xPosSph, 2.0f, 0), Sphere.GetComponent<Transform>().rotation);

            ///This will set the connected body of this sphere's joint to the rigidbody
            //of the previously instantiated capsule.
            CurrentSphere.GetComponent<Joint>().connectedBody = CurrentCapsule.GetComponent<Rigidbody>();
        }

        //Adds the script for restricting the sphere's (angular) velocity.
        CurrentSphere.AddComponent<Movement>();

        //All rigidbodies are not kinematic in the beginning. Only the currently active object should be kinematic, which is handled in
        // "KinematicController".
        CurrentSphere.GetComponent<Rigidbody>().isKinematic = false;
```

Image A3: The second part of the CreateLinks script, showing the first part of the loop within the Start() method, instantiating the rope objects.



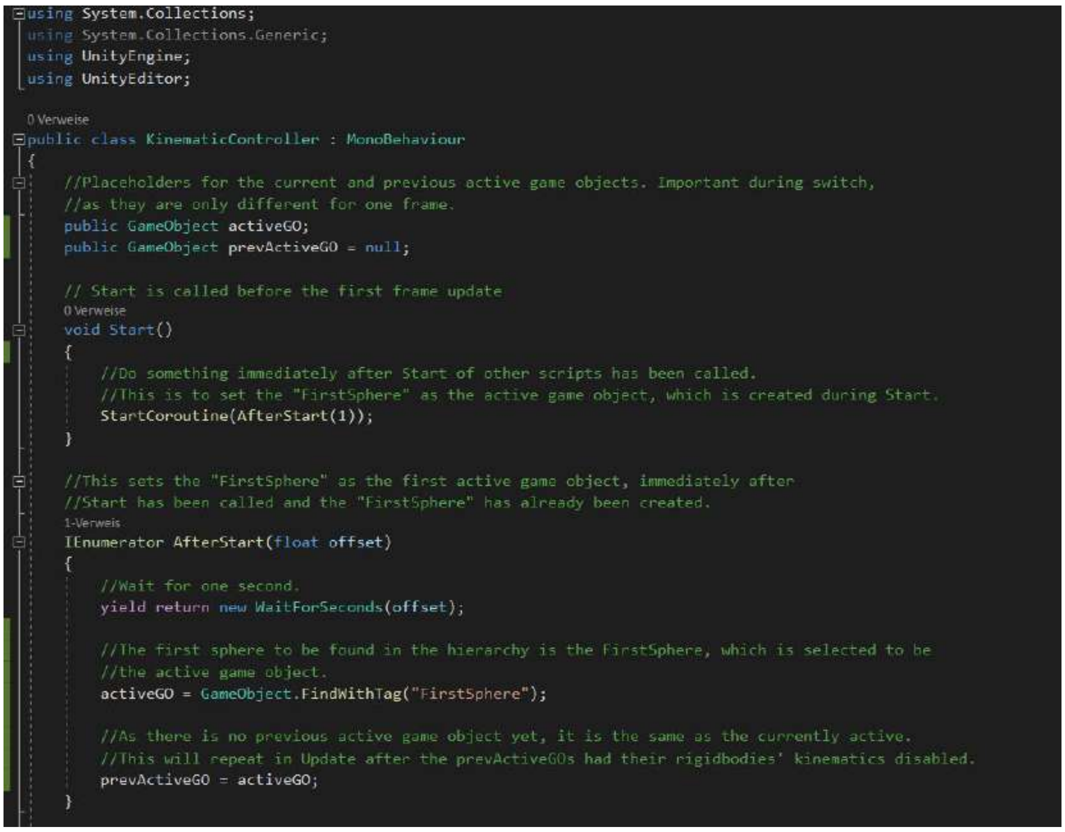
```
        //The inertia tensor (a tensor represents a matrix) has to be set manually to avoid jittering.
        //Inertia determines the necessary torque for angular acceleration, as it is similar to mass
        //(when applying force on an object in space) if you want to spin an object in space.
        //It is a little more complicated though, as you can apply torque on an axis but the object could spin along a different axis.
        //Here, the higher the values, the more stable the joints are.
        CurrentSphere.GetComponent<Rigidbody>().inertiaTensor = new Vector3(50, 50, 50);

        //This will instantiate the capsule prefabs before the first frame update.
        //It is saved as CurrentCapsule so that the next sphere's joint can be connected
        //to this capsule's rigidbody.
        CurrentCapsule = Instantiate(Capsule, new Vector3(xPosCap, 2.0f, 0), Capsule.GetComponent<Transform>().rotation);

        //Adds the script for restricting the capsule's (angular) velocity.
        CurrentCapsule.AddComponent<Movement>();

        //All rigidbodies are not kinematic in the beginning. Only the currently active object should be kinematic, which is handled in
        // "KinematicController".
        CurrentCapsule.GetComponent<Rigidbody>().isKinematic = false;

        //This will set the connected body of this capsule's joint to the rigidbody
        //of the previously instantiated sphere.
        CurrentCapsule.GetComponent<Joint>().connectedBody = CurrentSphere.GetComponent<Rigidbody>();

        //Again, the inertia tensor has to be set manually to avoid jittering.
        CurrentCapsule.GetComponent<Rigidbody>().inertiaTensor = new Vector3(50, 50, 50);

        //This updates the x positions so that the next prefabs will be loaded accordingly to form a chain.
        xPosSph += 2.5f;
        xPosCap += 2.5f;
    }
```

Image A4: The third part of the CreateLinks script, showing the end of the loop instantiating the objects.



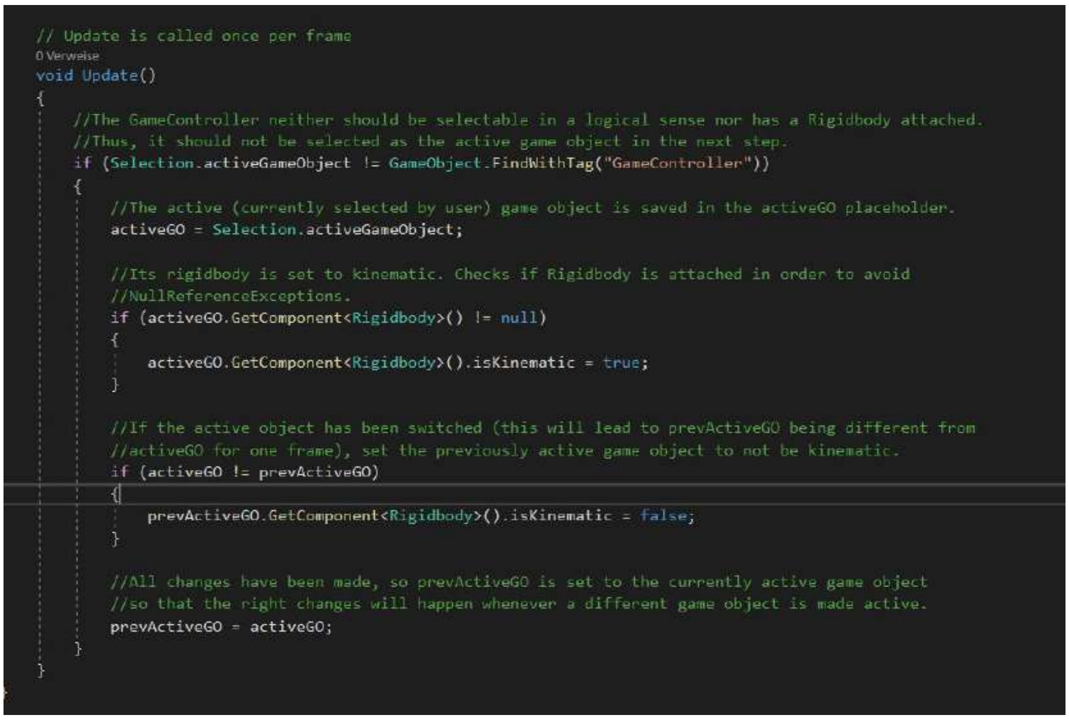
```
//This instantiates a sphere at the end of the chain.
CurrentSphere = Instantiate(Sphere, new Vector3(xPosSph, 2.0f, 0), Quaternion.identity);
CurrentSphere.AddComponent<Movement>();
CurrentSphere.GetComponent<Rigidbody>().isKinematic = false;
CurrentSphere.GetComponent<Joint>().connectedBody = CurrentCapsule.GetComponent<Rigidbody>();
CurrentSphere.GetComponent<Rigidbody>().inertiaTensor = new Vector3(50, 50, 50);
```

Image A5: The last part of the CreateLinks script, showing the last sphere being instantiated.



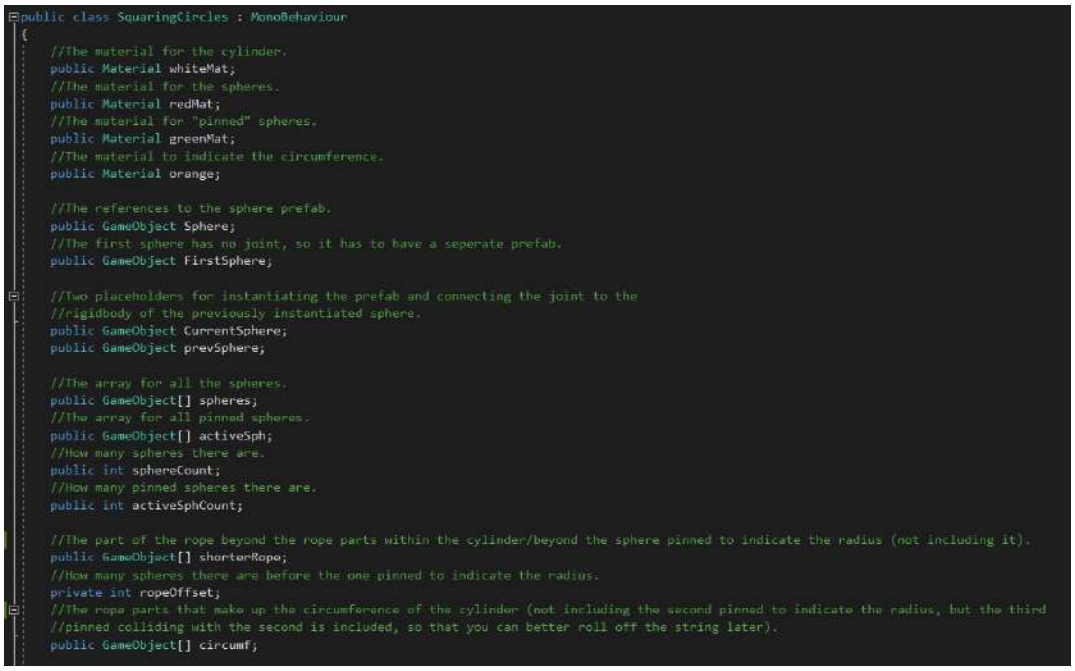
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;

0 Verweise
public class KinematicController : MonoBehaviour
{
    //Placeholders for the current and previous active game objects. Important during switch,
    //as they are only different for one frame.
    public GameObject activeGO;
    public GameObject prevActiveGO = null;

    // Start is called before the first frame update
    0 Verweise
    void Start()
    {
        //Do something immediately after Start of other scripts has been called.
        //This is to set the "FirstSphere" as the active game object, which is created during Start.
        StartCoroutine(AfterStart(1));
    }

    //This sets the "FirstSphere" as the first active game object, immediately after
    //Start has been called and the "FirstSphere" has already been created.
    1-Verweis
    IEnumerator AfterStart(float offset)
    {
        //Wait for one second.
        yield return new WaitForSeconds(offset);

        //The first sphere to be found in the hierarchy is the FirstSphere, which is selected to be
        //the active game object.
        activeGO = GameObject.FindWithTag("FirstSphere");

        //As there is no previous active game object yet, it is the same as the currently active.
        //This will repeat in Update after the prevActiveGOs had their rigidbodies' kinematics disabled.
        prevActiveGO = activeGO;
    }
```

Image A6: The first part of the KinematicController script, showing the setup for the kinematic switch mechanic.



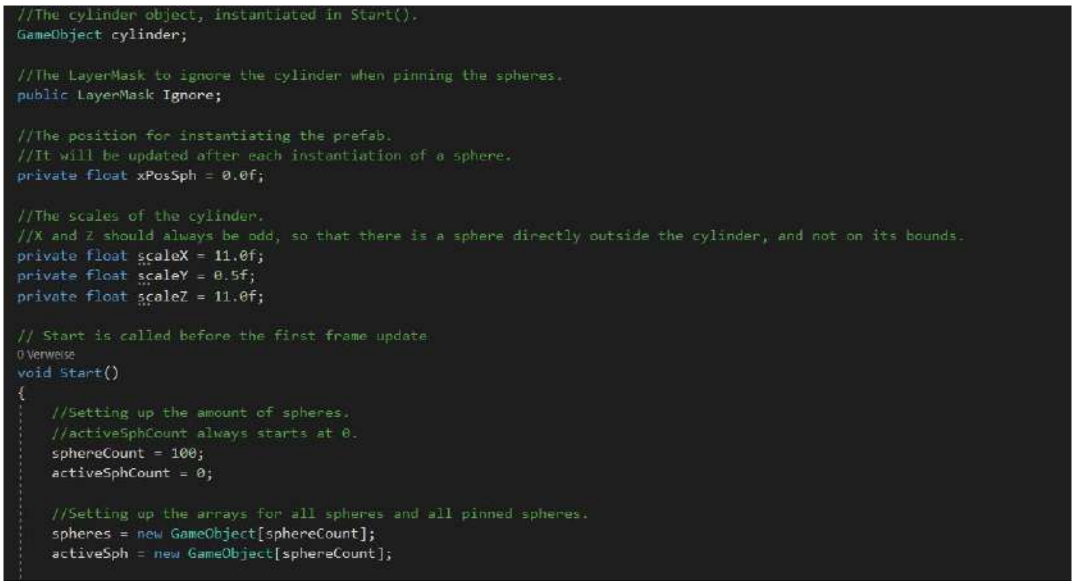
```
// Update is called once per frame
0 Verweise
void Update()
{
    //The GameController neither should be selectable in a logical sense nor has a Rigidbody attached.
    //Thus, it should not be selected as the active game object in the next step.
    if (Selection.activeGameObject != GameObject.FindWithTag("GameController"))
    {
        //The active (currently selected by user) game object is saved in the activeGO placeholder.
        activeGO = Selection.activeGameObject;

        //Its rigidbody is set to kinematic. Checks if Rigidbody is attached in order to avoid
        //NullReferenceExceptions.
        if (activeGO.GetComponent<Rigidbody>() != null)
        {
            activeGO.GetComponent<Rigidbody>().isKinematic = true;
        }

        //If the active object has been switched (this will lead to prevActiveGO being different from
        //activeGO for one frame), set the previously active game object to not be kinematic.
        if (activeGO != prevActiveGO)
        {
            prevActiveGO.GetComponent<Rigidbody>().isKinematic = false;
        }

        //All changes have been made, so prevActiveGO is set to the currently active game object
        //so that the right changes will happen whenever a different game object is made active.
        prevActiveGO = activeGO;
    }
}
```

Image A7: The second part of the KinematicController script, showing how the kinematic property is enabled and disabled immediately after change of active game objects.



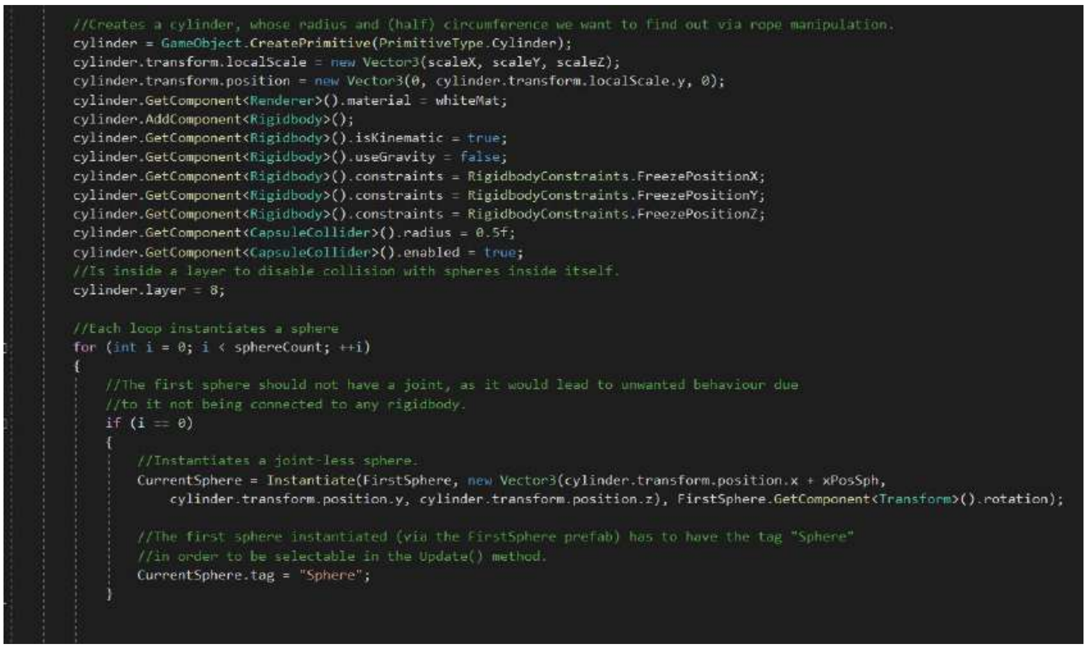
```
public class SquaringCircles : MonoBehaviour
{
    //The material for the cylinder.
    public Material whiteMat;
    //The material for the spheres.
    public Material redMat;
    //The material for "pinned" spheres.
    public Material greenMat;
    //The material to indicate the circumference.
    public Material orange;

    //The references to the sphere prefab.
    public GameObject Sphere;
    //The first sphere has no joint, so it has to have a seperate prefab.
    public GameObject FirstSphere;

    //Two placeholders for instantiating the prefab and connecting the joint to the
    //rigidbody of the previously instantiated sphere.
    public GameObject CurrentSphere;
    public GameObject prevSphere;

    //The array for all the spheres.
    public GameObject[] spheres;
    //The array for all pinned spheres.
    public GameObject[] activeSph;
    //How many spheres there are.
    public int sphereCount;
    //How many pinned spheres there are.
    public int activeSphCount;

    //The part of the rope beyond the rope parts within the cylinder/beyond the sphere pinned to indicate the radius (not including it).
    public GameObject[] shorterRope;
    //How many spheres there are before the one pinned to indicate the radius.
    private int ropeOffset;
    //The rope parts that make up the circumference of the cylinder (not including the second pinned to indicate the radius, but the third
    //pinned colliding with the second is included, so that you can better roll off the string later).
    public GameObject[] circumf;
```

Image A8: The first part of the variables for the SquaringCircles script, showing all Materials, GameObjects and placeholders, arrays and counters.



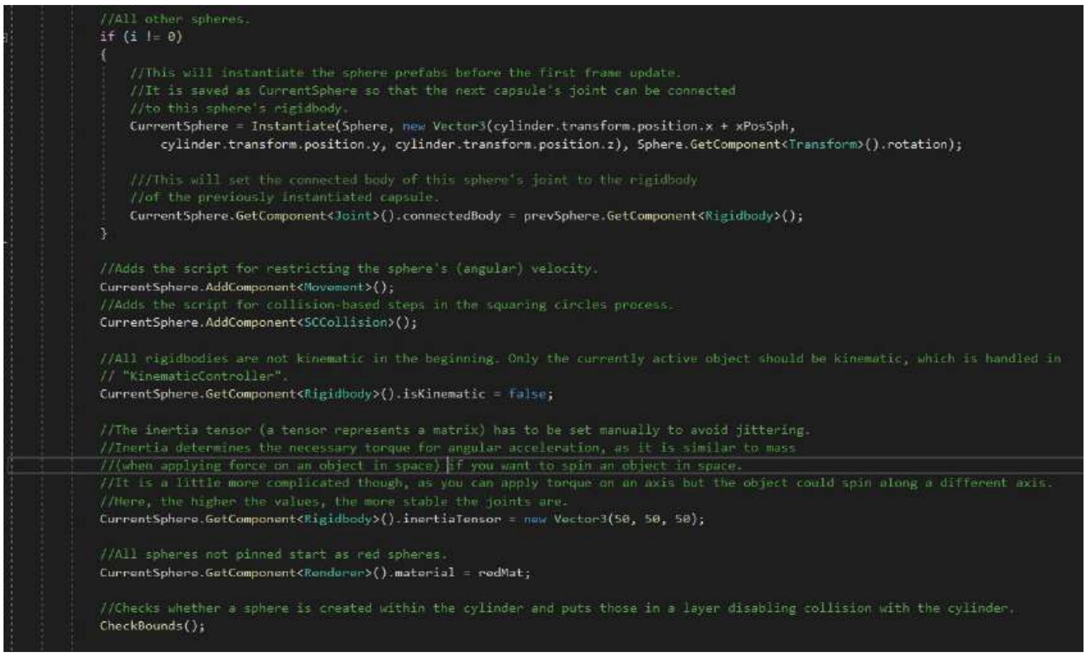
```
//The cylinder object, instantiated in Start().
GameObject cylinder;

//The LayerMask to ignore the cylinder when pinning the spheres.
public LayerMask Ignore;

//The position for instantiating the prefab.
//It will be updated after each instantiation of a sphere.
private float xPosSph = 0.0f;

//The scales of the cylinder.
//X and Z should always be odd, so that there is a sphere directly outside the cylinder, and not on its bounds.
private float scaleX = 11.0f;
private float scaleY = 0.5f;
private float scaleZ = 11.0f;

// Start is called before the first frame update
0 Verweise
void Start()
{
    //Setting up the amount of spheres.
    //activeSphCount always starts at 0.
    sphereCount = 100;
    activeSphCount = 0;

    //Setting up the arrays for all spheres and all pinned spheres.
    spheres = new GameObject[sphereCount];
    activeSph = new GameObject[sphereCount];
```

Image A9: The last variables for the SquaringCircles script and the setup within the Start() method.



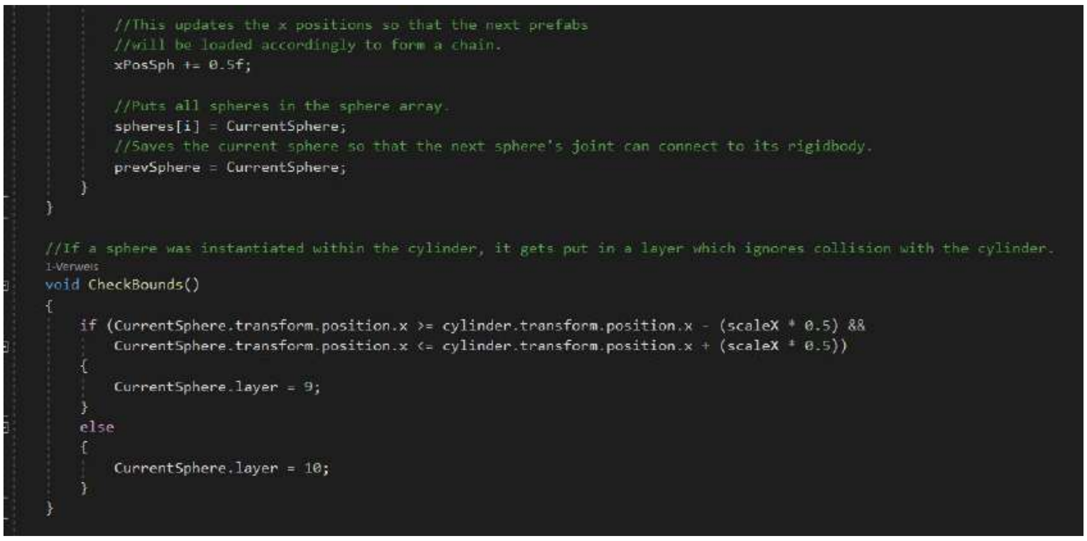
```
//Creates a cylinder, whose radius and (half) circumference we want to find out via rope manipulation.
cylinder = GameObject.CreatePrimitive(PrimitiveType.Cylinder);
cylinder.transform.localScale = new Vector3(scaleX, scaleY, scaleZ);
cylinder.transform.position = new Vector3(0, cylinder.transform.localScale.y, 0);
cylinder.GetComponent<Renderer>().material = whiteMat;
cylinder.AddComponent<Rigidbody>();
cylinder.GetComponent<Rigidbody>().isKinematic = true;
cylinder.GetComponent<Rigidbody>().useGravity = false;
cylinder.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionX;
cylinder.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionY;
cylinder.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionZ;
cylinder.GetComponent<CapsuleCollider>().radius = 0.5f;
cylinder.GetComponent<CapsuleCollider>().enabled = true;
//Is inside a layer to disable collision with spheres inside itself.
cylinder.layer = 8;

//Each loop instantiates a sphere
for (int i = 0; i < sphereCount; ++i)
{
    //The first sphere should not have a joint, as it would lead to unwanted behaviour due
    //to it not being connected to any rigidbody.
    if (i == 0)
    {
        //Instantiates a joint-less sphere.
        CurrentSphere = Instantiate(FirstSphere, new Vector3(cylinder.transform.position.x + xPosSph,
            cylinder.transform.position.y, cylinder.transform.position.z), FirstSphere.GetComponent<Transform>().rotation);

        //The first sphere instantiated (via the FirstSphere prefab) has to have the tag "Sphere"
        //in order to be selectable in the Update() method.
        CurrentSphere.tag = "Sphere";
    }
```

Image A10: SquaringCircles Start() method continued, showing the instantiation of the cylinder and the first sphere.



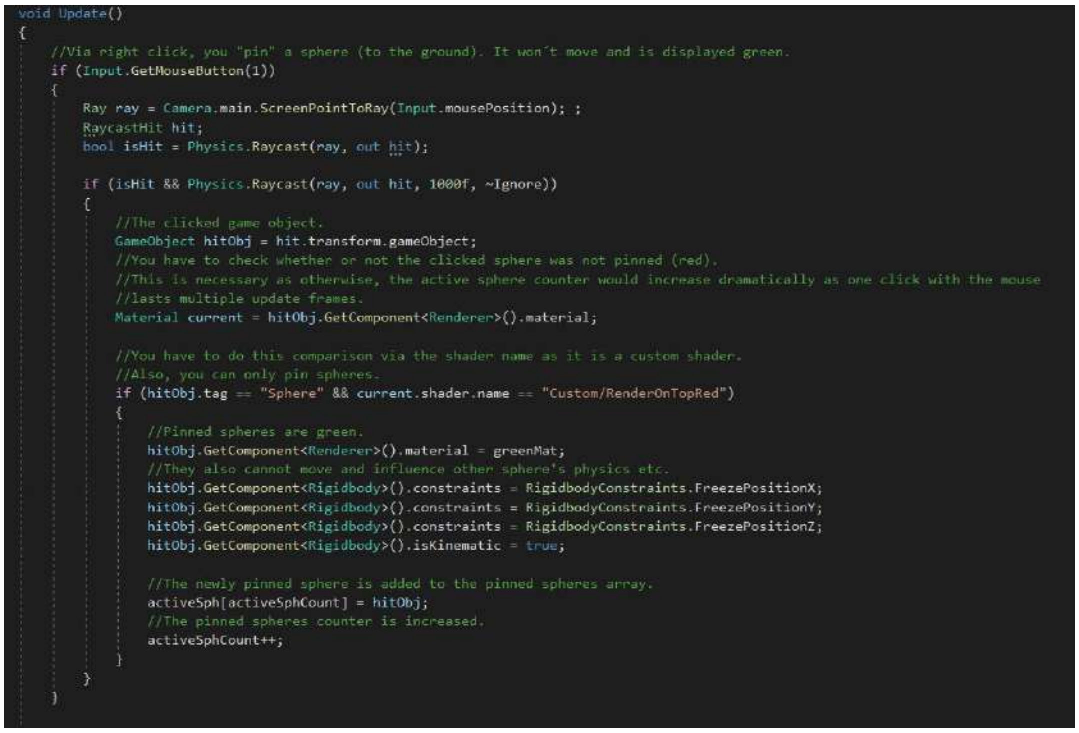
```
    //All other spheres.
    if (i != 0)
    {
        //This will instantiate the sphere prefabs before the first frame update.
        //It is saved as CurrentSphere so that the next capsule's joint can be connected
        //to this sphere's rigidbody.
        CurrentSphere = Instantiate(Sphere, new Vector3(cylinder.transform.position.x + xPosSph,
            cylinder.transform.position.y, cylinder.transform.position.z), Sphere.GetComponent<Transform>().rotation);

        ///This will set the connected body of this sphere's joint to the rigidbody
        //of the previously instantiated capsule.
        CurrentSphere.GetComponent<Joint>().connectedBody = prevSphere.GetComponent<Rigidbody>();
    }

    //Adds the script for restricting the sphere's (angular) velocity.
    CurrentSphere.AddComponent<Movement>();
    //Adds the script for collision-based steps in the squaring circles process.
    CurrentSphere.AddComponent<SCCollision>();

    //All rigidbodies are not kinematic in the beginning. Only the currently active object should be kinematic, which is handled in
    // "KinematicController".
    CurrentSphere.GetComponent<Rigidbody>().isKinematic = false;

    //The inertia tensor (a tensor represents a matrix) has to be set manually to avoid jittering.
    //Inertia determines the necessary torque for angular acceleration, as it is similar to mass
    //(when applying force on an object in space) if you want to spin an object in space.
    //It is a little more complicated though, as you can apply torque on an axis but the object could spin along a different axis.
    //Here, the higher the values, the more stable the joints are.
    CurrentSphere.GetComponent<Rigidbody>().inertiaTensor = new Vector3(50, 50, 50);

    //All spheres not pinned start as red spheres.
    CurrentSphere.GetComponent<Renderer>().material = redMat;

    //Checks whether a sphere is created within the cylinder and puts those in a layer disabling collision with the cylinder.
    CheckBounds();
```

Image A11: SquaringCircles Start() method continued, showing the instantiation of the other spheres and the call for the CheckBounds() method.



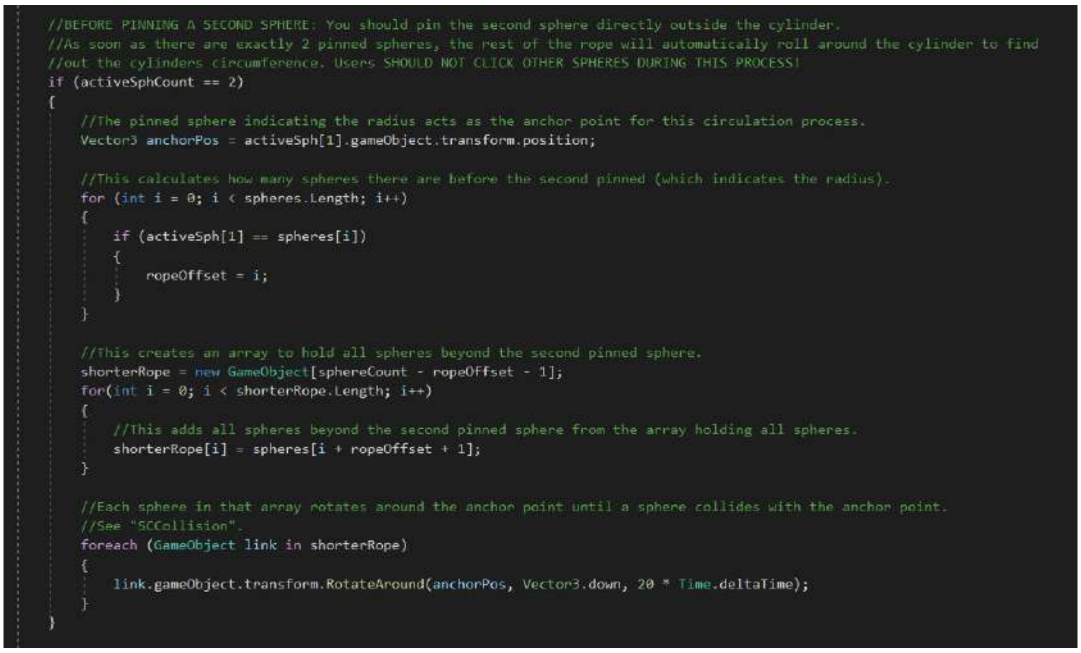
```
            //This updates the x positions so that the next prefabs
            //will be loaded accordingly to form a chain.
            xPosSph += 0.5f;

            //Puts all spheres in the sphere array.
            spheres[i] = CurrentSphere;
            //Saves the current sphere so that the next sphere's joint can connect to its rigidbody.
            prevSphere = CurrentSphere;
        }
    }

    //If a sphere was instantiated within the cylinder, it gets put in a layer which ignores collision with the cylinder.
    1-Verweis
    void CheckBounds()
    {
        if (CurrentSphere.transform.position.x >= cylinder.transform.position.x - (scaleX * 0.5) &&
            CurrentSphere.transform.position.x <= cylinder.transform.position.x + (scaleX * 0.5))
        {
            CurrentSphere.layer = 9;
        }
        else
        {
            CurrentSphere.layer = 10;
        }
    }
```

Image A12: The end of the Start() method of the SquaringCircles script, preparing the next loop, and the CheckBounds() method, checking whether a sphere is instantiated within the bounds of the cylinder or not.



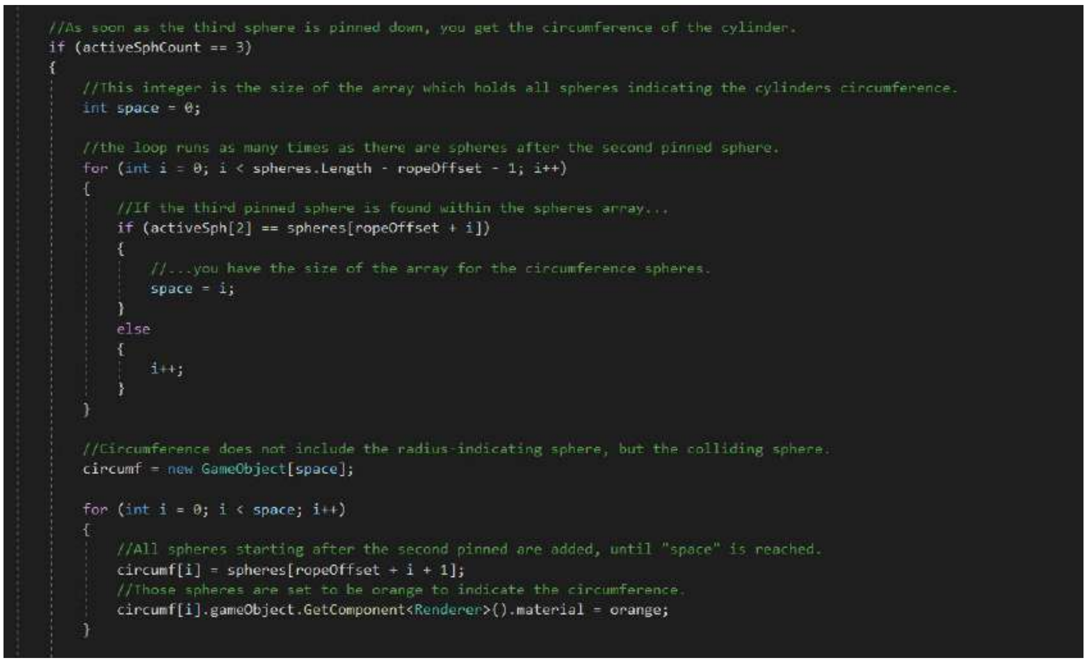
```
void Update()
{
    //Via right click, you "pin" a sphere (to the ground). It won´t move and is displayed green.
    if (Input.GetMouseButton(1))
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition); ;
        RaycastHit hit;
        bool isHit = Physics.Raycast(ray, out hit);

        if (isHit && Physics.Raycast(ray, out hit, 1000f, ~Ignore))
        {
            //The clicked game object.
            GameObject hitObj = hit.transform.gameObject;
            //You have to check whether or not the clicked sphere was not pinned (red).
            //This is necessary as otherwise, the active sphere counter would increase dramatically as one click with the mouse
            //lasts multiple update frames.
            Material current = hitObj.GetComponent<Renderer>().material;

            //You have to do this comparison via the shader name as it is a custom shader.
            //Also, you can only pin spheres.
            if (hitObj.tag == "Sphere" && current.shader.name == "Custom/RenderOnTopRed")
            {
                //Pinned spheres are green.
                hitObj.GetComponent<Renderer>().material = greenMat;
                //They also cannot move and influence other sphere's physics etc.
                hitObj.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionX;
                hitObj.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionY;
                hitObj.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionZ;
                hitObj.GetComponent<Rigidbody>().isKinematic = true;

                //The newly pinned sphere is added to the pinned spheres array.
                activeSph[activeSphCount] = hitObj;
                //The pinned spheres counter is increased.
                activeSphCount++;
            }
        }
    }
```

Image A13: The start of the Update() method of the SquaringCircles script, showing how the clicking of a sphere is detected and how it is “pinned down”.



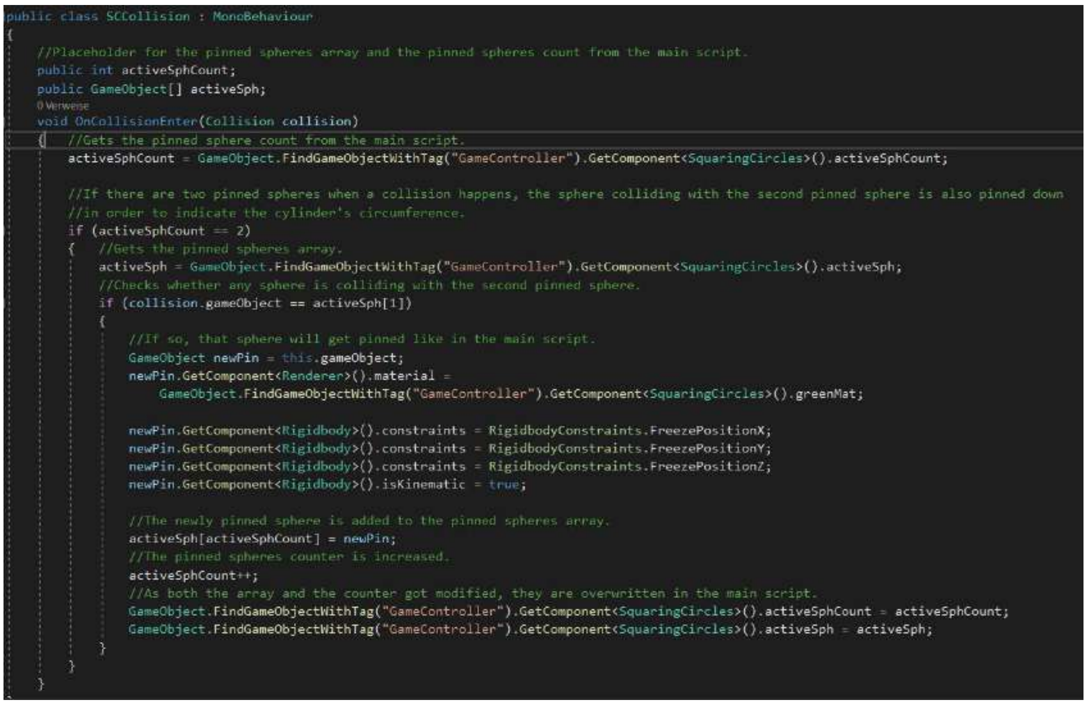
```
//BEFORE PINNING A SECOND SPHERE: You should pin the second sphere directly outside the cylinder.
//As soon as there are exactly 2 pinned spheres, the rest of the rope will automatically roll around the cylinder to find
//out the cylinders circumference. Users SHOULD NOT CLICK OTHER SPHERES DURING THIS PROCESS!
if (activeSphCount == 2)
{
    //The pinned sphere indicating the radius acts as the anchor point for this circulation process.
    Vector3 anchorPos = activeSph[1].gameObject.transform.position;

    //This calculates how many spheres there are before the second pinned (which indicates the radius).
    for (int i = 0; i < spheres.Length; i++)
    {
        if (activeSph[1] == spheres[i])
        {
            ropeOffset = i;
        }
    }

    //This creates an array to hold all spheres beyond the second pinned sphere.
    shorterRope = new GameObject[sphereCount - ropeOffset - 1];
    for(int i = 0; i < shorterRope.Length; i++)
    {
        //This adds all spheres beyond the second pinned sphere from the array holding all spheres.
        shorterRope[i] = spheres[i + ropeOffset + 1];
    }

    //Each sphere in that array rotates around the anchor point until a sphere collides with the anchor point.
    //See "SCCollision".
    foreach (GameObject link in shorterRope)
    {
        link.gameObject.transform.RotateAround(anchorPos, Vector3.down, 20 * Time.deltaTime);
    }
}
```

Image A14: The SquaringCircles' Update() method continued, showing how the rope is rolled around the cylinder.



```
//As soon as the third sphere is pinned down, you get the circumference of the cylinder.
if (activeSphCount == 3)
{
    //This integer is the size of the array which holds all spheres indicating the cylinders circumference.
    int space = 0;

    //the loop runs as many times as there are spheres after the second pinned sphere.
    for (int i = 0; i < spheres.Length - ropeOffset - 1; i++)
    {
        //If the third pinned sphere is found within the spheres array...
        if (activeSph[2] == spheres[ropeOffset + i])
        {
            //...you have the size of the array for the circumference spheres.
            space = i;
        }
        else
        {
            i++;
        }
    }

    //Circumference does not include the radius-indicating sphere, but the colliding sphere.
    circumf = new GameObject[space];

    for (int i = 0; i < space; i++)
    {
        //All spheres starting after the second pinned are added, until "space" is reached.
        circumf[i] = spheres[ropeOffset + i + 1];
        //Those spheres are set to be orange to indicate the circumference.
        circumf[i].gameObject.GetComponent<Renderer>().material = orange;
    }
```

Image A15: The end of the Update() method of the SquaringCircles script, showing how the circumference of the cylinder is indicated by the rope. (Afterwards, there would be commented-out code, intended for the next steps for squaring the circle. We did not implement the further steps as the vital step in regards to strong spatial cognition has been implemented and due to the timeframe of this thesis.)



```
public class SCCollision : MonoBehaviour
{
    //Placeholder for the pinned spheres array and the pinned spheres count from the main script.
    public int activeSphCount;
    public GameObject[] activeSph;
    0 Verweise
    void OnCollisionEnter(Collision collision)
    {   //Gets the pinned sphere count from the main script.
        activeSphCount = GameObject.FindGameObjectWithTag("GameController").GetComponent<SquaringCircles>().activeSphCount;

        //If there are two pinned spheres when a collision happens, the sphere colliding with the second pinned sphere is also pinned down
        //in order to indicate the cylinder's circumference.
        if (activeSphCount == 2)
        {   //Gets the pinned spheres array.
            activeSph = GameObject.FindGameObjectWithTag("GameController").GetComponent<SquaringCircles>().activeSph;
            //Checks whether any sphere is colliding with the second pinned sphere.
            if (collision.gameObject == activeSph[1])
            {
                //If so, that sphere will get pinned like in the main script.
                GameObject newPin = this.gameObject;
                newPin.GetComponent<Renderer>().material =
                    GameObject.FindGameObjectWithTag("GameController").GetComponent<SquaringCircles>().greenMat;

                newPin.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionX;
                newPin.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionY;
                newPin.GetComponent<Rigidbody>().constraints = RigidbodyConstraints.FreezePositionZ;
                newPin.GetComponent<Rigidbody>().isKinematic = true;

                //The newly pinned sphere is added to the pinned spheres array.
                activeSph[activeSphCount] = newPin;
                //The pinned spheres counter is increased.
                activeSphCount++;
                //As both the array and the counter got modified, they are overwritten in the main script.
                GameObject.FindGameObjectWithTag("GameController").GetComponent<SquaringCircles>().activeSphCount = activeSphCount;
                GameObject.FindGameObjectWithTag("GameController").GetComponent<SquaringCircles>().activeSph = activeSph;
            }
        }
    }
```

Image A16: The SCCollision script, handling all collisions between spheres. If a sphere collides with the second pinned sphere, it is also pinned down in order to indicate the cylinder’s circumference via the rope.