Rube Goldberg Project

Julia Duong - jduong@kth.se
Leif Thysell Sundkvist - leifsun@kth.se
Malin Larsson - malilars@kth.se

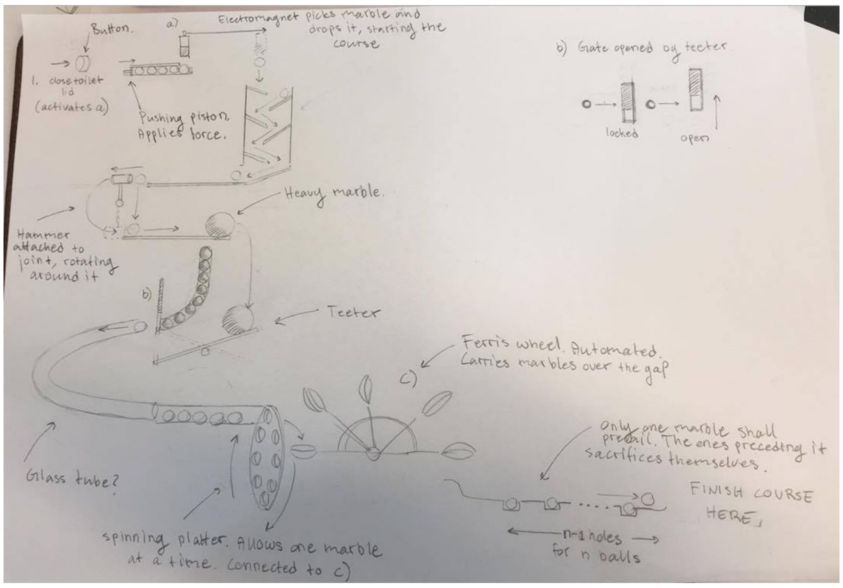
1. Background

In a dictionary you can find the word “Rube Goldberg” and the description “(of machines and devices) having a very complicated design, especially when used to perform a very simple task; not practical”[1]. This adjective is named after Reuben Lucifer Goldberg who was a mining engineer from San Francisco born in 1883. He did not became famous for his engineering but for his cartoons of advanced, complicated and pointless machines for solving simple problems[2].

One reason that Rube Goldberg became so successful and popular might have been because he used his engineering skills to do something totally different that people found amusing, and still do. A typical “Rube Goldberg”-machine make use of physics and mechanics with e.g. gear wheels, dominoes and rolling marbles acting together to in the end accomplish a simple task like opening a door or pressing a button.

1.1 Idea and vision

In this project we have constructed and built a “Rube Goldberg”-machine using Blender and Unity. It is well suited as a project since it makes it possible to implement a sequence of events caused and depending on different functions and physics.


Button.
a)
Electromagnet picks marble and drops it, starting the course
b) Gate opened by teeter
1. close toilet lid (activates a)
Pushing piston. Applies force.
locked
open
Heavy marble.
Hammer attached to joint, rotating around it
b)
Teeter
Ferris wheel. Automated. Carries marbles over the gap
c)
Only one marble shall prevail. The ones preceding it sacrifices themselves.
Glass tube?
FINISH COURSE HERE.
spinning platter. Allows one marble at a time. Connected to c)
n-1 holes for n balls

The picture illustrates the first draft of the machine and which functions that had the highest priority in the beginning, when they all were implemented new features and functions were implemented.

We had the idea that our machine should perform a task that everybody is doing everyday and that is how the idea to push down the toilet lid was born.

[1] Oxford Learners Advanced Dictionary
[2] Wonder Polis

This Rube Goldberg Machine (RGM) is helping humans to push down the toilet lid without getting their hands dirty.

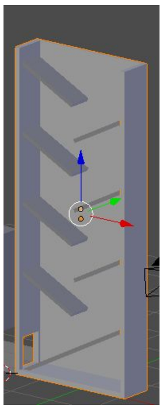
2. Functions and features

In the following section the functions and features in the RGM will be described in detail. The features will be described considering both performance (i.e. what they are used for) and implementation. The features are ordered by appearance in the RGM.

2.1 Marbles

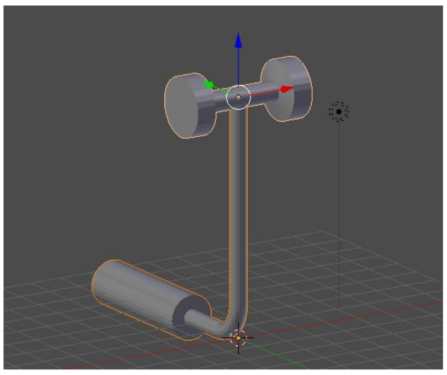
Performance: Through the whole RGM marbles are used and they are rolling with the help of gravity and trigger different events.
Implementation: Unity's Sphere collider and rigid body. (Subject to magnetic forces, as exerted by the Magnet objects in the RGM). The marbles serving as magnets will receive the corresponding *ElectroMagnetController,* allowing them to communicate with other electro magnet objects within the scene, as well as a "Magnet" shared between all magnet objects so that they can find and identify each other.

2.2 Skyscraper



Performance: Used as a steep path with obstructing shelves for the marble to pass through.
Implementation: Unity's mesh collider.

2.3 Hammer



Performance: When the marble hit the hammer the hammer is affected by the force and swing down and hit a plate. When the plate is pushed in a bunch of marbles are released through a hole in the plate.
Implementation: Unity's collider, rigid body and joints. The swing will be attached to a single point using a joint component, and this point will serve as an anchor. The resting position of the hammer will be sensitive to collision, as a very low force is required to set the hammer's swing in motion.

2.4 Gears, cogwheels and pistons

`Assets/Scripts/Gears`

Performance: The marbles are passing through a disk with holes in and they are pushed by a piston. When they have passed the disk they are transferred by a wheel with small bins that can hold one marble each.

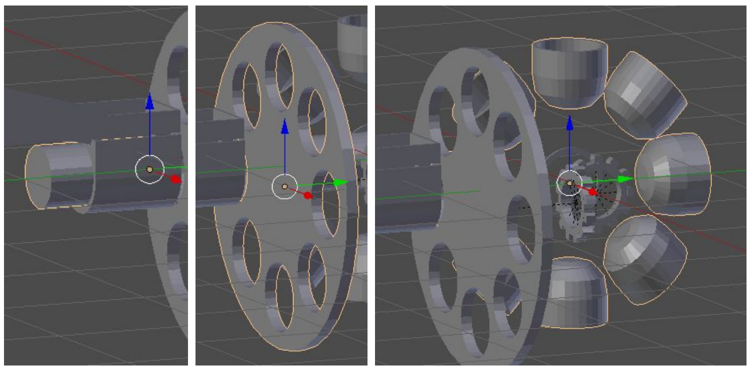
Implementation: Unity's mesh colliders as well as a *CogWheelController* script controlling each cogwheel's rotation and a *PistonController* script controlling the piston's motion. This is done in three phases, though all of the components in the three phases share the same periodical movement in a synchronized fashion;

1. The first phase is dominated by the piston, which rocks back and forth using the *PistonController* script. The piston has a mesh collider attached to it, and pushes marbles through the disk with holes in phase 2.

2. The second phase revolves around the disk with holes, connected to a cogwheel rotating with the same period as the piston from the previous phase. Each hole is synchronized to the maximum amplitude of the piston; this means that when the piston is pushed in at a maximum, a hole in the disk will be placed just in front of the hole so that a marble is sure to be pushed right through the hole.

3. The third phase, like the second, contains a rotating cogwheel but which has a "ferris-wheel"-esque component attached to it. This ferris wheel contains cup-formed meshes with mesh colliders and, in the same manner as how the disk with holes was synchronized with the piston, this component is too. Each cup is synchronized with the disk with holes, and so whenever a ball passes through a hole in the disk it will proceed to fall into one of the cups.

   The following images show each phase separated into individual images; the main components of each phase are outlined in orange.

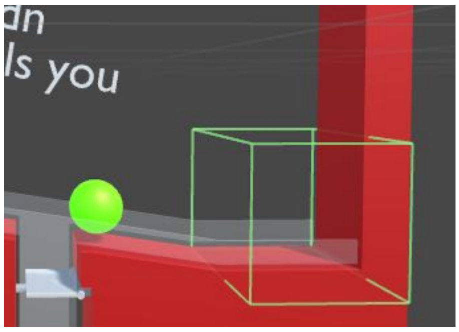


2.5 Shifted gravity

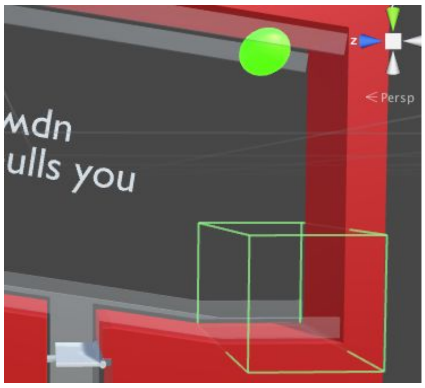
Performance: When a marble passes a certain point in the RGM the gravity is shifting and the marble is dragged to the ceiling where it is acting exactly the same way as it was doing before the gravity was shifted.

Implementation: The shifted gravity field is an *Empty* object with a box collider set to *Trigger.* This means that it will not collide or repel any colliding objects, but will instead send a trigger signal which can be received using the *OnTriggerEnter()* routine provided by Unity. Whenever a marble is detected to be passing through a field such as this one, we assign the marble the *GravityShift* script. Any object with the *GravityShift* script will receive a force upon each frame that is minus one times the current gravity value defined for the scene. In order for this to work, we tell the object's rigid body not to use Unity's gravity, but instead to use our overriden one, because we never found out how to modify Unity's own value for gravity. If we could, we would have just set that to -1. This means that each object with this script will be pushed upwards with a reversed gravitational force, and thus it will appear as though gravity has shifted direction.

The following image shows a marble just before it is about to enter the gravity field. Here, its gravity is as it is by default.



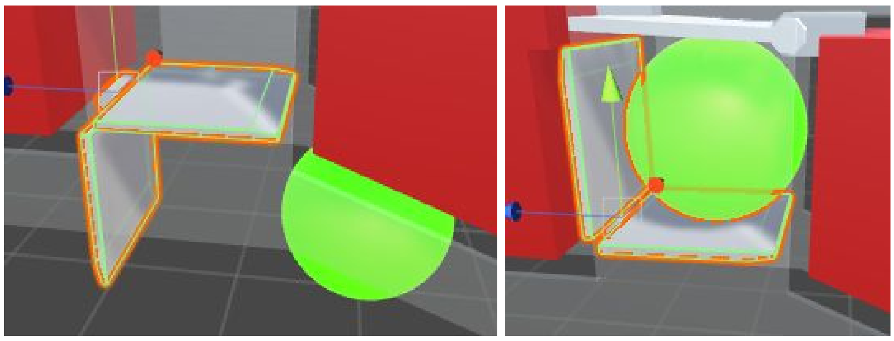
As soon as it triggers the gravity field, its *GravityShift* script is enabled and it receives the upward force, pushing it upwards.



2.6 Flip bridge

Performance: When a marble role in the door is flipped and the marble will fall down. The flipped door will make a new way for the second marble.

Implementation: (Note that this component is used when the marbles has their gravity flipped) The flip door is triggered when a object collide with it and a script called *BridgeController* that flips the door 45 degrees over time depending on the given rotational speed is activated. The following two images show the flip bridge the moment just before a marble collides with it, and the moment after when the bridge has flipped 45 degrees, forming a "bridge" so that another marble may pass.

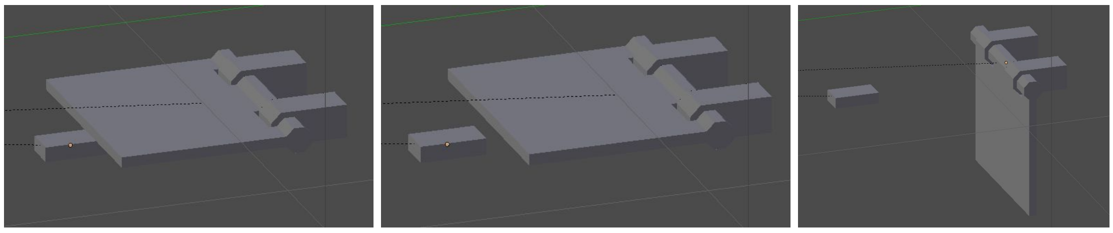


2.7 Trap door

Performance: The trap door is activated when a button is pushed and there are two trap doors in the RGM.

Implementation: Each trap door consists of a platform blocking the path and a hinge that serves as the anchor for the platform to rotate around using a hinge joint. Furthermore, each trap door also has a lock mechanism consisting of a single cube object that is placed on the wall opposite of the one that the trap door is attached to, which prevents the trap door from opening by using a simple box collider and rigid body component.

Each lock has a script called *LockController* attached to it which, when enabled, makes the lock move a short distance away until no longer colliding with the trapdoor, allowing it to open. This script is currently enabled in two cases in the scene; more specifically by two buttons. Whenever these buttons are pressed, they will activate their respective lock mechanism that they have been assigned, after which the corresponding trap door will open and allow objects to pass through.



2.8 Domino

Performance: A trap door that falls down will activate the first domino brick and the rest is falling down one by one; the domino effect.

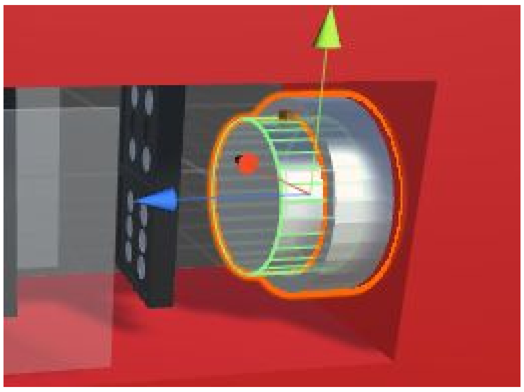
Implementation: Each domino brick model has a simple Box collider attached to it, as well as a rigid body.

2.9 Button

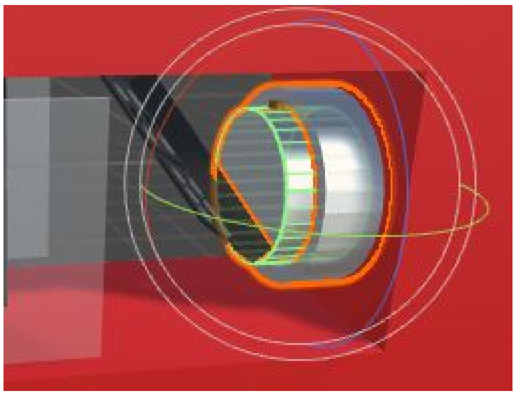
Performance: The buttons have a collider and will act as a trigger for different functions in the RGM.

Implementation: Whenever a button collides with another object, Unity's *OnCollisionEnter()* routine will be called whereas the mechanism will be called on using the *ButtonController* script. Two of the buttons will also start/stop the magnet in the scene from moving, as demonstrated by the provided simulation in the project, using the *MagnetActivateController* script which enables a given magnet's *Waypoint* and *ElectroMagnetController* scripts.

The following image shows the button the moment before a domino brick will fall over and push it.



The domino brick then falls and pushes the button inwards, which, upon collision and as according to the *ButtonController* script, activates its corresponding mechanism.

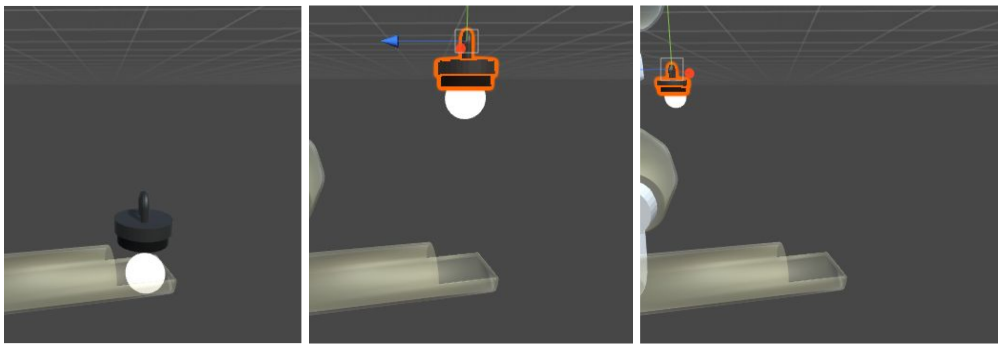


2.10 Magnet

Performance: A magnet pick up a marble and transport it to a specific point where it is dropped.

Implementation: Script based physics, script based magnetic force. The script for the magnet is taking use of the formula for electro magnetic force and is enabled and disabled with the helt of buttons that triggers the events. To make the magnet attract a marble the marble has been given magnetic properties such as permeability and magnitude. The Magnet has a stronger magnitude and therefore attracts the marble. A script that takes use of the build in transform.position moves the magnet according to given coordinates in a list.

The following images shows the magnet before its electro magnet script is turned on, and before it starts moving, then how it looks when its magnet properties are enabled and how it picks the marble up and starts moving away with it.



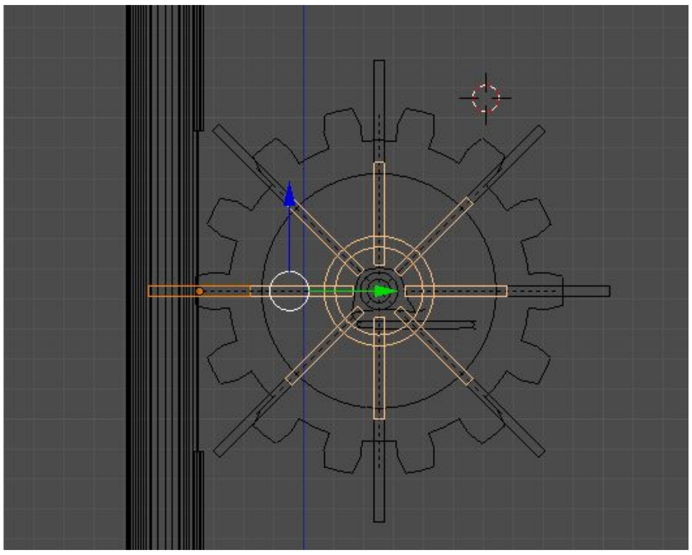
Constraints that has been applied to all of the components in the implementation are that no air resistance or friction exists.

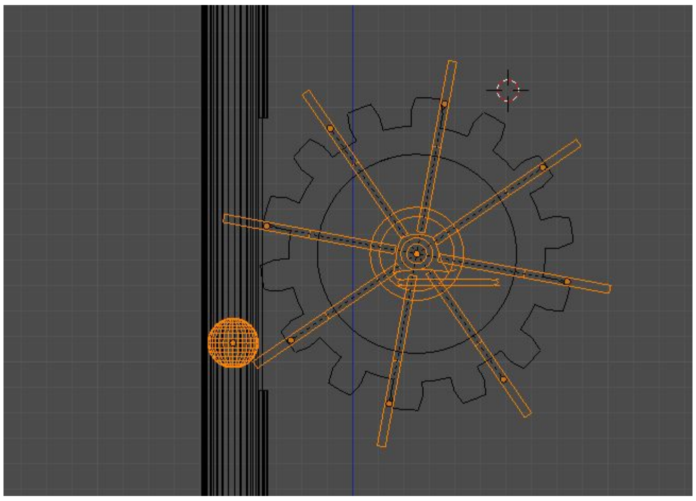
2.10 Toilet

Performance: A toilet with a gear and cylinder attached to it. When a marble drops through the cylinder and hits the rotating handle on the cogwheel, the cogwheel starts to spin after which it turns the toilet lid and closes it shut.

Implementation: Uses Unity's mesh colliderdynamics. The cylinder has a hole in it in which the cog wheel is inserted in, as shown by the below image.

Mesh: Aside from the other meshes in the simulation, this mesh was the only one that we borrowed and downloaded from the *Blender 3D repository*.

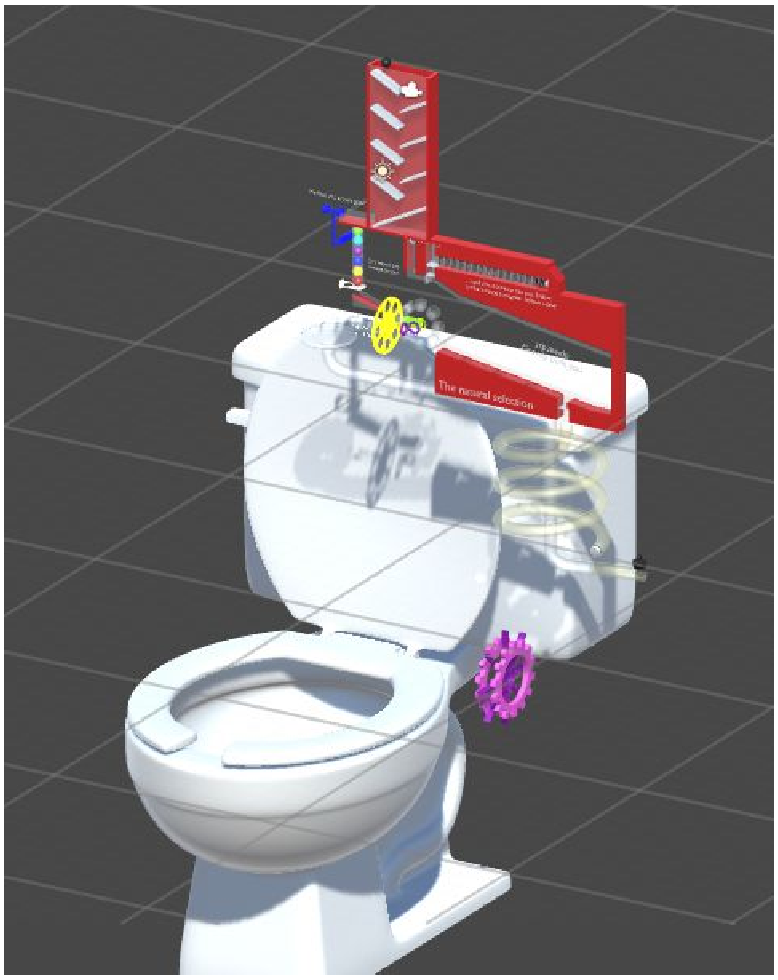




When the marble, with its sphere collider and rigid body, lands upon the rotating handle of the cog wheel, the same script *BridgeController* as demonstrated in the *Flip Bridge* above, is activated, and the cogwheel rotates for a set amount of angles with a given rotational speed. More exactly, it rotates for 90 degrees so that it appears as though the toilet lid closes shut, and then it stops.

3. Result

Provided below is an image of the final version of the RGM. We wanted it to be quite compact, and to appear as though it was a real component that could be found on a real toilet. When all the functions and features were assembled and put together, we just had to make some minor adjustments in order to make it work as originally intended.



3.1 Degree of simulation

In the end, we managed to simulate the RGM as according to the sketch made in the earliest phase of the project. In order to include as many features and functions as possible, we chose to omit certain specifics such as friction between materials and air resistance. The simulation was still realistic, and using Unity provided several benefits, one of them being that it was easy to create the meshes in Blender and import them straight into Unity, and another was that running a simulation in Unity results in a deterministic behavior; however, as one starts using the Unity physics system, one may see that this deterministic behaviour ceases. Unity's physics relies on how fast the calculations are made and how many calculations that can fit into one frame of the simulation, and sometimes an object with a rigid body, for instance, may produce different behaviour when compared to the round before. Despite this, the behaviour was enough deterministic to make the simulation very easy to test; just edit the transforms,

the components and the general layout of the RGM and press the play button. If something goes wrong, adjust it, and play the simulation again.
The physics implemented in this RGM was fairly simplistic. Once again, in order to implement as many exciting features as possible and to make the RGM long and entertaining enough (we didn't want it to contain too few features), we decided to just use the basic physics for those physicalities that we decided to use. For instance, magnets may attract one another with forces that are much more detailed in reality than what we decided to implement; we just used the basic formula that pulls objects defined as magnets towards each other, without thinking of air resistance or friction etc.

3.2 Group composition and member contribution

The group has been highly engaged throughout the whole of the project, from the very start. A group size of three was a good number, because it allowed for everybody to partake in all of the tasks for the project whilst still maintaining good communication within the group. Some differences existed regarding software knowledge (Unity and Blender). Leif had more knowledge in these two programs and therefore held the main responsibility for the modelling and tutoring the other members, which was more efficient concerning time constraints, because that way was more effective than having to spend hours learning the mechanics and basics of Blender and Unity elsewhere (watching tutorials online, for instance).

Regarding the design process of the RGM in general, every group member has participated and contributed and produced equally valuable ideas and thoughts. We had many brainstorming sessions where we sat down and spawned ideas together, and drew various ideas of the features to be used in the final version of the RGM.

5. Reflections and Analysis

It turned out that this project was a really good project for exploring and learning unity and blender. It has been possible to extend ideas and to change direction along the way without deviating from the goal. It is also a project that can be extended after this course because there is so much more things that are possible to do. A Rube Goldberg Machine is modular, and it is easy to come up with new ideas and inventions and, since Unity is very simple to learn and use, to add these to the simulation.

In this project there is only a few physics that has been implemented from scratch, and this was also a part of the plan to see what was possible to do with unity. What has been discovered is that you can do a lot using only the built in physics and features in unity and where it gets hard with the physics it is always a way around it to make it look realistic. The few things that was implemented from scratch was not difficult at all to do. There were only a few lines of code and if you know how a force is acting on a object it is just to implement the physical law. To solve a lot of problems we realised that there are a lot of unity forums, and a lot of help out there. It took a while to navigate and learn how to find different scripts, but after a while we realised that the limit is what we allow it to be.

5.1 Future plans and possibilities

Since RGM's are highly modular, i.e. it is very easy to create and add any modules or features of one's choice, this means that in the future this project could be extended and the RGM course that we made could be made even longer and contain more exciting function and features.

One thought is that this could make a nice example of what is possible to do in blender and Unity in a short time to inspire young people to start learning the different softwares. In the 90th century, when Rube Goldberg was active, his drawings inspired young people and made them extend their imagination and views. Maybe these kind of things can do the same today, who knows?

6. References

Oxford Learners Advanced Dictionary, http://www.oxfordlearnersdictionaries.com/definition/english/rube-goldberg?q=Rube+Goldberg, [2017-02-14]

Wonder Polis, 2014–2017 National Center for Families Learning, http://wonderopolis.org/wonder/what-is-a-rube-goldberg-machine, [2017-02-14]

Unity, Tutorials, https://unity3d.com/learn/tutorials/topics/physics, [2017-02-14]

Unity, Manual, https://docs.unity3d.com/Manual/class-BoxCollider.html , [2017-02-17]

Rune Alphonce, Lars Bergström, Per Gunnvald, Erik Johansson, Roy Nilsson, *Heureka! Fysik Kurs 2 Lärobok,* 2012