Modelling Project, TNM085

Simulating and Animating the Domino Effect

Pontus Arnesson
Isak Engström
Vera Fristedt Andersson
Adam Morén
Christoffer Paulusson

August 20, 2020

Abstract

Simulations and animations are a big part in almost every area of science. Constructing models of reality and studying the properties of real-life physical systems is therefore a major part of science. In computer science and technology, the behaviour and visualization of objects is of great importance when creating a virtual environment. Therefore, it is interesting to analyze the process of simulating and animating a system and to see what models that can be used for this purpose. The domino effect includes several physical laws hence a difficult system to simulate. The problem when simulating and animating a model is to generate an approximate solution that is close enough to reality and its laws of nature. When simulating a model, you start from the mathematical description and use numerical methods to generate the approximate solution. Which method is used depends largely on the model itself. The question is which simplifications can be made without loosing important aspects of the system. The approach for making progress in this project was to choose one simulation method, one animation method and to then make use of mathematical models obtained by hand and from documentation. Different applications for simulating and animating has not been taken into consideration. Instead, the extent of the work was comparing and analyzing the results of the variables within the chosen applications. The final result of the project is an animation of the domino effect that can be played and viewed simply by running a finished program. The conclusion of the project is that even though simplifications of the model are made, a reasonable result can be obtained.

Contents

List of Figures

Chapter 1

Introduction

This chapter covers an introduction to the project with a background, aim and project issues.

1.1 Background

The instructions for the project are to find a physical system to animate. The system has to be dynamic which implies that there has to be at least one differential equation that describes the system. The project should start off with a simple model and work towards a more complex final product. Before animating the system at hand, a simulation of the system has to be made using numerical methods.

1.2 Aim

The aim of this project is to simulate and animate the domino effect. To start off, a simple model will be made with only one domino brick falling over. After this, a more complex model will be implemented with several domino bricks falling on one another in order to create the domino effect. In both cases an impulse will be applied towards the first domino brick in order to make it fall over. The numerical method that will be implemented is Euler and the system will be animated in the game engine *Unity*.

1.3 Project Issues

- What makes Unity useful when animating simulations of systems?
- What simplifications can be made, when simulating the domino effect, that still give a reasonable result?
- What further studies and improvements can be made if more time is given?

Chapter 2

The Physical System

The physical system that is to be simulated and animated is described in this chapter.

2.1 One Domino

The simple model of the system is where one domino is to fall over. This is done when an impulse is applied to the object which makes the brick topple. A force hits the object making the brick rotate about its pivot point until it hits the ground. The object should however only topple when the force is large enough to make the brick go past its toppling angle. If the force is not large enough, the brick should return to its original standing position.

2.1.1 Physics

A domino brick of the following dimensions were created: $height = 4.353cm$, $width = 2.156cm$, $thickness = 0.679cm$. In Fig. 2.1 a toppling domino brick is shown from the side with the thickness and height visible. It is shown that when a force *F* hits the brick, the brick starts rotating with the angular velocity ω about its pivot point (PP). The variable *r* denotes the distance from where the force hits the brick to the pivot point. The variable *l* is the distance between the center of mass (CM) and the pivot point. The red arrow shows a distance *d* which is the distance between the pivot point and the projection of the center of mass on the ground. How much the brick has toppled is shown with the angle θ. The toppling angle is the angle which places the center of mass right above the pivot point along an axis that is perpendicular to the ground.

2.1.2 Mathematical Model

In order to simulate the falling domino, a differential equation is needed that describes the movement of the system. The relationship between the force hitting the domino brick and the resulting torque can be described by (2.1) [1].

$$\tau = Fr \tag{2.1}$$

Where *F* is the force and *r* is the distance from the pivot point to where the force hits the object as seen in Fig. 2.1. However the gravitational force also has an impact on the object which has to be taken into account. The relationship between the gravitational force and the resulting torque can be described by (2.2) [1].

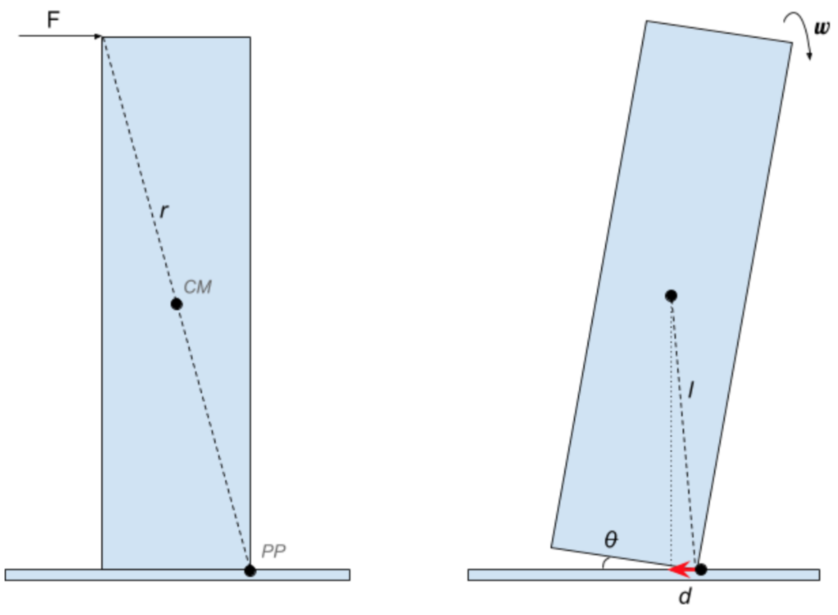


Figure 2.1: The physics of one toppling domino brick

$$\tau_g = F_g l = -mgd \tag{2.2}$$

Where *m* is the mass of the domino brick, *g* is the gravitational acceleration and *d* is same distance as mentioned before and as shown in Fig. 2.1.

From τ and τ_g the angular acceleration α of the object can be found in (2.3) [1].

$$\alpha = \frac{\tau_{tot}}{I} = \frac{Fr + mgd}{I} \tag{2.3}$$

The moment of inertia of the object is represented by *I* and it is given by (2.4). The parameters *H* and *T* indicate the height and thickness respectively [1].

$$I = \frac{m(H^2 + T^2)}{12} \tag{2.4}$$

2.2 Several Dominoes

A more complex model of the physical system is where several domino bricks are standing up close to each other. The first brick has an impulse applied to it which makes it topple and fall onto the next domino brick, causing that domino brick to fall onto the next and so on. This is what is called the domino effect.

2.2.1 Physics

The first domino brick in this system has the same physics as mentioned in section 2.1.1. However, for the remaining bricks the physics can be shown with Fig. 2.2. A brick falls onto another giving it a push to fall over in the same way as seen in Fig. 2.1. By knowing how far apart the bricks are standing from each other it is possible to find the angle that the first brick has tilted before it hits the second brick. From this it is possible to find where the first brick hits the second one, giving a different distance *r* than what was seen when studying the simple system with only one domino brick.

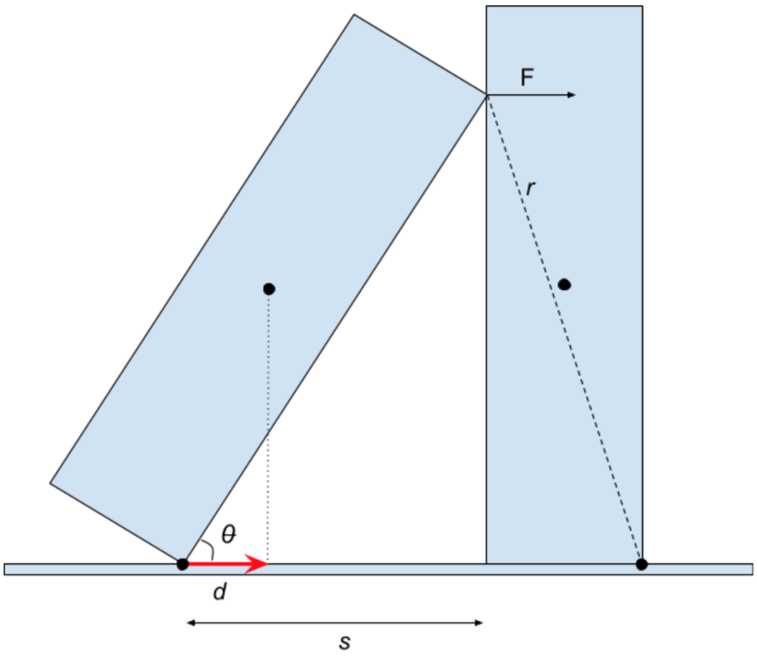


Figure 2.2: The physics of two toppling domino bricks

2.2.2 Mathematical Model

The mathematical model is simplified by assuming that when a domino brick hits another brick, the collision can be defined as an impulse. By assuming this, the force that the first brick would have on the other brick at all times does not have to be calculated, hence calculations will be made easier. The force in the impulse can be found by using (2.5) [1].

$$FH = \tau = \frac{dL}{dt} = I\alpha \Leftrightarrow F = \frac{I\alpha}{H} \tag{2.5}$$

The force *F* is found with the help of the objects angular momentum *L*. The moment of inertia *I* is the same as before and the height *H* is the distance from the pivot point of the first brick to where it hits the second brick (as seen in Fig. 2.2).

When a brick has been hit with an impulse from a previous brick, the movement continues in the same way as was described and calculated in section 2.1. However in this case, the force and the distance *r* are different.

2.3 Simplifications

For the modelling of the system some simplifications have been made in order to make it easier to implement. It is assumed that each brick is locked around its pivot point. This hinders them from sliding across the ground or against other bricks. Another consequence of the brick being secured about the pivot point is that the rotation of each brick is two-dimensional.

The collisions between the bricks are assumed to be elastic. Which implies that when a brick hits another, an impulse is applied to the second brick and not a constant force throughout the fall, as discussed in section 2.2.2.

Chapter 3

Simulation

This chapter describes how the simulation of the physical system was implemented.

3.1 Euler

The Euler method was used to solve the *ordinary differential equations* (ODE) which described the falling dominoes. It can be hard and even impossible to retrieve an exact solution to differential equations representing real life physical scenarios, therefore, numerical methods are needed. The method took the *initial value problems* (IVP) and generated approximated solutions to the ordinary differential equations, (3.1) and (3.2). Where $\alpha(n)$ is the angular acceleration, $\omega(n)$ is the angular velocity and $\theta(n)$ is the angle. The accuracy of the method is dependant on its step size, indicated by h. The global error is proportional to h and the local error is proportional to h^2, this means that a smaller h gives a more exact value [2].

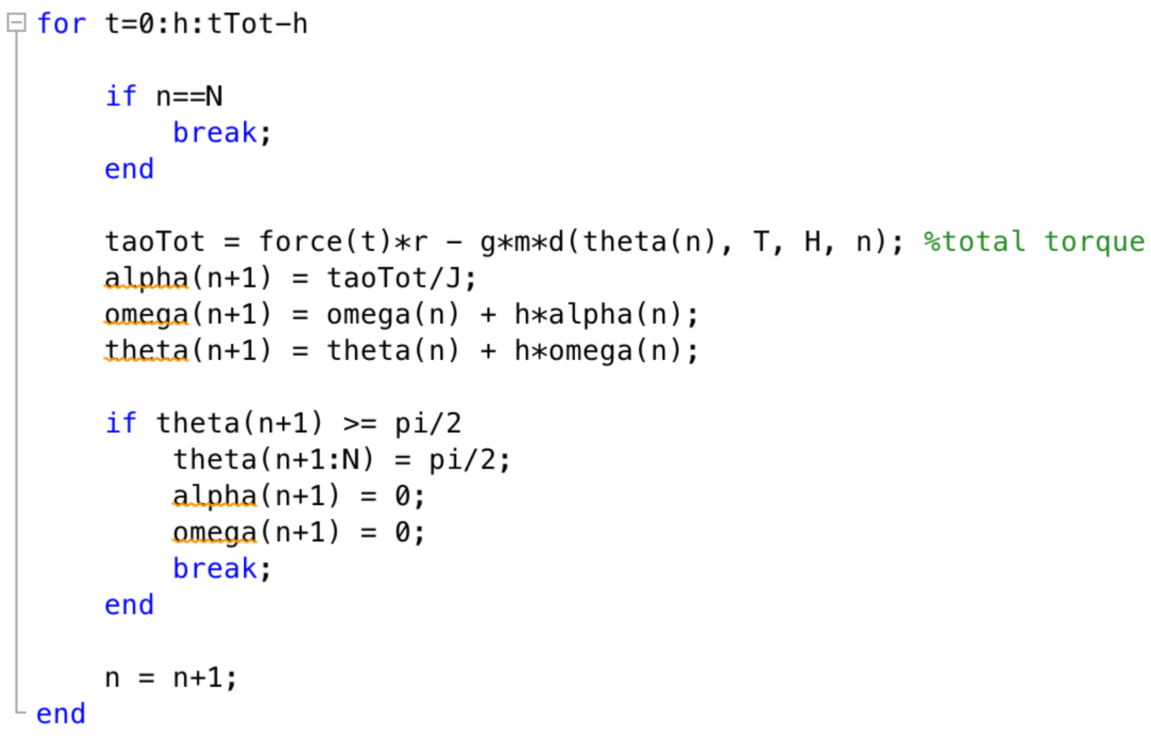
$$\omega(n+1) = \omega(n) + h\alpha(n) \tag{3.1}$$

$$\theta(n+1) = \theta(n) + h\omega(n) \tag{3.2}$$

3.2 Implementation

MATLAB [3] was used to implement the mathematical model into a simulation as shown in Fig. 3.1.

In Fig. 3.1, *alpha* represents the angular acceleration, *omega* is the angular velocity, and *theta* is the angle. A step size *h = 0.0005* was used for the implementation.



```
for t=0:h:tTot-h

    if n==N
        break;
    end

    taoTot = force(t)*r - g*m*d(theta(n), T, H, n); %total torque
    alpha(n+1) = taoTot/J;
    omega(n+1) = omega(n) + h*alpha(n);
    theta(n+1) = theta(n) + h*omega(n);

    if theta(n+1) >= pi/2
        theta(n+1:N) = pi/2;
        alpha(n+1) = 0;
        omega(n+1) = 0;
        break;
    end

    n = n+1;
end
```

Figure 3.1: Euler implemented in MATLAB

Chapter 4

Animation in Unity

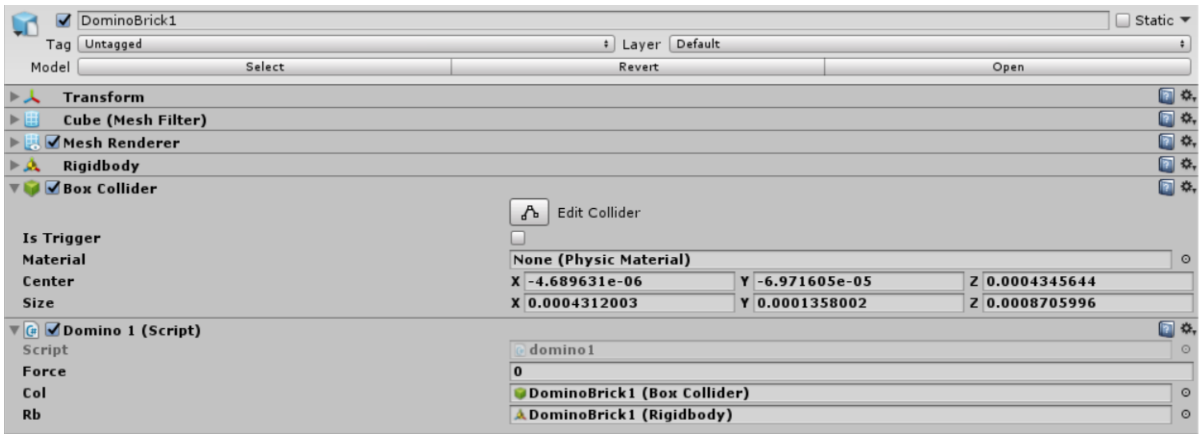
To animate the falling dominoes and their physical relationship, the game engine *Unity* [4] was used. In Unity, all kinds of content begin with a *GameObject*. The behavior of a GameObject is controlled by *components* with a number of different *variables*. The variables are manipulated by *scripts* written in the programming language *C#* and thus physics can be applied to the GameObjects [5]. This chapter describes how the animation was implemented.

4.1 GameObjects

The objects involved in the animation are the domino bricks. One brick was created according to the dimensions based on a standard domino brick. This brick was then added as an *asset*, which could then be used to create all GameObjects (the domino bricks) in the scene.

4.2 Components and Variables

Several components were added to the domino GameObject. These can be seen in Fig. 4.1.



✓ DominoBrick1 ☐ Static
Tag: Untagged | Layer: Default
Model: Select | Revert | Open

Transform
Cube (Mesh Filter)
✓ Mesh Renderer
Rigidbody
✓ Box Collider
Edit Collider
Is Trigger: ☐
Material: None (Physic Material)
Center: X -4.689631e-06 Y -6.971605e-05 Z 0.0004345644
Size: X 0.0004312003 Y 0.0001358002 Z 0.0008705996
✓ Domino 1 (Script)
Script: domino1
Force: 0
Col: DominoBrick1 (Box Collider)
Rb: DominoBrick1 (Rigidbody)

Figure 4.1: Components for the domino GameObject

The *Transform*-component is used for placing the GameObjects at different locations in the scene. The *Box Collider* is used to check when two domino bricks collide with each other. It is an important component for the script to work. Information about the script can be read in section 4.3.

4.3 Scripts

Scripts were attached to the domino bricks to describe their behaviour. The code from the simulation in MATLAB was rewritten to a C#-script. This made it possible to decide the behaviour of the bricks in the scene. The code from MATLAB, however, only simulates one toppling domino brick. To simulate multiple bricks hitting each other, the code rewritten in C# needed to be duplicated with a few tweaks.

Two different scripts were created; one for the first brick in the sequence and one for all the bricks following. The reason for having different scripts for these bricks was that they needed to behave slightly differently. The first brick got an impulse to start the tipping. All the bricks following are affected by the force from the previous brick.

4.4 Graphics

The bricks in the animation needed textures to imply that they were in fact domino bricks and not just random bricks. The image in Fig. 4.2 was used as the texture for the bricks.

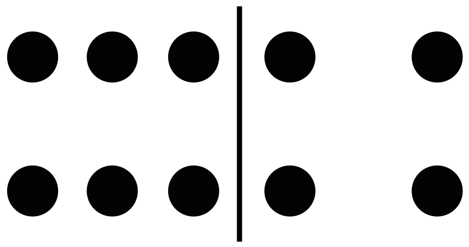


Figure 4.2: Texture used on the bricks

4.5 Scene Implementation

This section describes how the animation was implemented to make it look as correct as possible.

4.5.1 Pivot Point

The first step was to implement the animation for the first brick. Unity, however, does not give the option to choose pivot points of objects which caused the rotation to be incorrect by default. As can be seen in Fig. 4.3, the object has toppled with incorrect pivot point which resulted in half of the object going through the ground.

To solve this, an object was first created in the computer graphics software *Blender* and then imported into Unity. This allowed the pivot point to be set manually directly in Blender to the desired location. The outcome of this was a domino brick that rotates around the edge instead of the middle of the brick. The brick with correct pivot point but wrong texture can be seen in Fig. 4.4.

4.5.2 Implementing More bricks

A couple of problems were encountered when more bricks were added to the scene. To begin with, simplifications had to be made with the collisions between the bricks. When one brick hits the next brick, the force of which it collides with was set to an impulse identical to the initial impulse on the

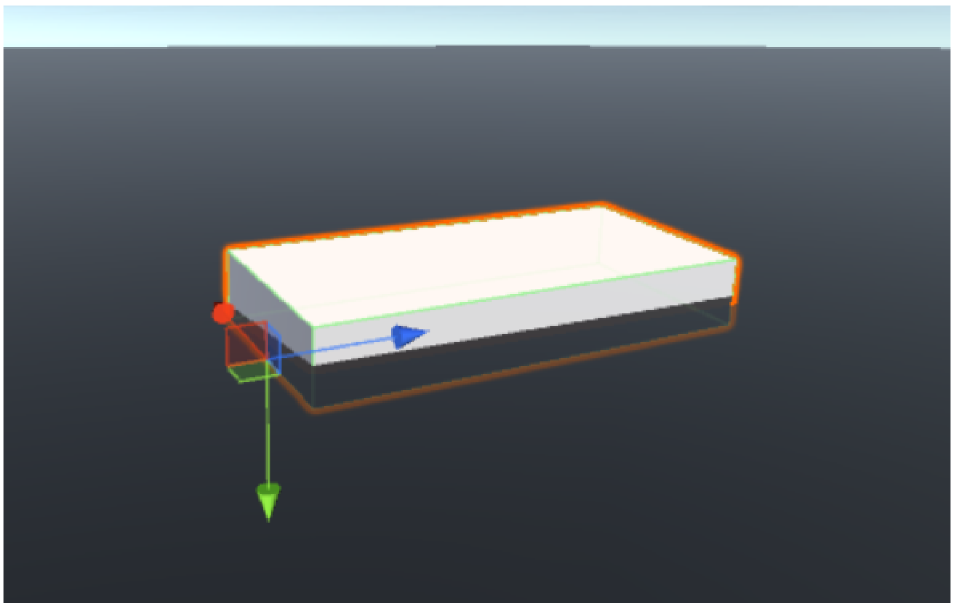


Figure 4.3: First brick rotating with incorrect pivot point

first brick. When the bricks collide, the brick leaning on the other brick has its angular velocity reset to zero, since we assume the collision to be elastic. The brick that had its angular velocity reset will then only be affected by gravity. Moreover, all dominoes rotated 90 degrees at first, merging all of the bricks, as can be seen in Fig. 4.5. The script had to be adjusted to make sure that the bricks stopped rotating at a specified angle to prevent merging, thus creating the illusion that the bricks stop when they hit the other bricks.

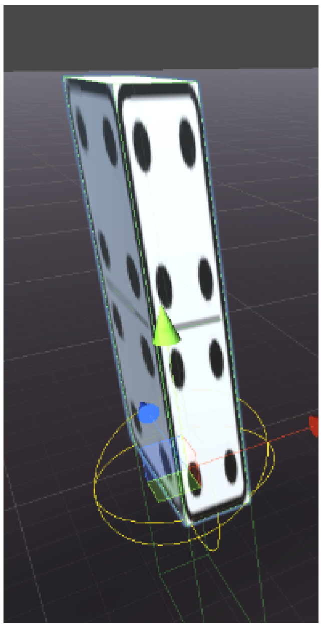


Figure 4.4: First brick with correct pivot point

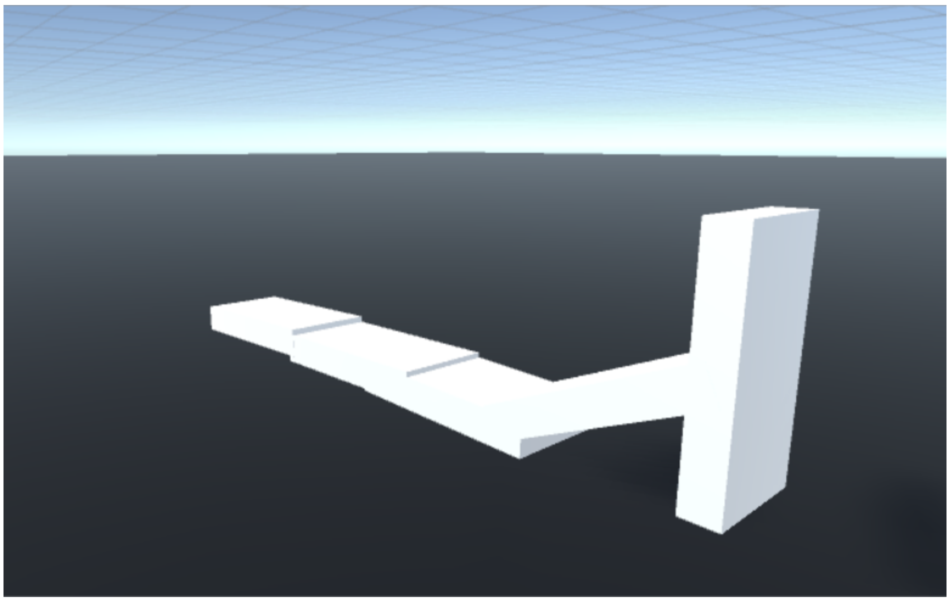


Figure 4.5: Program creating an animation of the domino effect with merging bricks

Chapter 5

Results and Discussion

This chapter demonstrates and discusses the results.

5.1 Results

The results from the simulation and animation are represented in this section.

5.1.1 Simulation

The simulation from MATLAB in Fig. 5.1 shows the behaviour of a domino brick as it topples. The impulse that is applied to the brick is of magnitude *F = 0.5 N*.

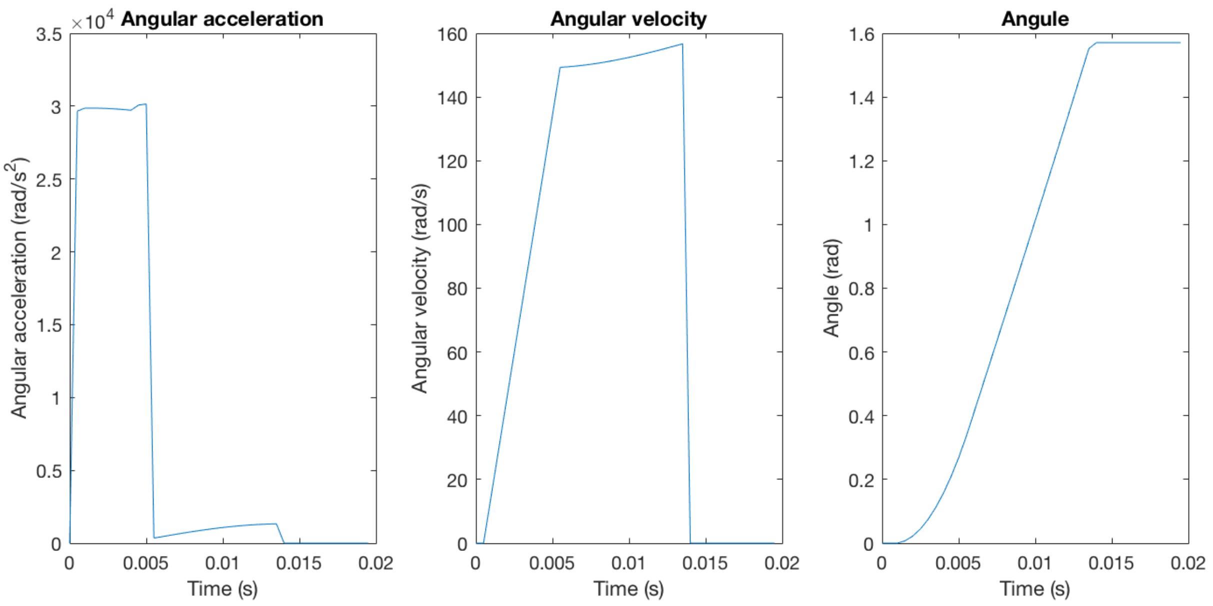


Figure 5.1: Simulation of one brick toppling

Fig. 5.2 demonstrates the behaviour of a domino brick that gets an impulse that is not enough to topple the brick. This only makes the domino wobble one time, and then return to its original position. The impulse that is applied to the brick is of magnitude *F = 0.005 N*.

Fig. 5.3 shows a MATLAB simulation of a brick that topples because of a collision with another brick. The impulse that hits the toppling brick was calculated with the help of (2.5). The bricks that collided

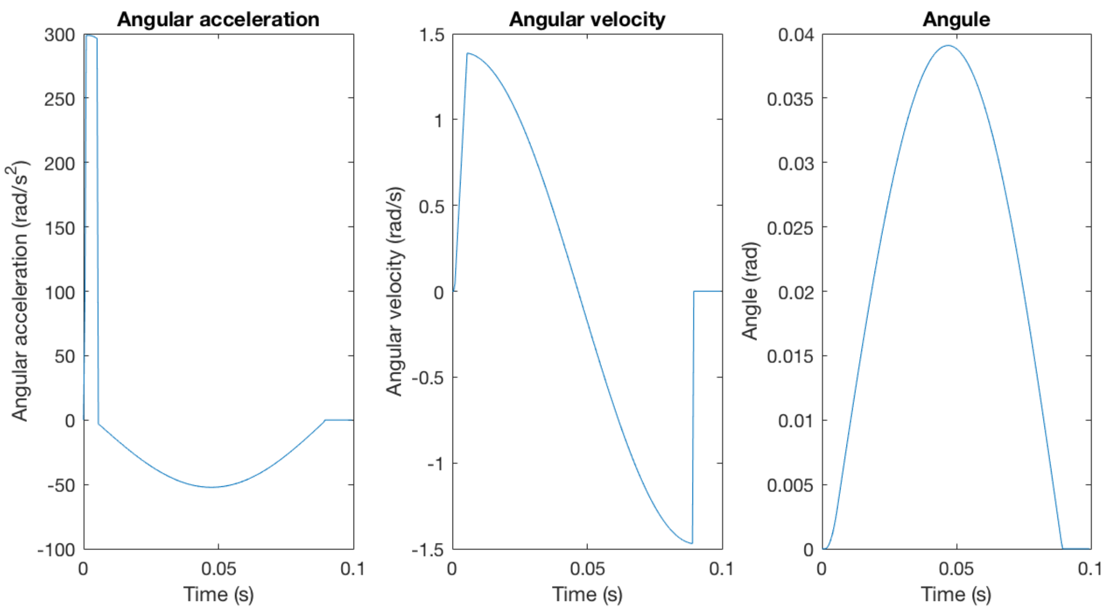


Figure 5.2: Simulation of a brick that does not topple due to the impulse being too small

were assumed to be standing 2 cm from each other which made it possible to calculate the angle of the first brick when the collision occured, which in hand provided the angular acceleration at that angle, which then made it possible to use (2.5). The impulse applied to the first brick had the magnitude 5 N which resulted in the impulse on the second brick having magnitude 5.0703.

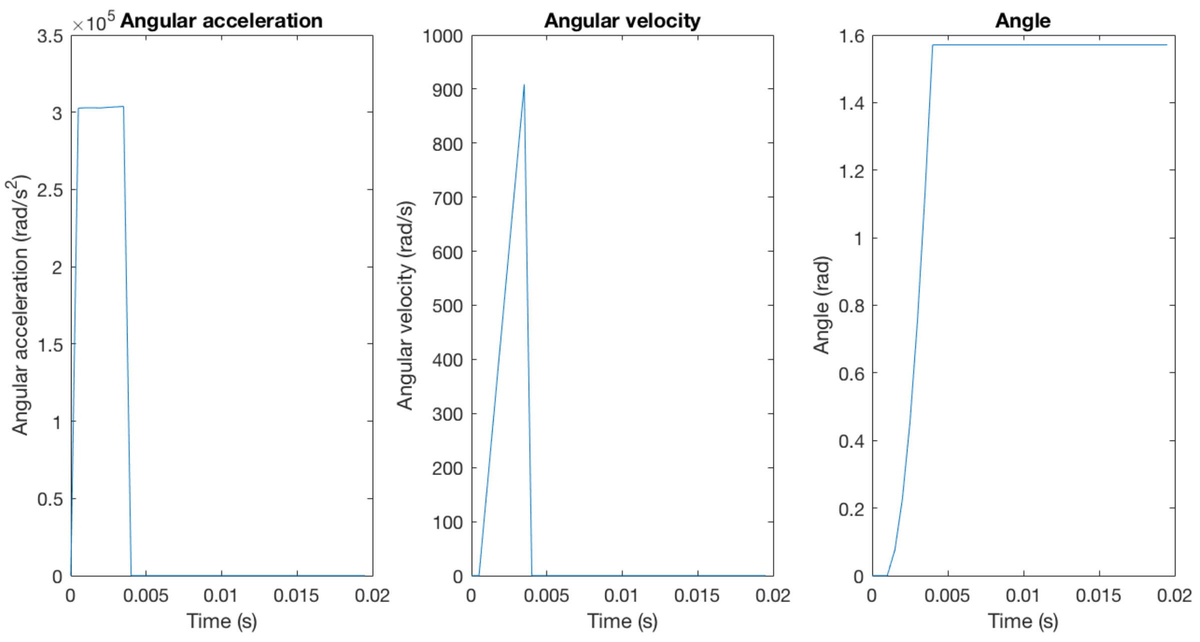
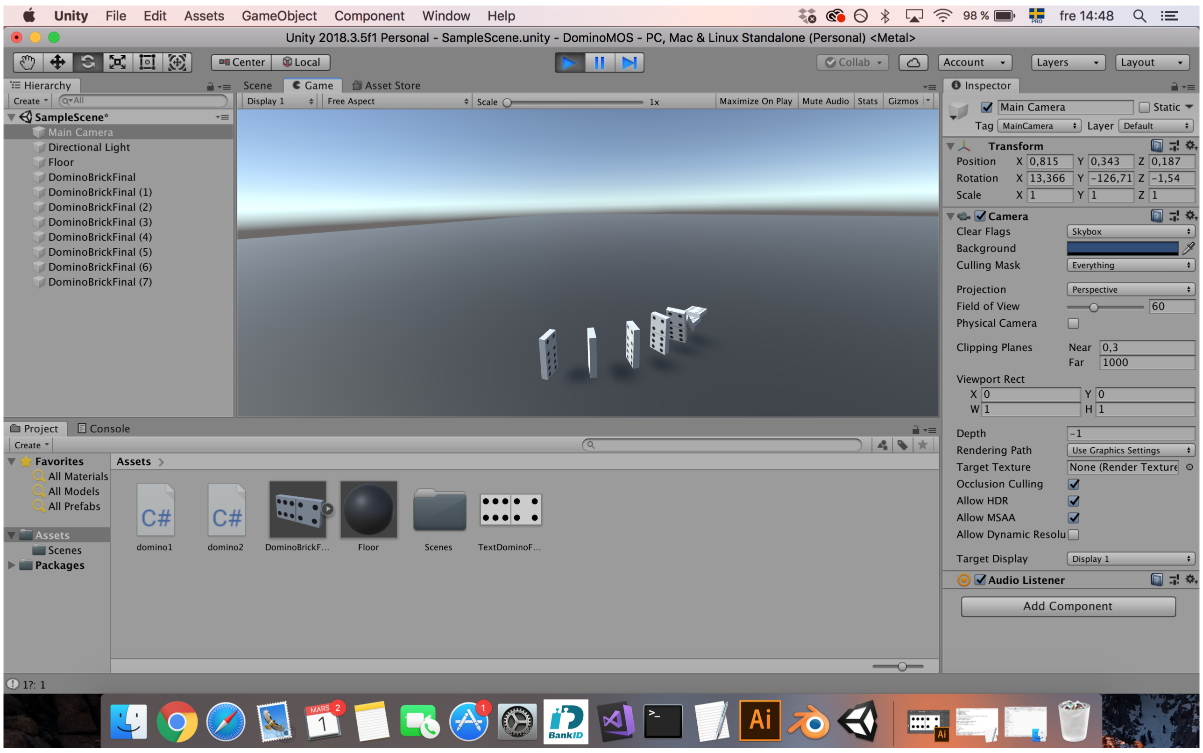


Figure 5.3: Simulation of a brick that topples because of a collision with another brick

5.1.2 Animation

Running the finished program with the domino objects, components and scripts, results in an animation of the domino effect. This can be seen in Fig. 5.4.



Main Camera — Static
Tag: MainCamera — Layer: Default

Transform
Position X 0,815 Y 0,343 Z 0,187
Rotation X 13,366 Y −126,71 Z −1,54
Scale X 1 Y 1 Z 1

Camera
Clear Flags: Skybox
Culling Mask: Everything
Projection: Perspective
Field of View: 60
Physical Camera: ☐
Clipping Planes: Near 0,3, Far 1000
Viewport Rect: X 0, Y 0, W 1, H 1
Depth: −1
Rendering Path: Use Graphics Settings
Target Texture: None (Render Texture
Occlusion Culling: ✓
Allow HDR: ✓
Allow MSAA: ✓
Allow Dynamic Resolu ☐
Target Display: Display 1

Audio Listener

Figure 5.4: Program running creating an animation of the domino effect

At the end of the animation all the bricks have fallen which results in a sequence of bricks in contact with each other, see Fig. 5.5.

5.2 Discussion

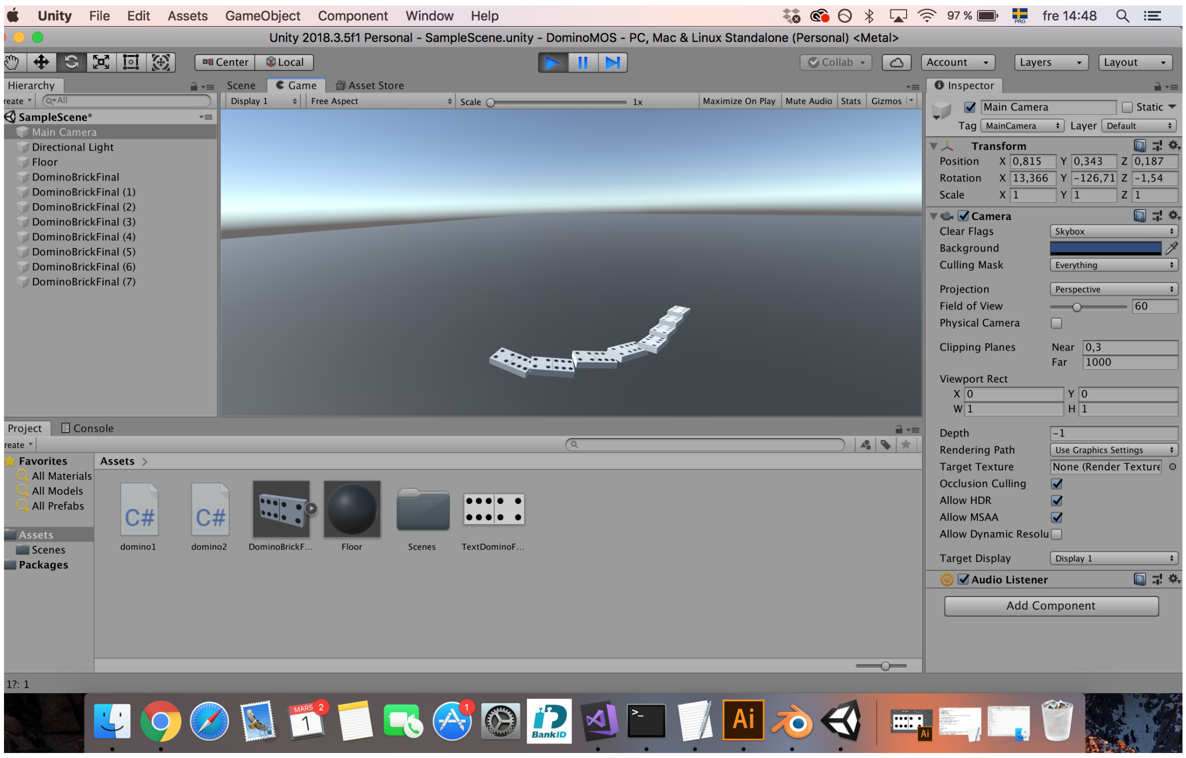
5.2.1 Simulation

Even though the simulation in MATLAB did not show a graphical representation, the mathematical model obtained by hand could be visualized in a detailed way by plots. The plots show how the brick acts and what to expect in further animations in a different software.

5.2.2 Animation

As mentioned multiple times throughout the report, Unity was used to animate the system. Unity was useful for many reasons. Not only does it provide plenty of documentation that make implementation easier, but it also has multiple built-in functions that speed up the process. Working with Unity is comfortable when implementing graphic visualizations. Since time was restricted, this felt like the best fit for our project at the given time.

There was not enough time to implement the calculations for several bricks in the animation. We had to use a shortcut by applying the same impulse to all bricks instead of calculating the actual force that hit each brick as was done in MATLAB, see Section 5.1.1.



Unity 2018.3.5f1 Personal - SampleScene.unity - DominoMOS - PC, Mac & Linux Standalone (Personal) <Metal>

SampleScene*
- Main Camera
- Directional Light
- Floor
- DominoBrickFinal
- DominoBrickFinal (1)
- DominoBrickFinal (2)
- DominoBrickFinal (3)
- DominoBrickFinal (4)
- DominoBrickFinal (5)
- DominoBrickFinal (6)
- DominoBrickFinal (7)

Inspector

Main Camera ✓ Static ☐

Tag MainCamera Layer Default

Transform

	X	Y	Z
Position	0,815	0,343	0,187
Rotation	13,366	-126,71	-1,54
Scale	1	1	1

Camera ✓

Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
Field of View	60
Physical Camera	☐
Clipping Planes	Near 0,3 Far 1000
Viewport Rect	X 0 Y 0 W 1 H 1
Depth	-1
Rendering Path	Use Graphics Settings
Target Texture	None (Render Texture
Occlusion Culling	✓
Allow HDR	✓
Allow MSAA	✓
Allow Dynamic Resolu	☐
Target Display	Display 1

Audio Listener ✓

Add Component

Figure 5.5: End of animation

5.2.3 Simplifications

The system had to be simplified to be implemented. It was, however, important that the simplifications did not compromise the quality of the animation. That being said, the domino bricks were simplified to fixed positions around their pivot points. This prevented the bricks from sliding on the ground, which would of course happen in the real world due to inertia.

The collisions are also simplified. A collision between two domino bricks would not be completely elastic, but due to such small differences in outcome, they are assumed as elastic anyway.

Another simplification made is when the first brick is hit with a force that is too small to topple. If the brick does not pass its tipping point, then it returns to its initial position without any oscillation. A real domino brick would oscillate before returning to its initial position. Since the dimensions of the dominoes are so small, the oscillations would be minimal, making them insignificant for this simulation.

5.2.4 Improvements

Further improvements can always be made when working with simulations of this sort. Adding a third dimension for the falling and landing of the brick is an improvement that, especially at close distance, makes the animation more realistic. The brick would tip in a more sensible fashion if it were to be unlocked from its pivot point. This would, together with friction, provide a better fall.

The animation can be improved by having a position-based domino effect, meaning that the exact angle of each brick is dependant on all the other bricks position and angle. This could be done by using trigonometry.

Chapter 6

Conclusion

Unity is a great tool for animation and integration of physics through scripting. Since it is a game engine – and depending on ambitions, skill and time available – more or less of the built in functions can be used to complement these scripts.

Even though the simplifications, discussed in section 2.3, were made, the simulation and animation demonstrate an acceptable representation of the domino effect. Simulations can always be improved with more time, but depending on the aim, a decision of what is good enough for the particular case has to be made.

Bibliography

[1] David Halliday, *Principles of Physics*, John Wiley & Sons 2014

[2] Lennart Ljung & Torkel Glad, *Modellbygge och simulering*, Studentlitteratur 2011

[3] Mathworks, *Matlab*, Retrieved: 2019-03-07
https://se.mathworks.com/products/matlab.html

[4] Unity Technologies, *Learn*, Retrieved: 2019-03-07
https://unity3d.com/learn

[5] Unity Technologies, *Programming in Unity*, Retrieved: 2019-03-07
https://unity3d.com/programming-in-unity