Seismic Cities
Technical Report

TU Delft

SEISMIC CITIES

TU Delft Delft University of Technology

Challenge the future

SEISMIC CITIES

TECHNICAL REPORT

by

Joris Bouwens
Andrea Donati
Fabian van Gent
Sander Kats
Remi Van Der Laan

1359827

IN4302TU Building Serious Games
Group 5 Earthquakes

Delft University of Technology & Science Centre Delft,
submitted by January, 2017

Supervisor:	Dr. ir. R. Bidarra	TU Delft
Commissioner:	MSc. Jules Dudok	Science Centre Delft

TU Delft Delft University of Technology

CONTENTS

1
INTRODUCTION

In this technical report will be addressed the main technical challenges and choices we made during the development of the Seismic Cities game, starting from the game engine we chose to the details of the implementation.
We will also illustrate different attempts we made to simulate the earthquake, even if finally we decided to not include them in the game.

2

Technical Choices

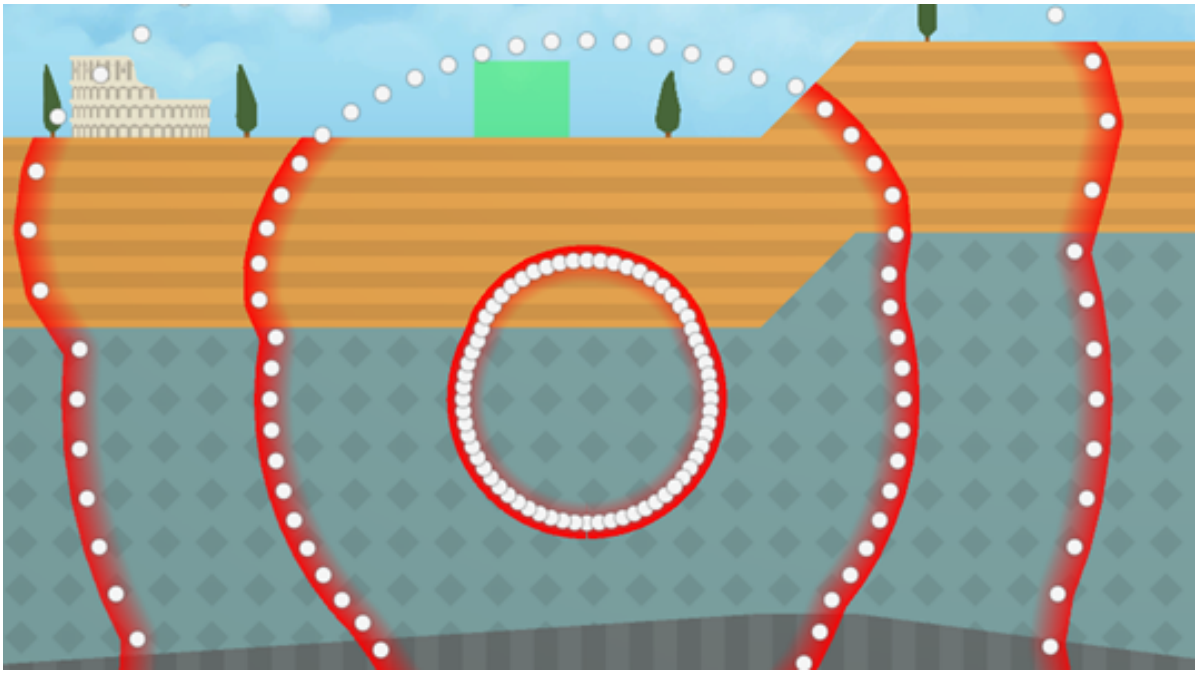
2.1. Game Engine

One of the requirements of the commissioner was that the game should be played in a browser. Therefore we considered using the Unity3D or the Libgdx game engines, but since there was more experience in the team with the former, we chose to work in Unity3D. Its editor also makes it easy to quickly creates mockups and prototypes, which is useful in a project of such a relatively short duration.

Due to Unity's implementation of the Component Pattern for every game object, we did not specifically had the need for inheritance in classes. Behaviours that needed to be reused could be split up into separate components.
For all components that were not controlling their own game object, but were managing (a part of) the game state, we used one object, which we call the Game Manager.

2.2. Earthquake Simulation and Visualization

After deciding that the game would be seen from a 2D side view of soil layers, we had to come up with a way to visualize a wave moving through the earth. After a while, we settled with a simple solution: Send out a large number of moving objects in a circle around the hypocenter of the earthquake and draw lines between them, which forms a wave. This way we could detect on which soil type which part of the wave was located and allowed us to simulate that waves move at different speeds through different soil layers. This however did not have a large effect on the gameplay in the end.
Using this method, there was the issue that the wave is visible outside of the terrain. To solve that, we wrote two custom shaders that makes uses of the stencil buffer. The terrain serves as a mask over the wave, so that it becomes transparent outside of the terrain.
The final result can be seen here, with a visible white point where every moving object is located:



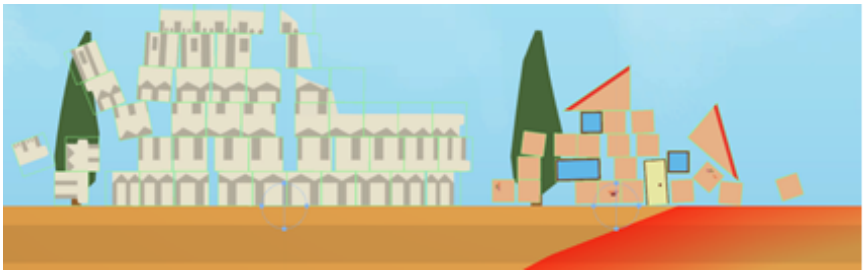
We have some ideas to make it more efficient, since it does cause a performance hit, but decided to leave it like this because it works good enough and would require spending precious time that could be spent on other features.
In addition to the visualization, a similar object is sent out to every building zone, so we can detect when the earthquake reaches any building and with which speed.

2.3. GAME MODE MANAGEMENT

To switch between game modes, such as the building, simulation and finish modes, the state pattern was used. Between transitions of one state to another, specified game objects were enabled or disabled, and also allowed us to call certain functions. This is used for example to show and hide the lists of buildings and upgrades, simulate the earthquake and to trigger certain dialogs.

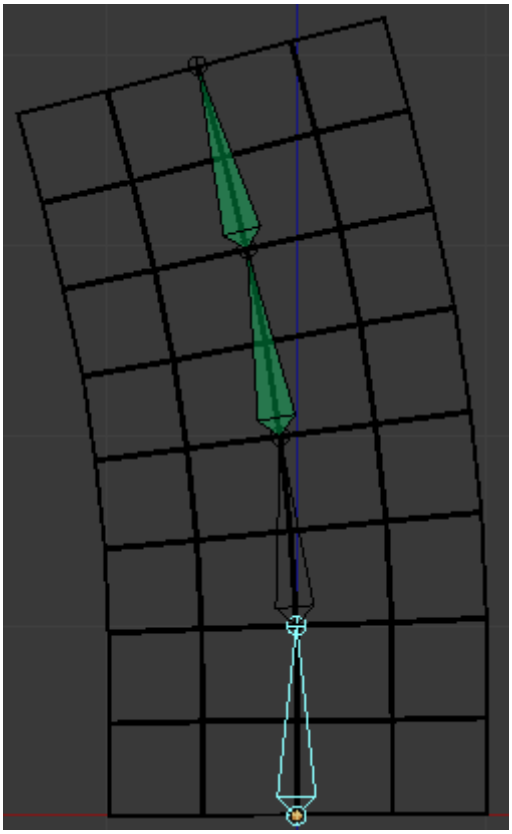
2.4. BUILDING DESTRUCTION

After many failed attempts at creating physically accurate buildings that should fall apart due to vibration, which will be further explained in Recommendations and Warnings, we eventually implemented a more controllable method.
We wanted to define for every combination of building type, soil type and upgrade whether a building would collapse or sink, but then the distance and magnitude of the earthquake would also have to be taken into account, which would make level design really complex.
Since we only have a small amount of levels, it seemed more straightforward that we define per building zone in each level which buildings are placed correctly. Depending on the type of soil, a building will either collapse or sink into the ground.
The collapsing is done by splitting the building sprite into several small pieces, each with a rigid body, and adding a small force to it. Sinking is done by simply moving and rotating the sprite into the soil.



2.5. GRAPHICS

Most of the graphics were created in Blender, a free and open source 3D creation suite.
The terrain was exported as a mesh, so we could easily apply colliders to it. Some simple patterns were created with dots, diamonds and lines and applied as texture so that the terrain would look more interesting.
Buildings and scenery objects were rendered as an image and imported as a sprite into Unity. The trees were animated by rigging a subdivided plane with three bones.
The menu screen globe was created by tracing a real world map and rendering it as a texture, which was also used to create a normal map. The 3D fault lines inside the globe are a separate object that was created by tracing a world map that included fault lines. In order to see the fault lines inside the globe, this object is rendered on a separate camera and it is overlaid on the main camera. To only see the fault lines on the side of the globe that is facing the camera, a shader was used that fades vertex colors to transparent depending on the distance to the vertex to the camera.
The background images of clouds were not created by us, but we did implement a parallax scrolling effect, which gives the effect of 3D when moving the camera.



2.6. LEVEL CREATION AND MANAGEMENT

In the beginning, we wanted to have every level in one single scene, so that if we had to change one object we would only have to do it once, and not in every single level.
Eventually we decided against this idea, which would have required spending much time on resetting objects and keeping track of all objects that might behave differently in separate levels. It could have also had an impact on the performance, since all levels would have to be loaded in memory at once.
We did find a middle road by using prefabs. Many objects that are shared between scenes are saved as prefabs and are imported in all levels.
This part of the game creation is one that could certainly use some improvement, therefore this will further be discussed in the Further development section.

2.7. DIALOG SYSTEM AND TUTORIAL

The two main characters of the game story are Poseidon and Athena and the dialogue system is used in order to let them talk with the player. The dialogues are displayed in foreground in the scene and the player can go through them by clicking with the mouse, while the speaking character is displayed in one of the two sides of the screen.
The system provides in the inspector three lists of dialogues, one to be displayed at the beginning of the level and two at the end, when the player fails or passes it. The lists are easily editable in the inspector.
Every dialogue item is a sentence of a character and it has two events attached, one is triggered when the dialogue is shown (the Enter Event) and the other when the player clicks and goes to the next one (the Leave Event). These events are used to perform particular actions like for example starting the tutorial in the first level after the initial dialogue is over, or for displaying the Retry and Next Level buttons at the end of the levels after the end dialogues.

A tutorial is shown in the first level in order to explain the main game controls. The player is guided through it by clicking on buttons and by camera movements. The tutorial is defined by a list of instructions in the inspector, where the game developer can easily modify the text and the position of the balloons containing the instructions.

3

Further Development

3.1. Level Management

If this game would be expanded upon at a later date, the first thing that should be looked into is the level management. In the last weeks, many small changes had to be made in all levels. Most had to be manually adjusted in every scene, which has cost us more time than necessary and also makes it easy to make mistakes or skip some levels by accident.
One solution would be to create a custom level manager, with a clear distinction between shared and individual components. This would also allow us to easily create an in-game level editor where the user can assemble their own terrain and solutions to pass the level.

To create such a level editor, all variables in a level need to be identified. Currently, the differences between levels are:

- The list of buildings and upgrades
- The building zones and which placements are allowed on them
- The terrain and scenery objects
- The position of the hypocenter
- The dialogs and tutorials

All other aspects of the game are shared between all levels and don't have to change on a level by level basis.

Another issue that we came across was in the code collaboration. If merge conflicts occurred between two of our branches in scene files, it was difficult to figure out which parts contained which changes and caused some changes to be lost in the process. Since this was a problem with only two programmers, it would only be more difficult if more people were to work together. At the start we thought this might be an issue, so we enabled a setting that all metadata files should be saved in a serialized text format, but it did eventually not make everything clear enough.
This could also be solved with the previous solution, because then the individual scenes would not have to be modified, but there would be a custom understandable file that defined each level.

4

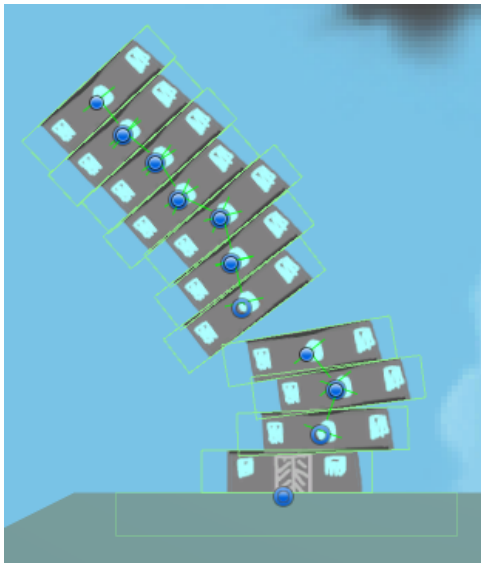
Recommendations and Warnings

4.1. Vibration Simulation

The first attempt on making an accurate physics simulation was based on invisible platforms.
We could not vibrate the whole terrain when an earthquake happened, because we wanted to let the user see the change in velocity of a wave through different soil types. Therefore came up with the invisible platforms on each building zone, that start shaking when a wave hits them.
The player is forced to put the buildings on these platforms instead of placing them in an arbitrary point on the surface. The platforms started shaking when hit by the first wave of the earthquake and they were shaken by applying side forces rhythmically. The forces applied to the platform depended on the type of soil where the platform was, in fact in softer soils correspond more intense shakings. The shaking intensity were also depending on the intensity of the earthquake (Richter scale) and on the distance from the hypocenter.
The placed building was connected to the platform by a Fixed Joint, a Unity type of joint that keeps two physics objects together at a fixed distance, but it can break if the amount of applied force is greater than the maximum that the joint can handle. In our case the maximum amount was Infinite because we wanted the building to stay in his place.

4.2. Physically Accurate Building Materials and Structures

A further improvement toward the physics simulation of the buildings was to introduce buildings made of blocks. The blocks then were connected together by Spring Joints, that is a Unity type of joint that connect two objects and behaves like a spring, so the objects can move a bit between each other but not too much, and if the distance increase they are put back to their position by the spring force. This was really useful for two main reasons: in this way the building could oscillate following the platform movement, generating a sort of internal waves going from the base to the top, and the second reason was that the buildings could collapse and fall apart in pieces when the maximum forces of the Spring Joints were reached.



One issue with this is that heavier buildings would weaken the effect of the vibrating platform, and they would not shake at all. We tried to remedy this by not using the physics engine for the platform, but to move it sideways in a sine wave with a predefined amplitude.
The behaviour of the physics simulation was actually very nice, but we realized that it was really unpredictable since it was based on some random forces and on the Unity physics engine. With this kind of unpredictability was very difficult to design balanced level where the player had to place the buildings carefully and their so-

lutions. For this reasons in the final game we decided to remove the physics simulation but we kept the idea of the building platforms to force the player to place the buildings only in some designated areas.