Procedural Map Generator

Ilana Ben Avraham

Supervisor:

Yaron Honen

Contents:

Introduction

Development tools

Development process

Application user guide

Link to project's demo on YouTube:

Introduction:

Procedural generation is a method of creating data algorithmically instead of manually.

Procedural map generation is a derivative of this method for creating unending worlds.

Imagine yourself as a character in a video game exploring and traveling through a vast virtual world.. when suddenly, you reach the end of this world, a cliff, if you take one more step you will fall to infinity never reaching the ground.

Procedural Map generation comes to solve this issue and allow you to explore the world without it ever ending around you.

My project allows the player to explore a building which is built in a procedural manner around the player online as it moves. The building is built from basic building blocks that were built in a specialized program. These basic blocks are connected according to a set of rules which was written beforehand.

Development tools:

Unity is a cross-platform game engine developed by Unity Technologies. It offers a scripting API in C#, option for adding plugins and .fbx files.
I have created a 3D project and used version 2019.2.17f1 of unity.

Microsoft Visual studio is an integrated development environment (IDE) from Microsoft.
It is used to develop computer programs and allows attached debugging to unity. I have used version 2017.

Blender is a 3D computer graphics software toolset used for creating visual effects, art, 3D models, motion graphics, interactive 3D applications, and computer games. I created 3D models of building blocks for creating the building, those included cubes, arches, stairs, and bridges etc. as will be explained later.

Development process:

The development process consisted of a number of stages:

Formulating the idea of procedural map generation:

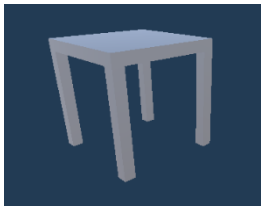
The world is divided by a grid of cubes, every cube is the size of 1X1X1. At run time two such grids are saved and updated, one is for the computational result of the algorithm as for what model (basic block) fits in what rotation at any location in the world ("calculated grid") and the second is for a pointer of the graphical representation of the model in every world position ("visible grid").
At the game settings the player can choose the size of grid being visible around him, let's assume the default value is chosen and its 7, as the game starts the player is placed at location (0, 0, 0) of the world while the graphical representation seen to him is $7/2$ cubes in every direction (up, down, left, right, front and back).
As the player moves, let's say in direction front, a new layer of the world grid is calculated and presented to the player while the farthest layer of the world in the players back is destroyed.
The calculation for which models are supposed to appear at the destroyed layer are saved in the "calculated grid" so when the player returns to the same place he will see the same world. The actual graphical representation is saved in the "visible grid" and the pointers there are actually destroyed but are reinstalled when the player retrace its steps according to the information saved in the "calculated grid".

There are two purposes for saving two grids:
One is to make every calculation once (reducing calculations made online) so the world calculated will be permanent.
The second is to reduce graphical calculations, the number of models presented at any given time is constant.

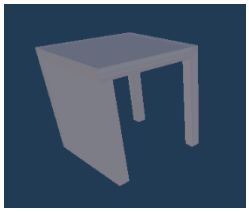
Modelling in blender:

I have built 11models using blender to serve as basic building blocks of buildings, these include the following:

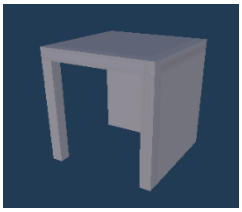
Cube



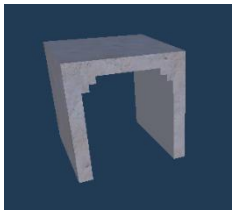
Table



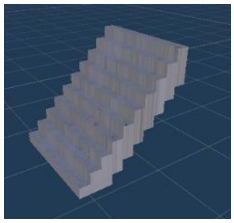
Plus



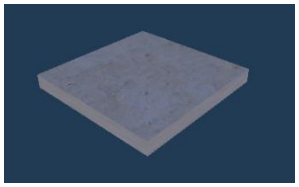
Turn



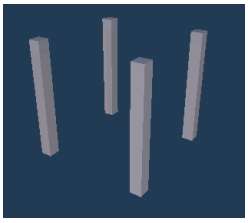
Arch



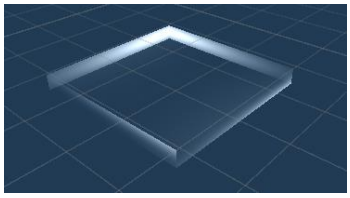
Stairs



Floor/Bridge



Columns



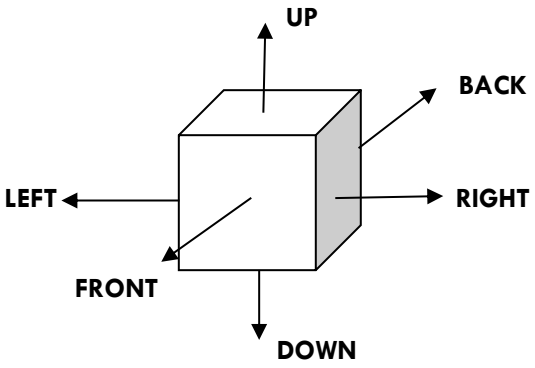
Glass bridge

*Air = empty cube space

Every model has an option of traversing rotation when placed by the procedural algorithm, meaning front connection (0 deg rotation) of a model can serve as right (90 deg rotation), back (180 deg rotation) and left (270 deg rotation).

Setting the rules:

Every place in the world is represented by a basic cube of the size 1X1X1, while this cube has 6 facets:



I created a set of rules which indicate for every basic model its connection name to every facet, for example:

```
model:_Stairs
top:stairsTop
bottom:cubeBottom
front:stairsFront
left:stairsSide
back:stairsPath
right:stairsSide
```

This set of rules also contain possible connections:

```
stairsPath-path
```

meaning model Stairs at its back facet has connection stairsPath, connection stairsPath-path exists so every model which has path connection at its either front, back, right or left facet can connect to the back of the stairs model.

Such set of rules needs to be complete, consistent and sound so there will be a logical connection for every cube on every its facet.

Of course, there are situations when a certain cube's facet has a number of possible connections, in such case the algorithm randomly picks one of them.

Writing the algorithm:

Pseudo code of the algorithm (the shortened version):

Create a new layer in the direction of the player's movement
Loop:
Observe – recalculate every grid position entropy and collapse* the one with the lowest one.
Propagate – update all new layer's grid positions according to the last collapse
Exit loop if all the new layer's grid positions were collapsed

***collapse = choose model randomly out of all optional, accordingly to the set of rules, models for the current grid position.**

Adding cool features to increase the user's experience:

Trail – as the user explores the world a bright trail is left after him so if he wants to return back to all the places he visited he will know which way to go.

Paint ball shooting – the player can shoot paint balls on the building's walls up to 50 meters from him such that whenever he visits those spots again the color stains will still be there. This way he can leave its own mark in the virtual world around him. There are 3 optional paint balls – green, purple and black (every fire the color is chosen randomly).

Application user guide:

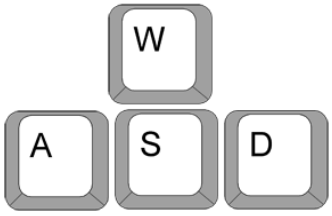
Main menu, to start exploring the world when its default value is 7X7X7 press “Play”. To exit the application press “Exit”.

To Adjust the world’s size press the “Adjust World Size” button and you will pass to the “Adjust World Size” menu:

To apply the axis size press “Apply” and to return to the main menu press “Back”.

Once you are positioned in the world you have two available cameras:
*Press TAB to switch between them.

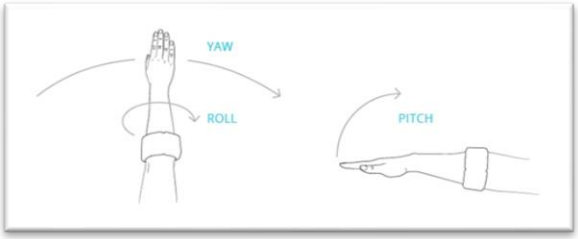
FPS Camera:



Movement is controlled by:
W = forward, S = backward, D = right, A = left.
Curser = controls the player’s head movement.

Scene Camera:

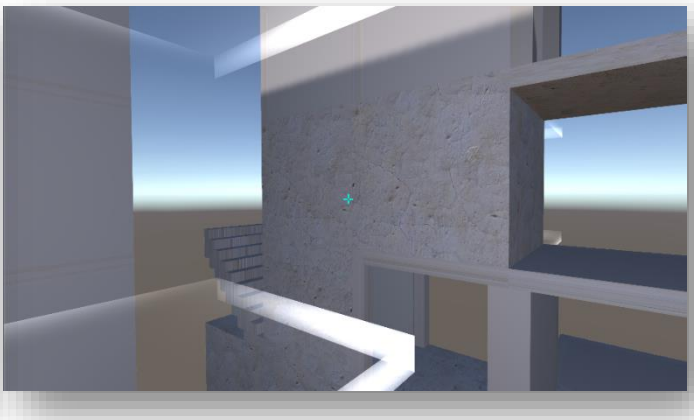
Movement is controlled by:
W = forward, S = backward, D = right, A = left.
Mouse:
Right button = controls cameras yaw and pitch
Mouse wheel = zoom in/out.



Shooting paint balls:

To activate and deactivate the crosshair for better aiming press right/left Alt.

To shoot up to 50 meters press ESCAPE.



While exploring press Esc to get to this menu:

To get to the main menu press “Start Over”.
To resume exploring exactly from the same state press “Resume”.
To exit the application press “Exit”.

Link to project’s demo on YouTube:

https://youtu.be/e7jrCKjlNk8