1st Oral

From

Team

Romain Biessy Renaud Gaubert Aenora Tye Erwan Vasseure

Contents

Chapter 1

Introduction

An ambitious 3d game such as ours requires a lot of work, to give you an example we spent not less than 100 hours each on the project since october also our project actually counts 120 000 characters and 15 000 lines of codes with not less than a 100 files.

Of course this would be meaningless if we wrote bad code however while not having the pretention to say that we write good code we have the pretention to say that we do note write bad code.

Throughout this report we will try hilight the fact that our work has payed and we already have good base to proceed without worries the developpement of our game.

Chapter 2

Gameplay

2.1 RTS, FPS & minecraft-like

2.2 Scenario

As time went on, we thought of a better scenario than the one elaborated for the book of specifications. And we came up with two scenarios of our own :

In the second part of the 24th century, human race is at war, you are a soldier, sent to unknown planet which seems to be only composed with flying islands and your goal is to take the island right under the ennemy's nose !

Un 2nd scénario ici

Chapter 3

State management

3.1 What have been done

The state management system has been implemented by Romain. The idea is simple and powerful. There is an abstract class State which can basically be started up, updated and shutted down. Then two classes inehrit State : MenuState and GameState with their own attributes. The most important class is the StateManager, it has a stack of State which can be easily poped and pushed from other classes using reflexion - we just have to specify the type of the state to the StateManager in order to push a state. Each frames, the StateManager updates only the first State of the stack so that the other States aren't destroyed but are liked paused since not updated.

This sytem has the advantage to structure the code. Besides, it makes the change from one State to another very easy.

3.2 What must be done

There isn't much work anymore fot the state management. All we need now is to implement some other States such as an OptionState. Then it could be launched from any other existing State.

Chapter 4

Character

4.1 Mesh

At the beginning we were using the mogre mascot called Sinbad. It was useful since it has predefined animations; the drawback is that he doesn't have the right size, and it isn't made for being integrated in a cubic world. After that Aenora and Erwan made a few different characters in blender, a 3d software. We encountered difficulties while trying to implement the new mesh file and its animations into the mogre engine. Some of the character textures were made with the UV mapping technique.

4.2 Move

We can move our character with the directional arrows or the WASD keys. Animations start by pressing one of these keys. There are four animations; two for the walking loop and two for the idle loop. Erwan integrated the keys manager, inspiring himself from the MOIS manager provided by mogre.

An issue Erwan encountered is the rotation of the player because we have a 3D world. If we needed some advanced rotations we would had to use quaternions, a powerful but non-intuitive mathematical tool similar to matrice in 4 dimensions. Fortunately an easier way is to decompose the rotation within the 3 axes (x, y, z). This is way simpler since we only need to rotations : yawing the player (around the y-axis) switch the horizontal movement of the mouse and pitching the camera (around the x-axis) switch the vertical movement of the mouse. Erwan implemented the rotation as well.

4.3 Animation

Romain intergrated a fade for the transitions of animation. Basicaly, animations have a weight between 0 and 1, 0 is no animation and 1 is the full animation, 0.5 would be that animation, but with a lower amplitude. The fade is made by downcreasing the old animation's weight to 0, while increasing the new one to 1. This method garantees a smooth transition between all the animations even if they are stopped before their end in game.

4.4 Collision

The hit-box of the character is represented by the 8 points of a parallelepiped rectangle. To test collision we look at the type of block around the character. For instance if we want to test the feet collisions, we get

the type of the block under the character, using the 4 lowest points. For the next presentation, Romain will have to finish all the collisions test.

4.5 Camera

We can switch among two cameras in our game the first person one and the debug one. The first person camera is following the player, it just has to be pitch depending on the mouse. The debug camera isn't supposed to be seen by a player, thought it is very useful for us since it's a free camera detached from the player. Thus, we can see the whole world and the player with its animations. They have been both implemented by Romain.

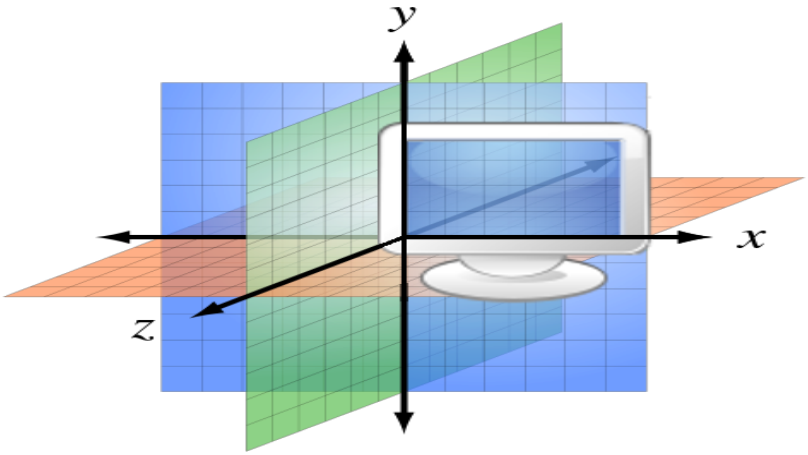
4.6 What must be done

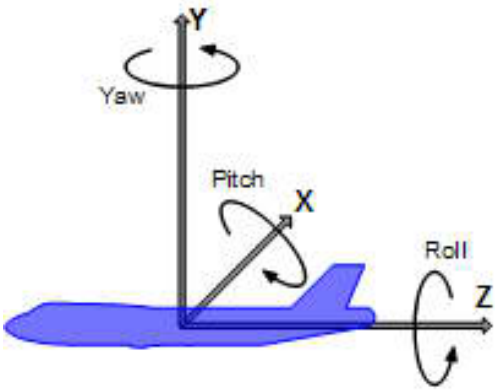
Chapter 5

Graphic engine and terrain

5.1 Ogre3D

A 3D world is at first sight more complex than you could imagine.The player moves via an FPS (Fisrt Person Shooter) type camera. While its position in space is given by a Vector3 (3 floats). The orientation of the camera uses rotations about the X and Y axes: this is called respectively the yaw and pitch (the roll, rotation around the Z axis is not useful in our cases).





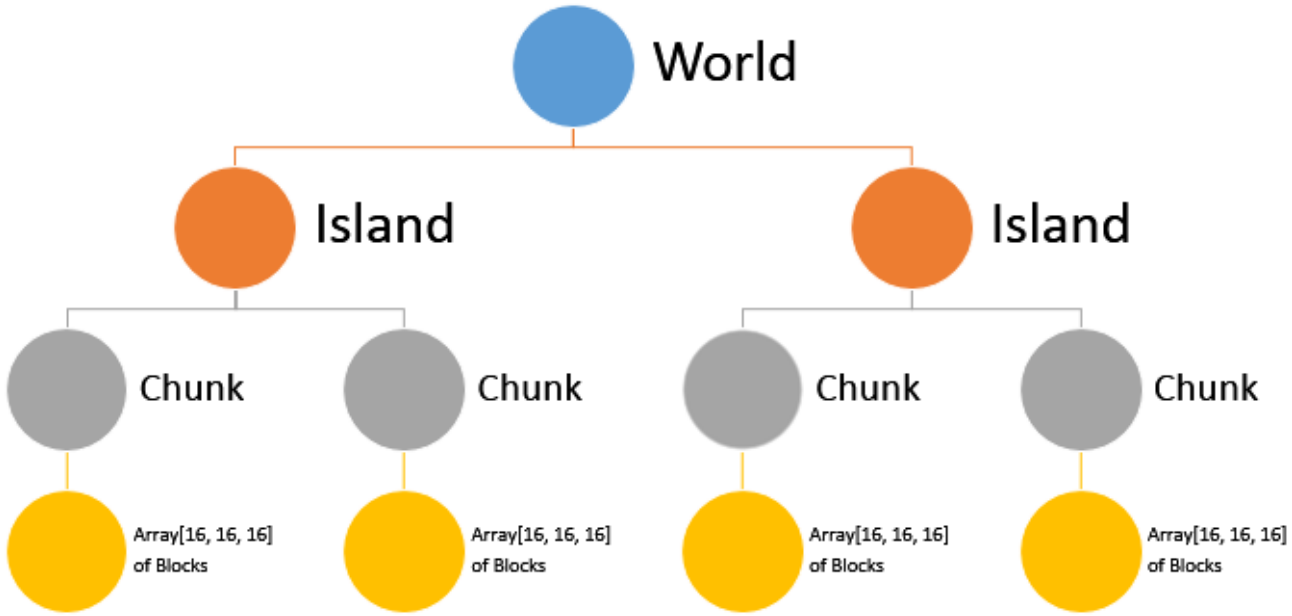
A trick to find if the rotation you want to apply is a positive rotation or a negative rotation is the following :

Using your right hand as a guide: point your thumb in the direction of an axis, curl your remaining fingers. The direction of the curl matches the positive rotation around that axis

5.2 Terrain Generation

When creating the terrain architecture we had the goal to separate the terrain in multiple arrays which we called chunks. Thus, when we will be able to save the terrain, we won't save it in a single file. Also this permits us to later, dynamically load the terrain chunks by chunks and not as a single entity.

The terrain Hierarchy is as follows : World contains a few islands which themselves contains a dictionary of chunks which contains a 16 * 16 * 16 array of blocks. It follows this chart :



test

The terrain itself is generated when we create the island, using a perlin 3d algorith we go through all the blocks in the islands (3 simple for) and for each block, we compute it's perlin noise value and check if it is superior to 0 to display it.

5.3 Terrain Display

5.4 Solved Issues and Unsolved

5.5 What must be done

Chapter 6

Conclusion