# Final Project-Anibot

Donika Todorova Dimitrova 1805163

28 June 2021

## Idea Anibot

Initially the platform was created to reproduce the sounds that some animals make. Later it was divided into two parts: farm and wood. Moving through the controls Anibot, a robot created by me, you can see the animals on which the name is written.

You can play Anibot here: https://sapienzainteractivegraphicscourse.github.io/final-project-dimi-team/


## User Guide

Y: Turn On/Off music
W: Move forward
S: Move Backward
A: Move Left
D: Move Right
SPACE: Move Up

Before starting the path in the platform there is a small user guide with the commands used to move Anibot. There is background music in the project that can be stopped or not. Also by clicking once on the animals you can hear their sound.

In addition to the keyboard, the user can also move the camera to the right / left and up / down with the mouse. Also click on the animals with the mouse.

## Libraries

- **babylon.js**
  A real time 3D engine using JavaScript library for displaying 3d graphics in a web browser via HTML5. It's a powerful open rendering engine that gives the possibilities to build meshes, models, lights, cameras and so on.
- **babylon.objFileLoader.js**
  Library used to load Obj files
- **babylonjs.loaders.js**
  Library used to import external objects and models.
- **babylonjs.proceduralTextures.js**
  Babylon.js offers a simple and straightforward way to use this type of texture. Procedural texturing is a way to programmatically create a texture. One main advantage of procedural textures is that they are written using a fragment shader (using GLSL in the case of Babylon.js). That means that the code generating the texture is executed by the GPU and not the CPU.
- **babylon.viewer.js**
  The Babylon.js viewer is the simplest way to display 3D content on a web page. The supported formats are: glTF, .babylon, .obj, and .stl format.
  There are plugins for 4 physics engines:
- **cannon.js**
  Library used for physic contents.
- **ammo.js**
- **Oimo.js**
- **babylon.grassProceduralTexture.js**
  Grass procedural texture
- **babylon.gui.js**
  The Babylon.js GUI library is an extension to generate an interactive user interface. It is built on the basis of DynamicTexture.
- **babylon.glTF1FileLoader.js**
- **babylon.glTFFileLoader.js**
  This loaders supports both glTF 1.0 and 2.0 and will use the correct loader based on the glTF version string.
- **babylon.objSerializer.js**
- **babylonjs.serializers.js**

This serializing a mesh

- **babylonjs.materials.min.js**
  The Babylon.js materials library is a collection of advanced materials to be used in a Babylon.js scene.
- **babylon.inspector.bundle.js**
  Babylon.js inspector is a visual debugging tool created to help pinpoint issues you may have with a scene.

## Scene

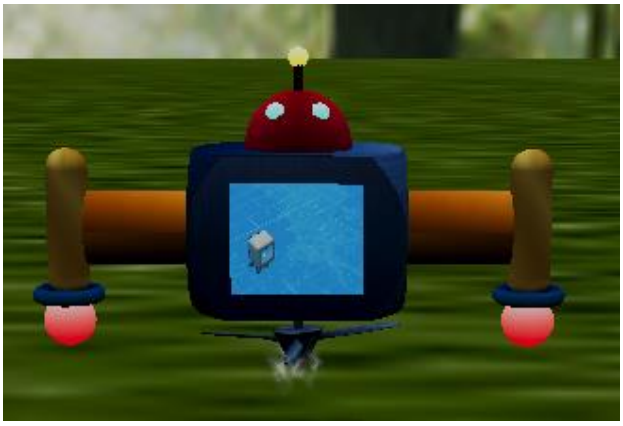
A scene is a container where meshes, lights and cameras are placed and, thanks to the engine, is made to work. In Platform there are two  scenes:

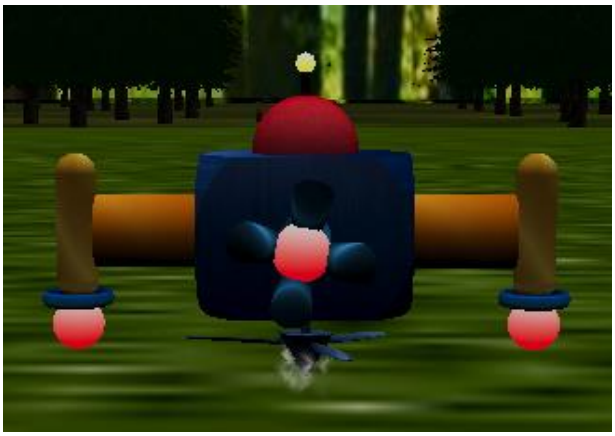
-Guide scene: Basic scene with a camera and a simple environment  that contains the imported  a short  guide, displays the title of the project and start button. Clicking on the button the application switches to Game scene.

- Game scene: The most important scene that contains all the game.

## Models

## Hierarchical model - Anibot





To satisfy the demand for a more complex hierarchical model of the sheep of the last homework, I decided to develop a robot in which most of the components were animated and which showed its own hierarchical structure.

The model consists of cubes, cones, spheres, pills, arcs of spheres and a torus. Each of them is linked to another with a parent attribute that each mesh has. The largest cube in the center (body) is the root parent of Anibot.

In order to make the model more realistic and less fake, i rounded the corners of the body by intersecting a cube and a sphere.

On the front central body there is a screen that transmits a video, while on the back there is a propeller that rotates when the robot moves forward and backward. The propeller is composed of a pill to which a sphere with emissive light is attached to one of the ends: red if the robot does not move forward or back, green during movement. The sphere is the parent of the propellers and by turning the sphere also the propellers which are arcs of sphere turn.

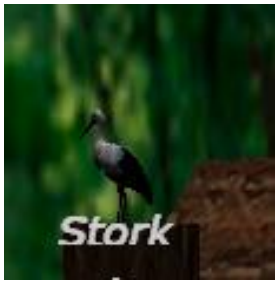
In the upper part of the body is attached the head which is composed of a red sphere, two spheres for the eyes with blue emissive light. In turn, the head is parent of the antenna which also has a yellow emissive light when the robot does not fly high, while it turns orange during height flight.

Two arms are attached to the left and right of the body which are composed of a cylinder which is parent of a pills. Attached to the pills is a torus and a sphere which has a coordinating emissive light like that of the helix sphere. Furthermore, the pills move backwards when the robot goes forward, they move forward when the robot goes backwards and remain vertical when it is stationary.

Finally, in the lower part of the body there is an imported propeller from which the smoke comes out. Smoking is a Particle System set up in such a way as to create a wake effect when moving.

## Some Imported Models





## Camera

In the game scene there is a follow camera. ”Follow” means that it’s possible to define a mesh as lockedTarget and the camera will center the vision to it. The chosen mesh is the imported ”robbox”, so every time the player moves it, camera will translate with it.

Camera has common attributes(radius, heightOffset, rotationOffset) but i decided to limit them in order for example to not permit to see under the ground or to evade from the skybox with a big radius value. It’s possible to move the camera clicking and dragging with the mouse .

## Lights and shadows

## Lights

There are one  Point-lights, one Hemispheric-light and two Directional Light. A point-light is a light source positioned in a point and it emits light in all directions. A hemispheric light is an easy way to simulate an ambient environment light. A directional light is defined by a direction.

The lights are defined based on the position of the robot.

## Shadows

Shadows are entrusted to Shadow Generator function that takes in input a light source.

There is a shadow generator for each light. Every mesh that the generator have to map must be added to the shadow caster list. Only robbox shadow is mapped for optimization. Shadows are handled with a blur exponential mapping.

In conclusion it's needed to set receiveShadows value to true in every mesh material that you want they catch shadows(ground, largeGround).

## GUI

Babylon.GUI uses a DynamicTexture to generate a fully functional user interface which is flexible and GPU accelerated. In this application it is fullscreen so it will cover the entire screen and will rescale to always adapt to your rendering resolution. Gui is composed of Rectangles and Textblocks that displays fps, guide and labels of the animal. All these components are above the screen and will follow you in the game. Animal names are visible based on the distance of the robot from the animal.

## Texture

In Babylonjs textures are applied to a material. There are different types of materials(grassMaterial for example) and different type of texture. Every object, model or mesh in the application has textures. The most used material is StandardMaterial and texture are images or colors applied to it. To define what material is linked to a mesh is needed to fill the attribute "material" of the mesh. The world is closed within a skybox to which sky images are applied as texture. Some meshes have the textures already downloaded, while others have been uploaded by me and are located in the textures folder.

## Animation

Babylon.js has its own method to animate properly objects. It use frames, keyframes and framerate to generate packaged animations. It doesn't support tweens.js library suggested in requirements so i decided to make animations by myself translating and rotating meshes in registerBeforeRender function and using physic features. Every time i discuss about axis in this section i am referring to local axis (axis relative to single mesh).

## Dog

The dog is visible at the beginning of the scene and you can see how it walks around the fire in a loop.

## Rabbit

The rabbit jumps forward and is possible by moving along the y axis and x axis at the same time.

## Anibot

Based on the button clicked, the various parts of the robot move as I described above.

## Duck and Turtle

The duck and the turtle make a translation along the z-axis in a loop. The turtle on the grass of LargeGround while the duck on the water at the bottom of the skybox.

## Physic

To enable physic in Babylonjs it's only needed to apply to scene the following function: scene.enablePhysics(gravityVector, physicsPlugin) where gravityVector is a 3-dimensional vector that define the direction and the intensity of the simulated gravity acceleration and physicPlugin is Cannonjs.

To allow interaction between objects, the physics engine use an impostor, which is a simpler representation of a complex object. The imposter can be assigned physical attributes such as mass, friction and a coefficient of restitution. Cannon has a lot of different impostor shapes(box, sphere, ground, particle, plane, mesh) but to make simple computations i used only boxes, spheres and plane.

The physics has been applied only on some elements for example: robot, a box on the woods to prevent the robot from entering, box at the beginning and end of the skybox which are invisible and on some buildings.