

**-Games Programming 2-**

**Coursework**

**Module number:** M3I626039-20-A.

**Name:** Alessio Scisci.

**Student ID:** S1826635.

**Disclaimer**

*“I confirm that the code contained in this file (other than that provided or authorised) is all my own work and has not been submitted elsewhere in fulfilment of this or any other award.”*

***Alessio S.***

Contents

[Project Showcase 3](#_Toc61711905)

[Overview 3](#_Toc61711906)

[Code Flow 4](#_Toc61711907)

[Meshes & Textures 4](#_Toc61711908)

[Collision Detection 5](#_Toc61711909)

[User Inputs 5](#_Toc61711910)

[Audio 5](#_Toc61711911)

[References 6](#_Toc61711912)

# Project Showcase

## Overview

Throughout the use of each laboratory’s instructions, a core architecture was fully developed, in order to create a simple, straightforward, and engaging videogame, that utilises SDL and OpenGL/OpenAL libraries.

The programming language used in this coursework is C++ and each class works together with the others, embedded in the system, so that the main game loop method is able to visualise and display all the necessary game object components.

Afterwards, the game concept was established, an endless flight experience, set in a forest, where a woodpecker is trying to migrate to a better environment as fall arrives. A simple condition is arranged, as long as the bird is flying the player will get points and it will increase their final score, if a collision with a tree is detected, meaning that the bird hit the trunk too harshly, it will faint, giving a conclusion to the migration flight.

Immagine che contiene pianta, foglia, felce

Descrizione generata automaticamente

*‘Forest Flight’ Overview Showcase – 2020*

## Code Flow

In order to achieve the final outcome aforementioned, mesh & texture rendering, collision detection, user input and audio elements were absolutely vital.

### Meshes & Textures

For the sake of displaying the latter on the screen, as a matter of fact, a display class that creates and set the boundaries of a window screen was implemented, together with a camera class is then possible to render a projection of the camera view directly inside the screen.

Immagine che contiene testo

Descrizione generata automaticamente

*‘Display Class’ code snap – initDisplay method – 2020*

Immagine che contiene testo

Descrizione generata automaticamente

*‘Camera Class’ code snap – header initCamera method – 2020*

Then, textures are loaded, specifying their location path, and bound using the OpenGL texture parameter and bind methods. A similar process is done with the meshes, with the only difference that if either a mesh is locally generated or loaded up as a 3D model, firstly the number of vertices needs to be established and in order to bind the mesh, vertex arrays and buffers are used instead.

Immagine che contiene testo

Descrizione generata automaticamente

*‘Texture Class’ code snap – init method – 2020*

Immagine che contiene testo

Descrizione generata automaticamente

*‘Mesh Class’ code snap – initModel method – 2020*

A vertex and a fragment shader are then initialised from folder path throughout the shader class to wrap everything up, adding any particular effect, and then updated based on the transform values of the meshes and the projection perspective of the camera.

Immagine che contiene testo, verde, screenshot, argento

Descrizione generata automaticamente

*‘Shader Class’ code snap – init method – 2020*

### Collision Detection

Utilising a struct in the mesh class, a collision mesh sphere with a certain radius is instantiated and can be placed in the same transform position of a specific mesh. The collision method takes in as arguments those two values of two different spheres, and simply calculates the distance between the latter with a trigonometric formula, if the spheres overlap, meaning that they collided with each other, the game-state is changed and the game over is executed.

Immagine che contiene testo

Descrizione generata automaticamente

*‘MainGame Class’ code snap – CheckCollision method – 2020*

### User Inputs

The transform class, already mentioned beforehand, is the one responsible to handle the position, rotation and scale vectors taking in consideration the matrices of these elements regarding the mesh models. This implies that changing the value of those variables, since the transform will be applied to the 3D mesh model, will consequently update the position, rotation and scale of the model itself.

Immagine che contiene testo

Descrizione generata automaticamente

*‘Transform Class’ code snap – header Transform & GetModel methods – 2020*

With the use of the mesh movement method inside the main game class and the windows library, it is possible to check if the user pressed a specific key throughout the virtual keyboard; based on the key pressed the position of the woodpecker mesh is then updated with the new values. The same concept is applied for moving the camera; the trees and falling leaves are moved autonomously using the same criteria but without user interaction. (the input can be achieved also by using the event system of the SDL library).

Immagine che contiene testo

Descrizione generata automaticamente

*‘MainGame Class’ code snap – MeshMovement method – 2020*

### Audio

In order to have background music and sound effects inside the game, an audio class was added into the project solution. It can load a Wave audio source file from the location path provided, reads in the text format lines of the sound file, setting the buffer up based on the bit formation of the lines. The audio source is then reproduced utilising the OpenAL library method load audio, taking in consideration the source ID and the vector position of the spot the audio should be played in terms of virtual space. This is done for all the sounds inside the game, particularly for the background music as the soundtrack is played inside the game loop to give the illusion of continuity, surely checking if the track is already playing before reproducing it again.

Immagine che contiene testo

Descrizione generata automaticamente

*‘Audio Class’ code snap – loadWAV method – 2020*

# References

All textures and models have been properly modified, to some extent, in order to fulfil the game purposes.

*Woodpecker Bird 3D Model:*

[*https://free3d.com/3d-model/humming-bird-74440.html*](https://free3d.com/3d-model/humming-bird-74440.html) *.*

*Woodpecker Feather Texture:*

[*https://www.pinterest.com/pin/387802217898313795/*](https://www.pinterest.com/pin/387802217898313795/) *.*

*Maple Tree 3D Model:*

[*https://free3d.com/3d-model/maple-tree-262328.html*](https://free3d.com/3d-model/maple-tree-262328.html) *.*

*Tree Bark Texture:*

[*https://barnimages.com/tree-bark-texture/*](https://barnimages.com/tree-bark-texture/)*.*

*Background Leaf Texture:*

[*https://www.everypixel.com/q/forest-floor-texture*](https://www.everypixel.com/q/forest-floor-texture) *.*

*Maple Tree Leaves 3D Model:*

[*https://free3d.com/3d-model/maple-leaf-v1--353224.html*](https://free3d.com/3d-model/maple-leaf-v1--353224.html) *.*

*Falling Leaves Texture*

[*https://www.hiclipart.com/free-transparent-background-png-clipart-jdtau*](https://www.hiclipart.com/free-transparent-background-png-clipart-jdtau) *.*

*Forest Ambience Sound:*

[*https://bigsoundbank.com/detail-0100-forest.html*](https://bigsoundbank.com/detail-0100-forest.html) *.*