Bring ideas to life
VIA Student Incubator

PROJECT REPORT
COURSE ASSIGNMENT
SGM I1

Alexandru Diaconu **143481**
Zoltan Szabolcs Cseri **169362**
Jonas Nikolajsen **231727**

Henrik Kronborg Pedersen
Michael Vilhelmsen

ICT Engineering
12.05.2016

CONTENT

1. Introduction

In this project we were asked to create a game for the pc, which had a special controller. The controller was a real life Tuk Tuk which we had to modify with hardware inputs so we had controls on the Tuk Tuk as inputs for the game. We were allowed to make the game with VR (Virtual Reality), but didn’t manage to implement it so everything was working. Our process of work is displayed and explained in the model below.
The final product is a game called Tuk Tuk the Rush where the player will drive a Tuk Tuk though different levels to save historical monuments. We have thereby fulfilled most of the requirement (All the must and should) that was set for this project.

2. Developing model

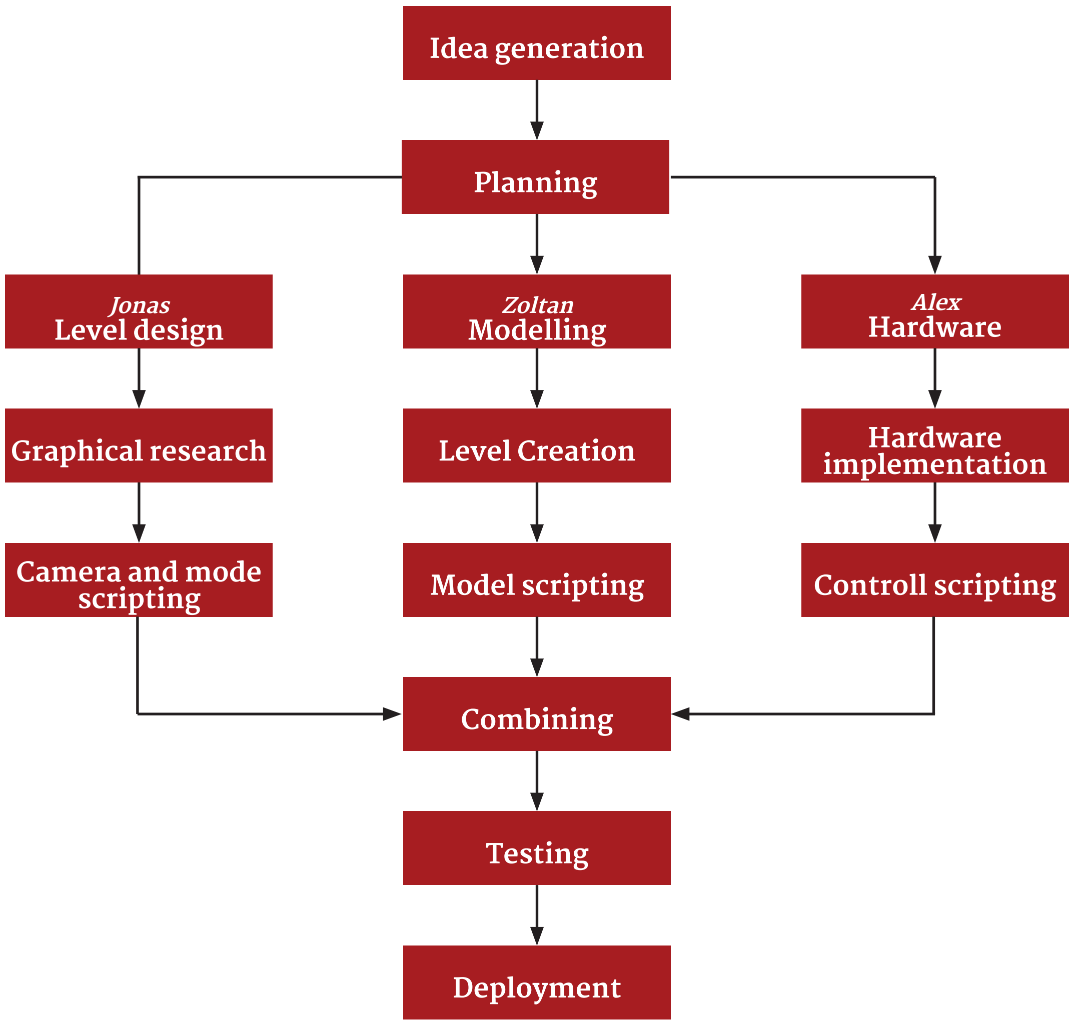


Fig 1. *Developing model*

The project will follow the developing model above. This model is used to delegate the work, but bear in mind that there are loose lines between all of the elements in this model. Alex could help Zoltán with some work on the level creation and the other way around. There are also elements of the developing process that is not in this model. All three group members have taken part of creating the finale edition of the game.

3. Design (10 pager)

3.1 Page 1

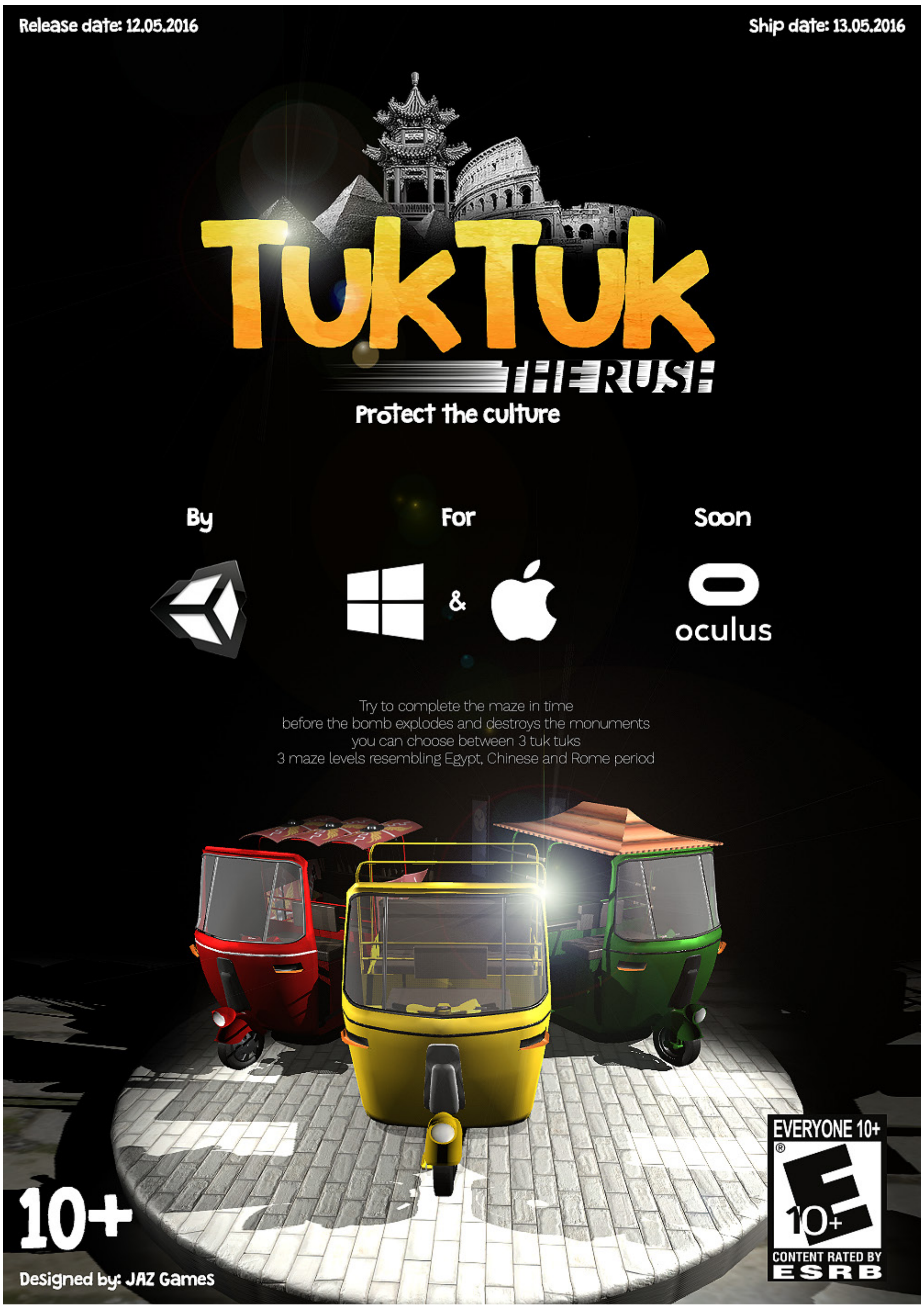

Release date: 12.05.2016
Ship date: 13.05.2016
TukTuk
THE RUSH
Protect the culture
By
For
&
Soon
oculus
Try to complete the maze in time
before the bomb explodes and destroys the monuments
you can choose between 3 tuk tuks
3 maze levels resembling Egypt, Chinese and Rome period
10+
Designed by: JAZ Games
EVERYONE 10+
CONTENT RATED BY
ESRB

Fig 2. Game Poster

3.2 Page 2

Story / Game Summary

TUK TUK the Rush brings awareness about preserving and protecting the cultural environment around the world. Being aware of an imminent danger the player has to find a way out from the mazes that are placed on the historical site of different monuments. The TUK TUK vehicle has a bomb that detonates in a certain amount of time. If the player reaches to it's destination (outside the maze) than the bomb is defused and the cultural site is safe.

During the game, the user has 3 bonus options that he can use to facilitate finding the good path, faster. However, every bonus lasts only for a few seconds and can be used only once per level. Therefore the player can stop the time and than navigate for a few seconds. After the time expires, the TUK TUK return to its original position from when the button was pressed. Another bonus is flying a drone for a few seconds in order to have an overview of the maze. The third and last option is the Road Run. This option is available only in the China and Rome levels. When pressed, the option proceeds in showing the good path by firing up torches situated on the walls. Therefore if you are on the good path you will see the torches light up, if not you will know that you are on a wrong path.

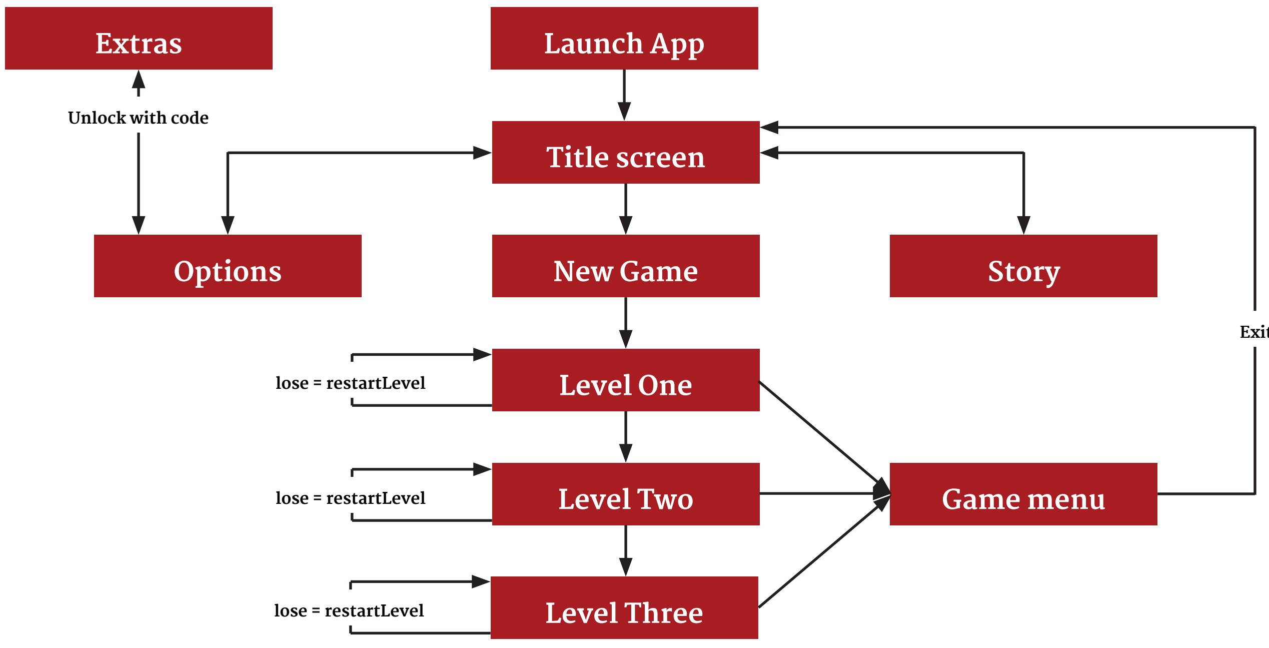


Fig 3. Game Description

Game Flow describtion

Running the .exe file will launch the app and bring the player to the title screen. From the tittle screen the player have four options: New game, Options, Story, Quit. New game will launc the first level of the game. By completion of this level, level two of the game will be loaded. If the player fails to complete the level will restart and the player can try again.

Options takes the player to another menu where you can select graphic and sound options. With an unlock the player get access to the Extras menu where a surprise is hidden.

The Story brings the player to a third menu where a looping story telling tells the player about the monuments he/she is trying to save.

3.3 Page 3

The game only has one character, the player.

Player: (Jonas)
The player is represented as a guy from the National preservation group. He is trying to rescue some of the old historical monuments around the world from destruction. The player will control this character driving a Tuk Tuk through maze obstacles trying to escape certain monuments with a bomb, and thereby getting it to safety to disarm it.

The Tuk tuk will work as an extension of the player. Changing skins each level to represent the eviroment you are playing in.

Character control: (Alex)

Handling the character is done by using the physical TUK TUK input controls.

Steering and Acceleration - steering the handler from right to left allows the character to rotate to right or left. By switching the right handler towards the user, it allows to input the acceleration for the vehicle.

Gear shifting - as start point, the gear system is set to neutral. as long as the input gear button is pressed the gear index moves back and forth: from F (forward) to N (neutral) to R (reverse) and backwards. The gear handler is located on the right Tuk tuk physical handler. The position of the gear index can be noticesed with the help of the thre LEDs, each gear position having it's own color.
Forward-> Green
Neutral ->Yellow
Reverse -> Red

Options input- on the left side of the handler, there are 3 switching buttons representing the input activation for the 3 game bonuses. The activation process triggers also the LED switching for each one of the buttons. Therefore, if a button is pressed, the LED, corresponding to that button switches on.

Breaking- breaking the vehicle is performed by pressing the physical pedal from the bottom right side.

3.4 Page 4

Game outline (Jonas)

Serious action-adventure game
The serious part of the game is the part where the player is trying to save old historical monuments from extension by bad people there is trying to blow them up with a bomb. Between each level, there will be a loading scene with an educational story about the environment the player is going to play in next.

This is where the game transition into the action part. The player has an amount of time to get the bomb away from the monuments.
The adventure part of the game is the player driving around in old historical environments. Like you will see in games like Indiana Jones.

Main Gameplay (Jonas)

Tuk Tuk the rush: Is a maze obstacle race against time. The player has to escape the maze before the time run out to save the monument. During the escape the player have three functionality's to assist him/her to escape the maze. (see the chapter Page 7)

All the controls in Tuk Tuk the rush is optimized for using a real life Tuk Tuk as the controller for the game. This is done by the use of an Arduino and hardware inputs. (see chapter Hardware for further explanation).

Fall back controls is also implemented.
As levels are completed they are unlocked for free play. Thereby the player can freely choose between the levels he/she have completed. And replay them to practice or to set a faster time.

Platform specific features

It is planned to implement the the real life Tuk Tuk controller together with VR(Virtual Reality). By doing this the player will have the real experience of sitting in the Tuk Tuk and driving it around inside the game. This is a platform specific future because the player need the Tuk Tuk and the hardware inputs to be able to try this.

The fallback controls will allow the game to be played on normal inputs as keyboard or a gaming controller.

3.5 Page 5

Game World

The game world is setup in different iconic eras of History such as Ancient Egypt, Medieval China and Rome where the player have to save the monuments of that time from destruction. The scenes are based on mazes placed near Monuments. In the prototype of the game, three levels are designed, each level with their own design based on a specific monument. All three levels are implemented with a similar concept:
• The maze walls
• Light sources (small sources along the road. When the helping function Road Run is activated these sources will light up the right path.)
• Directional Light to give to base lightning for the scene
• Environmental background

Level 1:
Level one is taking place in the old Egypt. The mission is to save the old pyramids and statues from destruction. The maze in this level has a triangular design to symbolize the pyramids. The walls are built with the Egyptian design by representing hieroglyphs and wall reliefs. The environment is based on the authentic setup of desert with the Pyramids as well as some huge statues around the maze to give a monumental experience.

Level 2:
Level two is placed in the Chinese Empire, where the mission is to save the Great wall. This maze has a square design and the walls in this level are based on traditional Chinese architectural design from the Medieval China. The landscape outside the maze the player will experience green forest and the Great wall running across the mountains. The maze is filled with trees and vases to honor the Chinese gardens.

Level 3:
Level three is placed in Rome, taking place inside the great Colosseum. The maze design for this level is ellipsoid to fit inside the Colosseum model. The walls are built to match the old Roman Empire walls where the columns and the bras reliefs were used as decorations. Also the lightning along the way when the Road Run us active is illuminated with the torches attached to the walls. As the maze is placed in the Colosseum it raising high around the player where the spectator area can be seen.

For future aspect of the game many different places and time periods can be covered such as:

• *Hanging Gardens of Babylon, Babylon*
• *Colossus of Rhodes, Greece*
• *Pyramid of the Sun, Mexico*
• *Taj Mahal, India*
• *Eiffel Tower, France*
• *Statue of Liberty, USA*
• *Hagia Sofia, Turkey*
• *Leaning Tower of Pisa, Italy*
• *Stonehenge, England*

3.6 Page 6

Gameplay experience

The first thing what the player sees is the start screen, the main menu with the options: New Game, Options, Stories and Quit. The New Game option starts the game in the first levelThe Option button gives access to the option menu where different options can be chosen and an extra feature is implemented (See the chapter Page 7 for more information) The Story button gives access to the story menu. Here is a story telling about each monument the player will save during the game. The Quit option is exiting the game.

When the player starts the game the gameplay begins with the player placed behind the steering wheel in the TukTuk. From here he has to escape from the maze before the countdown reaches zero. The world is set back to the corresponding era of the scene with the atmospheric music and the environment.

The overall feel of the game and its world of classic American action-hero movies with a bit of twist with the timeline where the fate of history is at stake and the danger of blowing up will create pressure on the player.

During the gameplay a countdown is displayed which informs the player how much time is left before the bomb will explode.

The Tuk Tuk is equipped with three special abilities which puts the science fiction elements into the game. (See the chapter Page 7 for more information about the abilities).
The thrill of finding the path before the bomb goes off and the joy of finding the exit creates a unique experience for the player. By finding the first exit the player is teleported into another level to save the next monument. With each level the maze is getting harder which builds up the pressure as well as the satisfaction of level completion.

The sounds, textures and models are matching the scenes to create an environment that gives the feeling of the historical time the player is presented in.

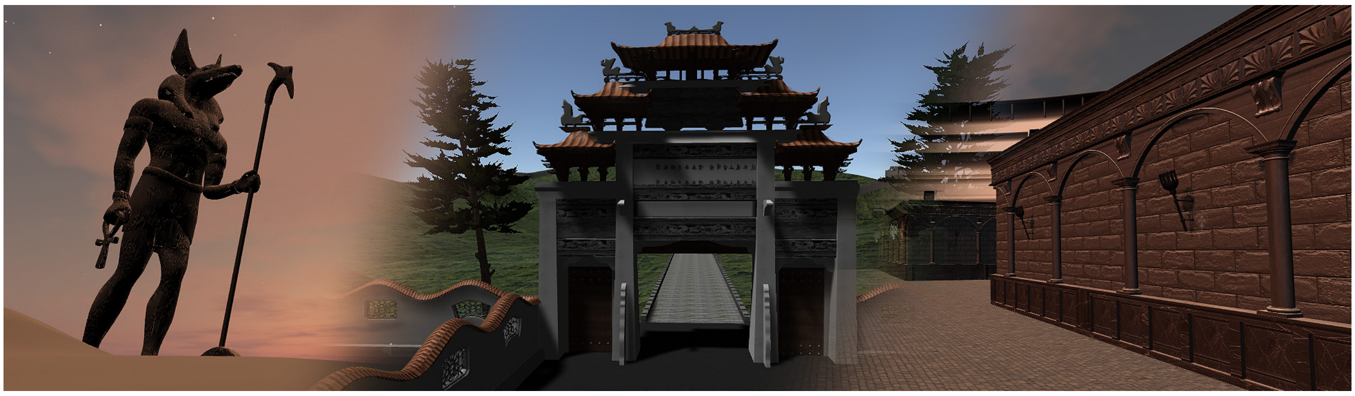


Fig 3. *Egypt, China & Rome levels*

3.7 Page 7

Mechanics

The main mechanic in "TUKTUK the Rush" is the Tuktuk what is driven by the player through a maze which is able to accelerate, brake and turn. By turning in sharp angle fast it can roll over, due to the fact it only has three wheels and therefor isn't very stable sideways, so the player has to be careful. To assist the player completing the maze three tools is presented; Drone View, Holo Stretch and Road Run (See description below). These three tools are usable one time each for all levels.

Power-ups

• **Drone View**
When activated the camera change position and fly up into the air, giving the player the view of a drone. Thereby the player has a certain amount of time to look around and study the maze to find the right road out of the maze.

• **Holo Stretch**
When activated the Tuk Tuk enters a holographic mode, stopping the time and giving the player some free time to drive around the maze without losing any time. When the ability runs out, the Tuk Tuk position is set to the same place the player activated the ability and time starts to count down again.

• **Road Run**
When activated the light sources in the scene is activated to display the right road to get out of the maze. The player then has a certain amount of time to follow the up lit road.

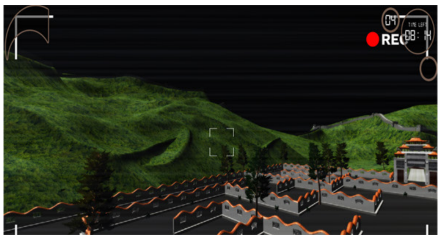


Fig 4. *Drone view*

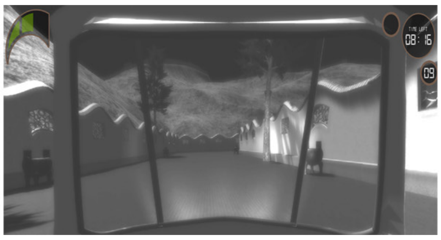


Fig 5. *Holo stretch*

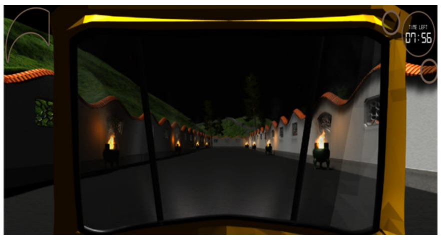


Fig 6. *Road Run*

3.8 Page 8

Enemies

This game doesn’t have AI or attacking enemies. The enemy in this game is the time and the players own actions and decisions. The game is based on a race against the time. If the player doesn’t manage to complete the level within the given time he/she dies. The bomb the player is trying to escape with, to save the maze, will explode and thereby kill the player and destroy the monument.
The player is his/her own worst enemy if the player takes a wrong turn in the maze time will be lost and thereby making it harder to escape the maze in time without the bomb exploding.

Future development

As part of future development of this project there is two aspects that could be added as “Enemies”

1. Obstacles / Slowing enemies
Obstacles could be added to the game world to make it harder to conquer the maze and escape the level in time.

Obstacles:
- Traps
- Gaps
- Ramps
- Moving walls

Slowing enemies could be another aspect to have as enemies. These would try to disrupt the player’s gameplay by shooting/throwing objects that will slow the drive speed of the Tuk Tuk and thereby valuable time will be lost.

2. Medal race / Competing
Medal race could be added to the game to make the race against time harder/more fun. Together with the medal race would follow unlockable new levels, custom skins and upgrades of helping abilities.

Power ups:
- Nitro
- Invincibility
- Slow time

Competing could be added to make the game more social. Compete against your friends to get the best time or compete against world ranks to have the best time in the world.

All these aspects are put in future development due to deadline time of the prototype.

3.9 Page 9

Bonus material

As one of the serious aspects of the game the story menu selection will have stories for each level that will explain the story behind the specific monument the player will be trying to save in the specific level. This will give the player insight to what he is trying to preserve from destruction. This can work as a motivational factor.

Changing the Tuk Tuk model is an extra feature that is unlocked by a special code entered at the options menu. When the code is entered correctly, it will grant access to the extra scene where the player can choose between three Tuk Tuk models to play with in all the scenes.

The original Via Tuk Tuk
The Chinese inspired Tuk Tuk
The roman inspired Tuk Tuk

When a Tuk Tuk is selected this Tuk Tuk will automatically be placed in all the levels.

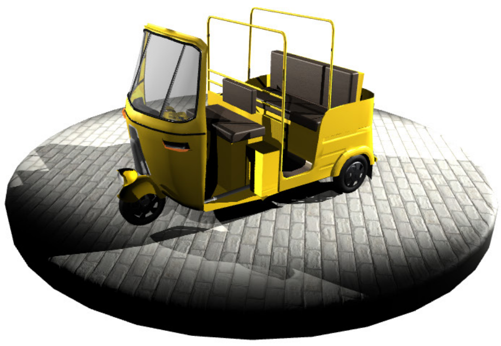


Fig 7. *Tuk Tuk VIA*

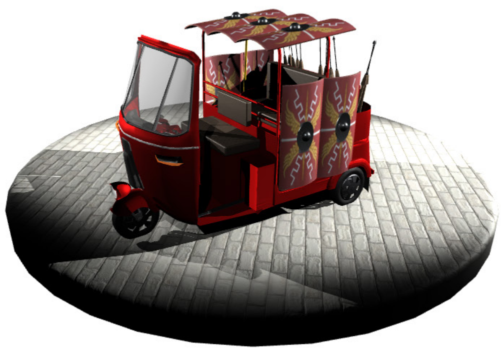


Fig 8. *Tuk Tuk Roma*

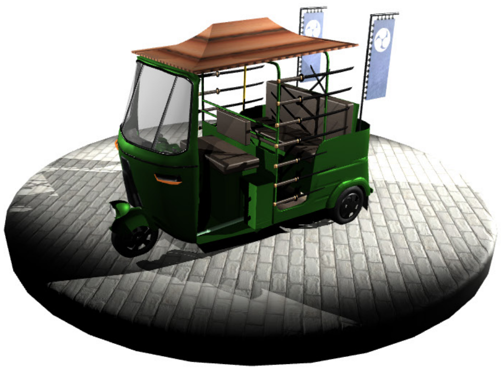


Fig 9. *Tuk Tuk China*

4. Implementation

4.1 Modelling

Level Design

As the main concept of the game is to save the monuments from different time areas the level design was quite easy from that point. Three iconic eras had been chosen for the game to take place: Ancient Egypt, Medieval China and Rome (fig 10). All of these scenes should contain a famous monument which creates the atmosphere.

As the key element of the game is the maze in each level a maze was set up. The player has to find the right route to escape and save the monuments. As an example in Egypt nothing can be more iconic than the pyramids and the desert so these elements had been recreated in the scene together with the maze and some site improvements such as huge Egyptian god statues to create the epic feeling while playing. To create the authentic experience for each scene unique walls had been designed for the era. Taking these into consideration some research had been made how the ancient Egyptian walls looked like in the temples and in the pyramids back in the time.

A low poly model was created with the distinctive features like: hieroglyphs, big stone bricked walls, sandstone columns and wall reliefs. The main objective for the levels was to utilize as few props as possible to smooth the runtime of the game and still create an immersive experience for the player.

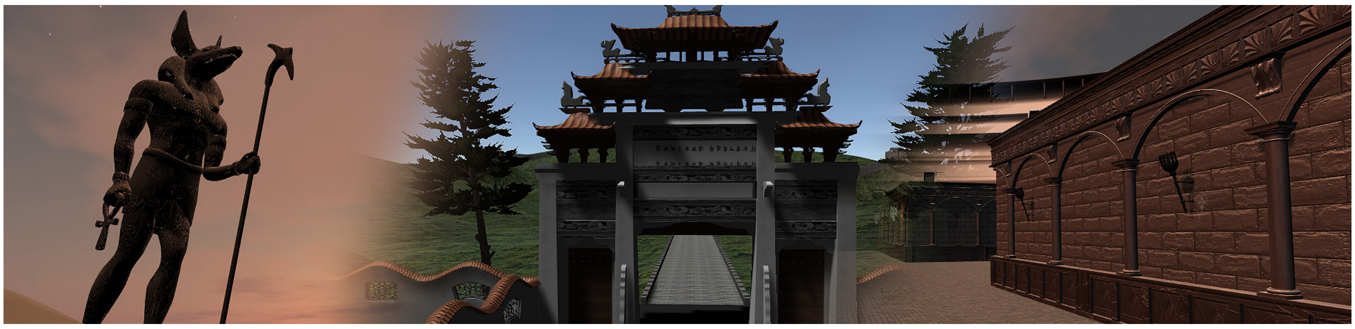


Fig 10. *Game screenshots*

3D model design

The models for the game have been created in Autodesk 3ds Max which is a specially designed tool for 3D modelling and animation. The challenge in the model design was to create detail rich content while keeping the polygon amount as low as possible (fig 11). Models made of vertices or in other name triangles and the more detailed it is the more vertices the model consists of. . In the game the Video Card have to render all these vertices in the scene and if the amount is to big the FPS (Frames per Second) will drop below 60 which will create lags. To avoid that, the faces which are not visible for the player were deleted to reduce the amount of triangles to be processed. The thumb of rule was to create models which are used multiple times in each scene with the polygon count around five to eight-thousand and max eighty thousand polygons where it has been used only once.

When exporting the models from 3ds Max to Unity the format and the vector alignment had to be matched. The standard format for models in Unity is .FBX and the vector alignment that the Y axis is facing upwards, while the standard format in 3ds Max is .MAX and the vector Z is facing up. By exporting the model the .FBX format and the Y axis facing upwards option had to be selected.

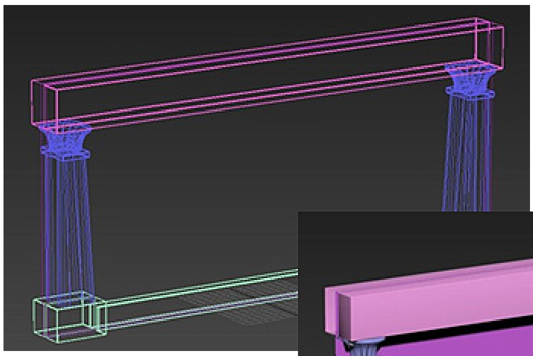


Fig 11. *Wireframe view*

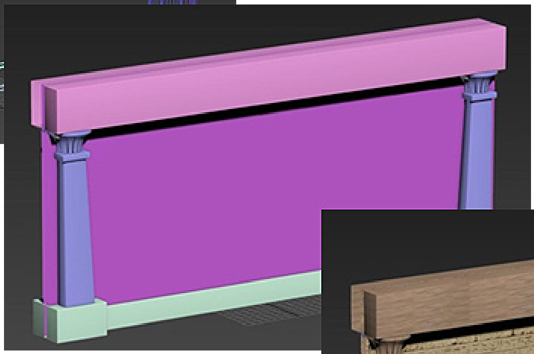


Fig 12. *Static*

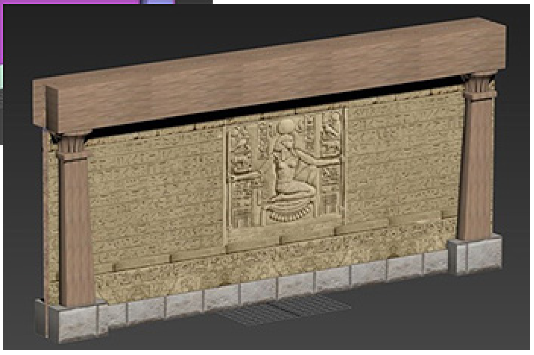
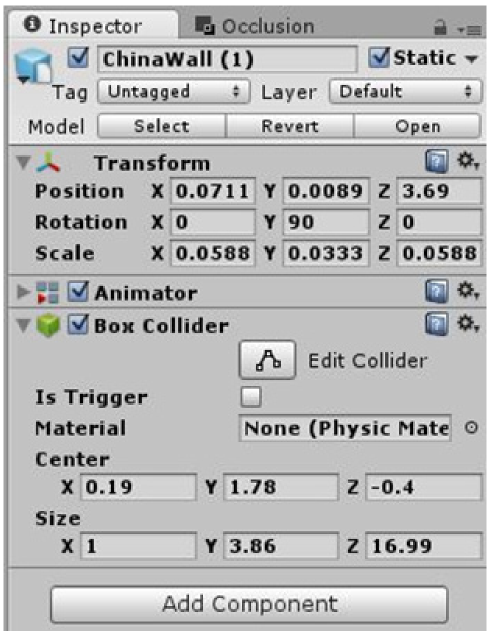


Fig 13. *UV mesh*

3D models to Unity

For adding external models into the scene they have to be imported. The procedure is to right click on the project area, import new asset and select the model to import. To make the model fully compatible with the lightning scene the UV maps has to be generated within Unity, to do that clicking on the model and selecting Generate UV´s and click on Apply button the system will automatically generate the UV´s (fig 13).

Some components had been added in Unity such as Box Collider for preventing the player to go through walls. The behaviour of the of the walls have been modified to Static (fig 12) this improves the runtime of the game by reducing the Draw Calls. Basically the system packs all the static elements and sends just one call instead of many individual calls which greatly smoothens the runtime. The models imported can be easily added by dragging and dropping them into the scene. There they can be freely edited like scale, rotate or move. By copying the same wall prefabs and not putting many unique elements into the scene the performance can be greatly boosted.



Inspector | Occlusion

ChinaWall (1) — Static
Tag: Untagged — Layer: Default
Model: Select | Revert | Open

Transform

	X	Y	Z
Position	0.0711	0.0089	3.69
Rotation	0	90	0
Scale	0.0588	0.0333	0.0588

Animator

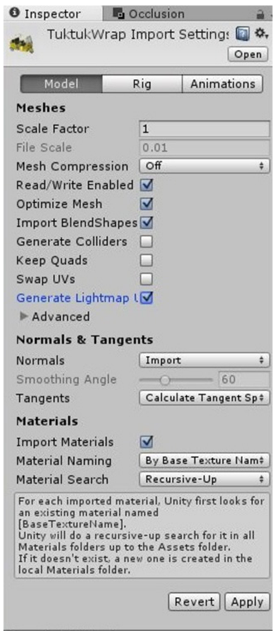
Box Collider

Edit Collider
Is Trigger: ☐
Material: None (Physic Mate

	X	Y	Z
Center	0.19	1.78	-0.4
Size	1	3.86	16.99

Add Component

Fig 14. Wall object



Inspector | Occlusion

TuktukWrap Import Settings — Open

Model | Rig | Animations

Meshes

Scale Factor	1
File Scale	0.01
Mesh Compression	Off
Read/Write Enabled	✓
Optimize Mesh	✓
Import BlendShapes	✓
Generate Colliders	☐
Keep Quads	☐
Swap UVs	☐
Generate Lightmap U	✓

Advanced

Normals & Tangents

Normals	Import
Smoothing Angle	60
Tangents	Calculate Tangent Sp

Materials

Import Materials	✓
Material Naming	By Base Texture Nam
Material Search	Recursive-Up

For each imported material, Unity first looks for an existing material named [BaseTextureName].
Unity will do a recursive-up search for it in all Materials folders up to the Assets folder.
If it doesn't exist, a new one is created in the local Materials folder.

Revert | Apply

Fig 15. Import settings

Materials and Texture

Materials are a vital part of the game because it creates the realistic look of the game. By standard each object in the scene is assigned with the Standard Material which is a normal, grey colour without any specific modifications. By creating different materials with unique properties and assigning them to the models will bring the props alive.

To create a material, it is a good idea to use textures or in other name images which will give the characteristic of the model (fig 14). By adding a Normal Map as well to the Material the realism factor will be boosted as it mimics the bumps and tweaks of lightning on the surface of the object. Moreover, by modifying the Smoothness and the Metallic properties of the material it is possible to create a close to real-life experience (fig 15).

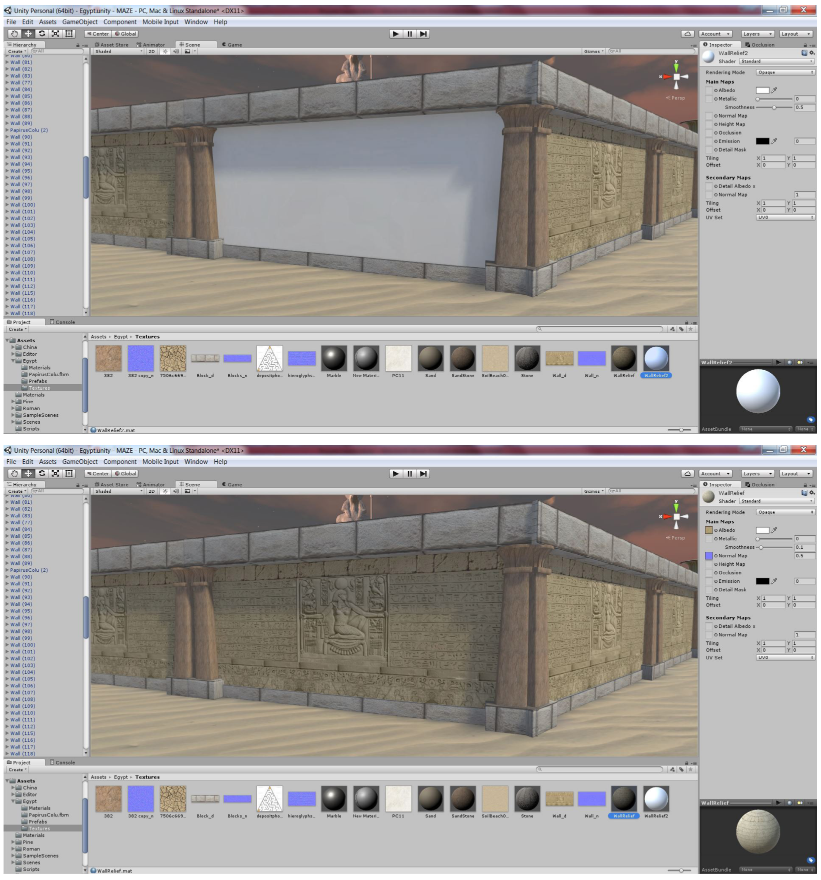


Fig 16. Apply material

Lighting and Shadows

In order to create a realistic look it is necessary to put lights and shadows into the game. It will create the real life impression for the user because there is nothing in the world without shadows and lights. The light comes from a Light source in Unity, it can be: Point light, Spotlight, Area light and Directional light. For most of the scenes a Directional Light have been used as main Light Source, which basically mimics the behaviour of the Sun, an point light which is infinitely far away. It will create the shadows in the scene but all of them are completely black as they don't have lights bouncing off from the other surfaces (fig 16).

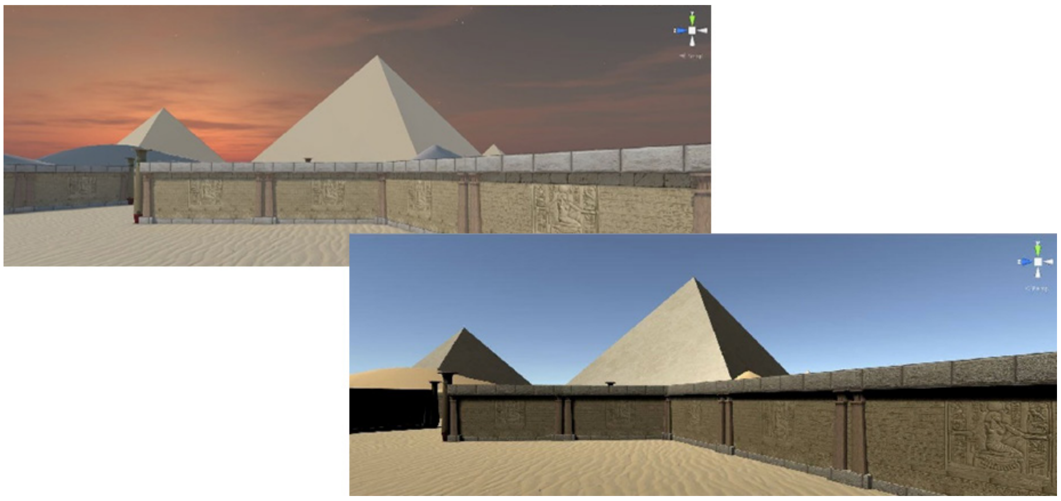
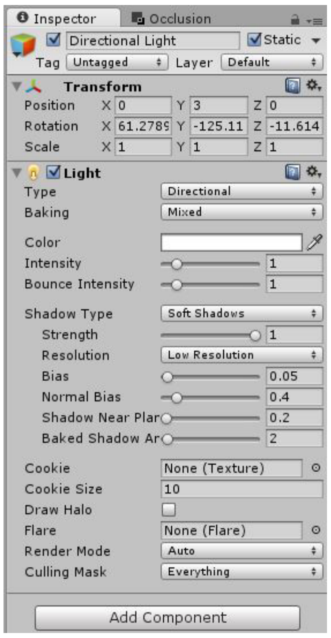


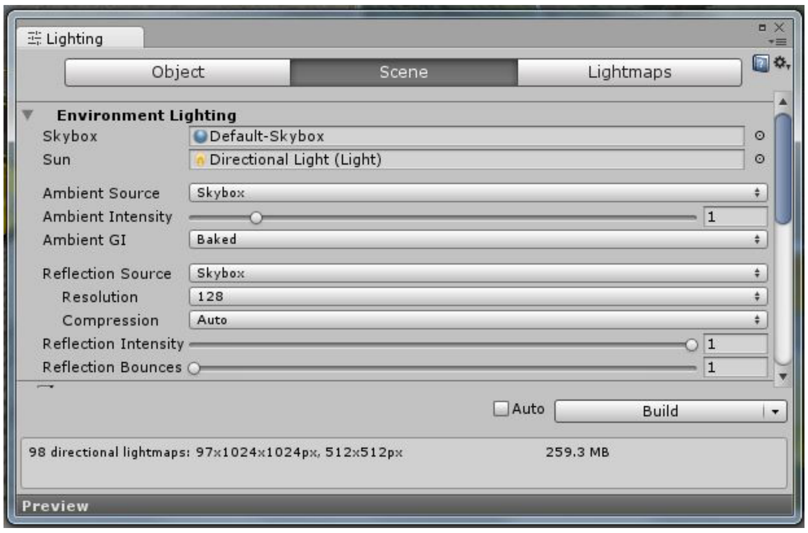
Fig 17. *Shadow baking*

To create a realistic look the Global illumination or GI has to be turned on. To enable it in the Directional light, the settings for shadow baking had to be set up to mixed, so it will use Real Time as well as pre-baked shadows for the game (fig 17). Real Time Shadow are extremely expensive performance wise, that is why it is good idea to pre-bake some of them in the scene which won´t change during the gameplay. In the same place it is possible to setup the amount of bounces, the colour of the directional light and the intensity. For the final step in the Lightning window the lights and shadows had to be baked and depending on the scene it will take a while (fig 18). In our game the scenes are supported with point lights as well as they are used for illuminating the area around the fire prefabs. They behave differently as they have another parameter called range. This is defining how far the light should affect the area around it.



```
[✓] Directional Light        [✓] Static
Tag  Untagged    Layer  Default

Transform
Position   X 0        Y 3        Z 0
Rotation   X 61.2789  Y -125.11  Z -11.614
Scale      X 1        Y 1        Z 1

[✓] Light
Type                 Directional
Baking               Mixed
Color
Intensity            1
Bounce Intensity     1
Shadow Type          Soft Shadows
  Strength           1
  Resolution         Low Resolution
  Bias               0.05
  Normal Bias        0.4
  Shadow Near Plar   0.2
  Baked Shadow Ar    2
Cookie               None (Texture)
Cookie Size          10
Draw Halo            [ ]
Flare                None (Flare)
Render Mode          Auto
Culling Mask         Everything
```

Fig 18. *Light properties*


Lighting

Object | Scene | Lightmaps

Environment Lighting
Skybox: Default-Skybox
Sun: Directional Light (Light)
Ambient Source: Skybox
Ambient Intensity: 1
Ambient GI: Baked
Reflection Source: Skybox
Resolution: 128
Compression: Auto
Reflection Intensity: 1
Reflection Bounces: 1

☐ Auto | Build

98 directional lightmaps: 97x1024x1024px, 512x512px 259.3 MB

Preview

Fig 19. Light baking

4.2 Hardware

In order to control the character by using unconventional inputs, several parts had to be installed for different processes regarding the user experience. Therefore, several electronic accessories like potentiometers, LEDs, switch buttons were installed on the physical Tuk Tuk. These allows an user to have a similar experience as driving an actual vehicle. The communication between the game engine and the user inputs is made with the help of an Arduino board.

Arduino

Regarding the virtual logic, the entire circuit is devided intro 4 main parts: Steering and acceleration, gear shifting, options input and breaking. All 4 parts, are receiving inputs from different sensors. These inputs are transformed and manipulated in order to be ready to sent to Unity Platform. Bas-sically, the communication is done by using the same serial port read-write in both platforms. The serial inputs processed by the Arduino board is then sent over the port to Unity, where it reads it line by line. There are a total of 11 variables sent from one platform to another (see fig)

```
  Serial.println(
    String(stateStart) + “ “ +
    String(h / 100) + “ “ +
    String(v / 100) + “ “ +
    String(readingGear) + “ “ +
    String(stateReverse) + “ “ +
    String(stateNeutral) + “ “ +
    String(stateForward) + “ “ +
    String(stateOption1) + “ “ +
    String(stateOption2) + “ “ +
    String(stateOption3) + “ “ +
    String(stateBrake));
  delay(20);
---------------------------------------OUTPUT:

0 0.93 0.00 0.03 0 1 0 0 0 1 0  //Button option 2 pressed
0 0.92 0.00 0.02 0 1 0 0 0 1 1  //Button option 3 pressed
1 0.90 0.00 0.04 0 1 0 0 0 1 1  //Button start pressed
```

Fig 20. Arduino code

Unity

These variables are then read and split in different elements in an array on the Unity part. Each element represents an input from a certain sensor.

```
        public class CarUserControl : MonoBehaviour
        {
            public string port;
            SerialPort serial;
            ...

        private void Awake()
        {
            // get the car controller
            serial = new SerialPort(port, 9600);
        }


        private void Update()
        {
                if (!serial.IsOpen)
                    serial.Open();

                //Read String lines from Arduino “console”
                string ArduinoData = serial.ReadLine();
                Debug.Log(ArduinoData);

                //Split the data taking the space as reference
                //and store it in an array (remember the order
                // sent by Arduino Serial Port)
                string[] data = ArduinoData.Split(‘ ‘);

                start = float.Parse(data[0]);
                steer = float.Parse(data[1]);
                accelerate = float.Parse(data[2]);
                readingGear = float.Parse(data[3]);
                reverse = float.Parse(data[4]);
                neutral = float.Parse(data[5]);
                forward = float.Parse(data[6]);
                o1 = float.Parse(data[7]);
                o2 = float.Parse(data[8]);
                o3 = float.Parse(data[9]);
                brake = float.Parse(data[10]);
...
```

Fig 21. *input data reading from Arduino to Unity*

Observations

The port that the Unity has to pair with it has to be the same in which the Arduino is connected with
Receiving and transmiting data within a FixedUpdate() method may cause laging
In order to not overcharge the port with information, a delay of 20 ms had to be placed in the Arduino code before sending new sets of data
Some sensor inputs were the original buttons from the Tuk Tuk vehicle. because of the lack of grounding part, the analog ports (from the Arduino board) had to be used as the input value would never reach 0 (only in case of current fluctuations).Therefore a threshold of 1020 has been set in order to check if the button was pressed or not. the oscilation in an analog input may range between 300 and 1023 units (depending on the voltage and current source).

4.3 Scripting

Controllers

Controller: Controller.cs

Functionality

• The functionality of the Controller.cs script is to set the timer before the explosion occures
• Set the attention timer to warn you that the time is about to end
• Set the drone duration for viewing the overall maze
• Set the explosion effects if the time runs out, as well as restarting the level
• Enable the fire for the good path
• Set up the GUI for the option and gear use
• Changing between tuk tuk models based on the selection process from the extras menu

Considerations

• The controller connects the main process from the input managment to execution.
• To change between the models, the controller is accessing a value of a script that is creating in the
•Menu Scene. However, the script contains the DontDestroyOnLoad(this.gameObject) method, which allows to keep the script "alive" between the scenes.

Output:

```
if (Input.GetButtonDown("Drone"))
            {
                DroneSkill();
            }
            if (droneBool)
            {
                textDrone.SetActive(true);
                droneTime -= Time.deltaTime;
                secondsDrone = (droneTime % 60).ToString("00");
                textDrone.GetComponent<Text>().text = secondsDrone;
            }

            if (Input.GetButtonDown("Holo"))
            {
                Debug.Log("Start Holo");
                HoloSkill();
            }
            if (Input.GetButtonDown("Path"))
            {
                Debug.Log("Start Path");
                PathSkill();
            }

---------------------------------------METHOD EXAMPLE:

void DroneSkill()
    {
        droneBool = true;
        o1.SetActive(true);
        cameraControling.mode2 = true;
        cameraControling.mode1 = false;
        StartCoroutine(timeDrone());
    }

    void HoloSkill()
    {
        o2.SetActive(true);
        holo.startNow = true;
        holo.record = true;
    }

    void PathSkill()
    {
        o3.SetActive(true);
        Debug.Log("Show Path");
        fire.SetActive(true);
    }
```

Fig 22. *Triggering different options*

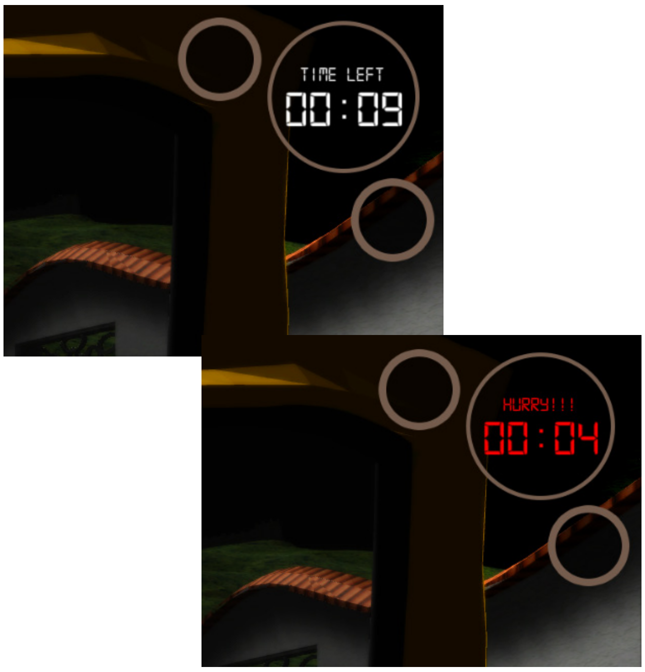


Fig 23. *Normal time vs critical*

```
if (timeLeft < timeAttention)
            {
                text.GetComponent<Text>().color = Color.red;
                label.GetComponent<Text>().color = Color.red;
                label.GetComponent<Text>().text = "HURRY!!!";
            }
```

Car User Control: CarUserControl.cs

Functionality
• The functionality of the Car User Control.cs script is to store all the values received from the Arduino
• Steer and accelerate the vehicle.

Considerations
The Car UserControl.cs script is an import element from the CrossPlatformInput. To be able to move the Tuk Tuk with the inputs from the Arduino sensors, the script was modified accordingly.

Output:

```
namespace UnityStandardAssets.Vehicles.Car
{
    [RequireComponent(typeof (CarController))]
    public class CarUserControl : MonoBehaviour
    {
        private CarController m_Car; // the car controller we want to use
        public string port;
        SerialPort serial;

        public bool tuktukenable;

        private void Awake()
        {
            // get the car controller
            m_Car = GetComponent<CarController>();
            serial = new SerialPort(port, 9600);
        }


        private void Update()
        {
            if (tuktukenable)
            {
                if (!serial.IsOpen)
                    serial.Open();

                //Read String lines from Arduino “console”
                string ArduinoData = serial.ReadLine();
                Debug.Log(ArduinoData);

                //Split the data taking the space as reference
                //and store it in an array (remember the order
                // sent by Arduino Serial Port)
                string[] data = ArduinoData.Split(‘ ‘);

                start = float.Parse(data[0]);
                steer = float.Parse(data[1]);
                accelerate = float.Parse(data[2]);
                readingGear = float.Parse(data[3]);
                reverse = float.Parse(data[4]);
                neutral = float.Parse(data[5]);
                forward = float.Parse(data[6]);
                o1 = float.Parse(data[7]);
                o2 = float.Parse(data[8]);
                o3 = float.Parse(data[9]);
                brake = float.Parse(data[10]);
            }

            // pass the input to the car!
            float h = CrossPlatformInputManager.GetAxis(“Horizontal”);
            float v = CrossPlatformInputManager.GetAxis(“Vertical”);
#if !MOBILE_INPUT
            float handbrake = CrossPlatformInputManager.GetAxis(“Jump”);

        //    float h = steer;
        //    float v = accelerate;
        //    float handbrake = brake;

            m_Car.Move(h, v, v, handbrake);
#else
            m_Car.Move(h, v, v, 0f);
#endif
        }
    }
}
```

Fig 23. *Passing values to game object*

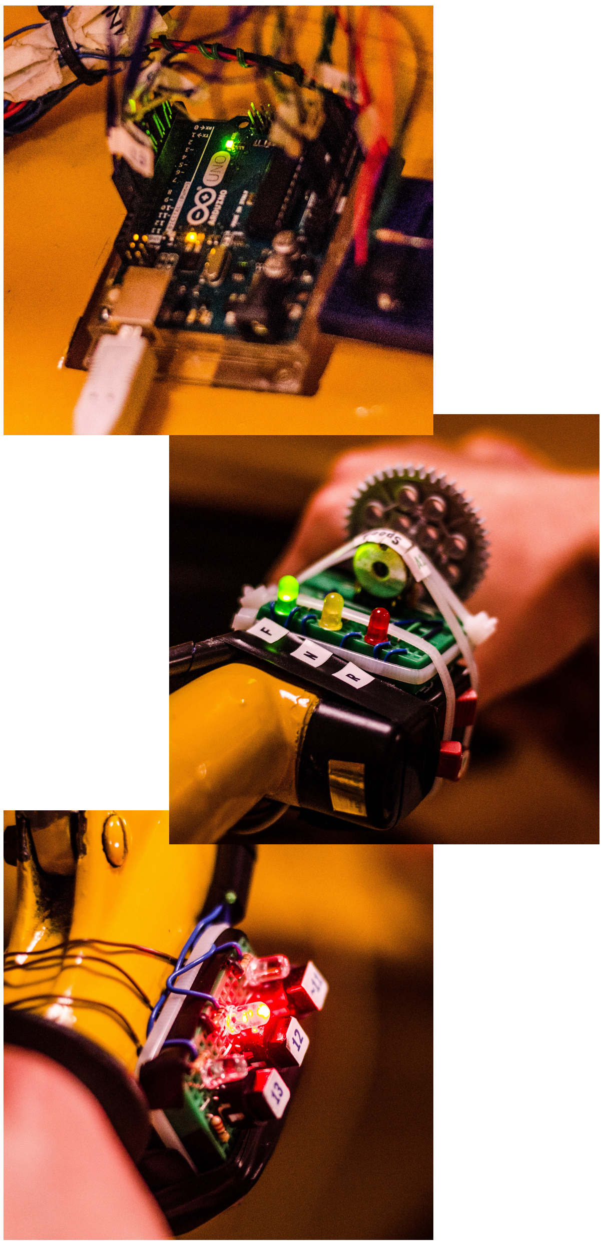


Fig 24. *Different physical input sensors*

Abilities

Holo stretch: Holo.cs

Functionality

· The functionality of the Holo.cs script is to start the event of the holo stretch ability.
· Store the position of the Tuk Tuk
· Pause the current game time
· Start a new timer
· Change camera effects

Considerations

To make a Tuk Tuk that could ride for free without spending time, to ways were considered.
1: Enable a secondary model and set the camera and controls to work on this model, so the player could drive this model around for some time and then return to the original model.
2: Store the position of the original model, put on some camera effects, pause the time and let the player drive this TukTuk around for some time before "teleporting" it back to the stored position.
The second option was chosen and the output was realized the following way.

Output:

```
void Update()
    {
        if(startNow == true)
         {
            if(startNow == true && record == true)
            {
                if (faded)
                    StartCoroutine(startHolo());


            }
            text.SetActive(true);
            timeLeft -= Time.deltaTime;
            seconds = (timeLeft % 60).ToString("00");
            text.GetComponent<Text>().text = seconds;

            if (timeLeft <= 1)
            {
                if (fadedStop)
                    StartCoroutine(stopHolo());
            }
        }
    }
---------------------------------------METHOD EXAMPLE:

IEnumerator startHolo()
    {
        controller.pauseTime = true;
        faded = false;
        fade.gameObject.SetActive(true);
        fade.GetComponent<CanvasRenderer>().SetAlpha(0f);
        fade.CrossFadeAlpha(1f, 0.3f, false);
        yield return new WaitForSeconds(0.3f);
        record = false;
        positionPlayer = player.transform.position;
        rotationPlayer = player.transform.rotation;
        bloomEffect.enabled = true;
        grayEffect.enabled = true;
        grayEffect.rampOffset = 1;
        fade.gameObject.SetActive(false);
        while (grayEffect.rampOffset > -0.05f)
        {
            grayEffect.rampOffset -= 0.01f;
            yield return new WaitForSeconds(0.001f);
        }
    }
    ...
```

Fig 25. *Part of Holo.cs script*

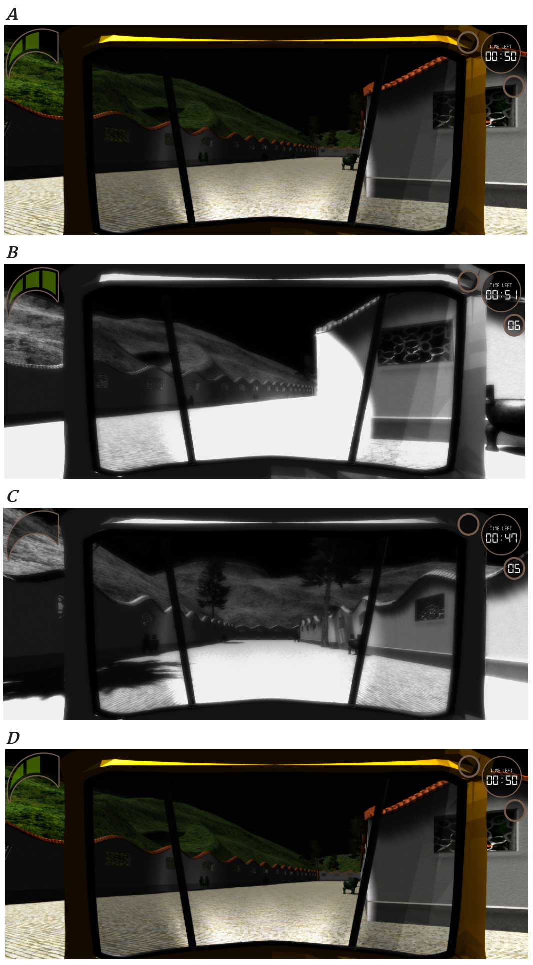


Fig 26. *A: normal B: start of holo C:end of holo D: back to position*

Drone view: Drone.cs

Functionality:
· Implementation of the Drone view ability.
· Activate canvas effect
· Start coroutine
· Move camera position
· Wait for amount of time
· Reset camera position

Considerations:
To make the drone view follow the Tuk Tuk and the camera go back to the right position the use of Empty child gameObjects were applied. These objects holds the position the camera should change to during the script. When the empty gameObjects were placed as children under the Tuk Tuk model, the camera would always follow the Tuk Tuk.

Output:

```
void Update()
    {

        if (mode1 == true)
        {
            if (!cameraChanging)
            {
                Debug.Log("NORMAL CAMERA");
                //      StartCoroutine(MoveOverSpeed(mainCamera, position1.position, 10f));
                mainCamera.transform.position = position1.position;
            }
        }
         else if (mode2 == true)
        {
            if (!cameraChanging)
            {
                mode1 = false;
                Debug.Log("DRONE VIEW");
                if (recStart)
                    StartCoroutine(recBeep());
                //Drone game mode. Camera switch position and floats in to the air.
                StartCoroutine(MoveOverSpeed(mainCamera, position2.position, 10f));
                droneUsed = true; //Drone has been used
            }
        }
    }
```

Fig 27. *Part of Drone.cs script*

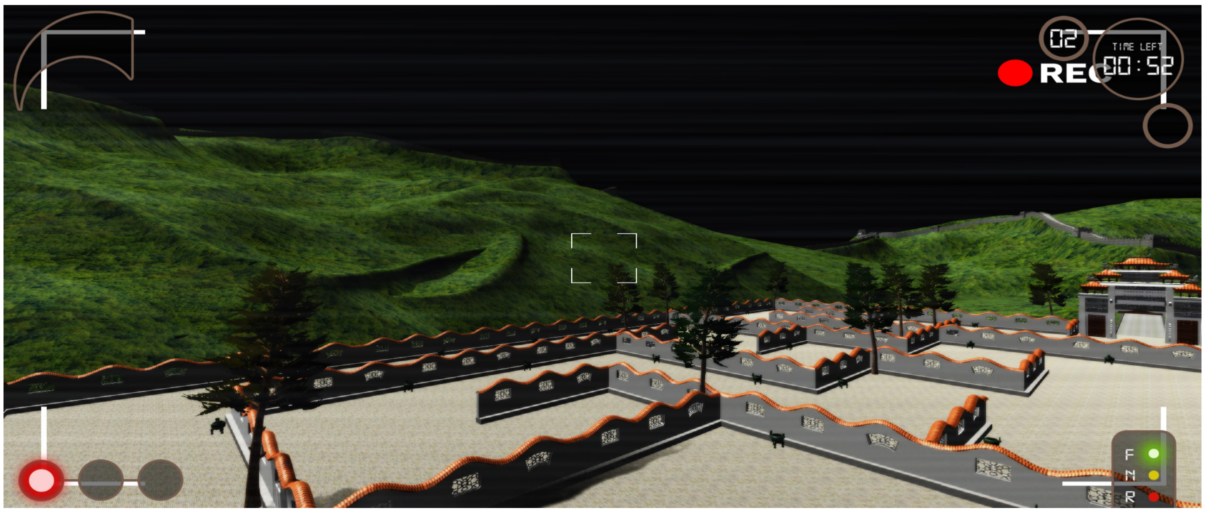


Fig 28. *Drone view*

Road Run: Controller.cs

Functionality

• The Road Run ability isn’t using much code, therefore the few lines are placed in the Controller.cs script
• Activate canvas effect
• Activate fire effects
• Wait amount of time
• Disable fire effects

Considerations

To make the right path light up with the fire effects two groups where made. A group for all the light sources along the wrong path and a group for all the light sources along the right path. Then the code should only light up the right path. This functionality is achieved by enabling all the fires under the right path group and thereby start an effect script that makes the fire burn for an amount of time and then fades out.
Output:

```
if (Input.GetButtonDown(“Path”))
   {
         Debug.Log(“Start Path”);
         PathSkill();
   }

void PathSkill()
   {
        o3.SetActive(true);
        Debug.Log(“Show Path”);
        fire.SetActive(true);
   }
```

Fig 29. *Path trigger*

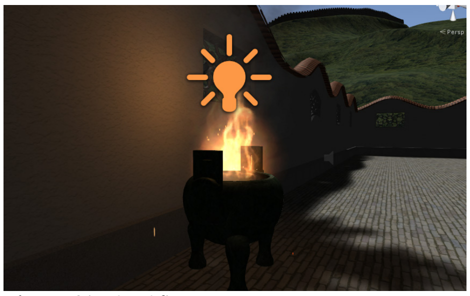


Fig 30. *China level fire*

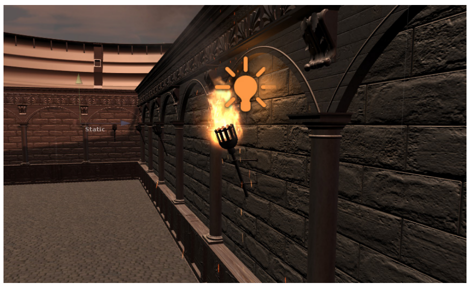


Fig 31. *Rome level fire*

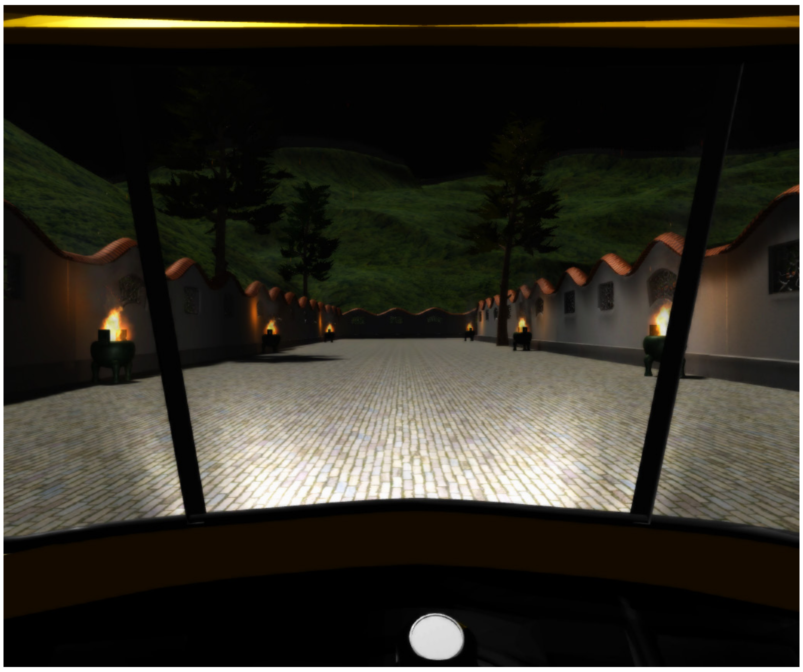


Fig 32. *Road Run*

Menus

Menus: MainMenuScript.cs

Functionality

• The menus are made by the use of several canvases on the same scene. Navigating from one menu to another is made by moving the camera position from one part of the scene to another.
• Sets the master volume
• Sets the resolution
• Sets the tuk tuk model
• Views the cultural story about the monuments from each level
• Quits the game

Considerations

In order to move from one position to another, several empty GameObjects where created with different transform positions. Therefore everytime a button triggers accessing another menu, the camera position swaps between the GameObjects positions.
In order to keep the choose of model, another script was created that stores the value. This script is not destroyed when new scene is loaded.
Navigation can be made also by using the tuk tuk steering (to move up and down) and brake (to Submit)

Output

```
if (optionPress)
        {
            main.transform.position = Vector3.Lerp(main.transform.position, optionsPosition.transform.position, 2f * Time.deltaTime);
            main.transform.rotation = Quaternion.Slerp(main.transform.rotation, optionsPosition.transform.rotation, 2f * Time.deltaTime);
            mainCanvas.SetActive(false);
            storyCanvas.SetActive(false);
            extrasCanvas.SetActive(false);
            //      main.transform.Rotate(0, 0, -(1f * Time.deltaTime));
            //      main.transform.Translate(new Vector3(-0.5f, 0, 0) * Time.deltaTime * 1f);
        }
        if (main.transform.position.x < 59f && main.transform.position.x > 18f)
        {
            optionCanvas.SetActive(true);
        }
```

Fig 33. *Part of MainMenuScript.cs script*

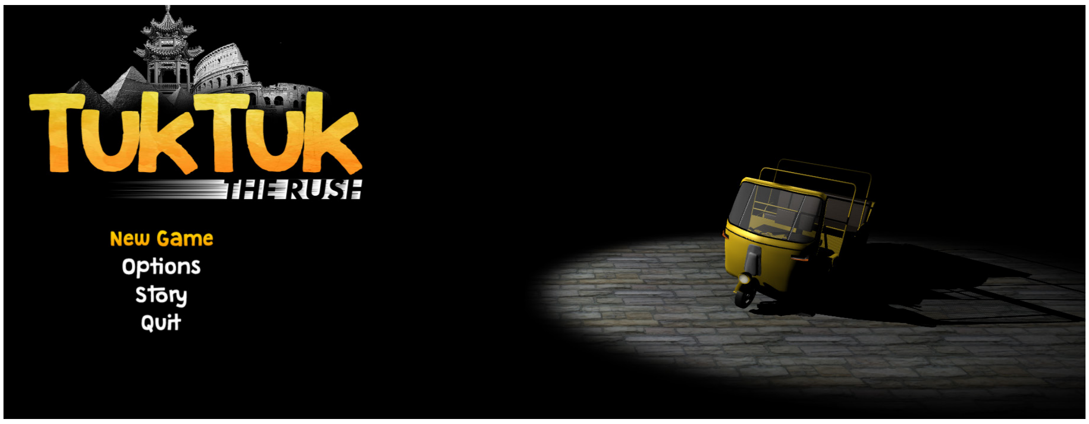


Fig 34. *Main menu*

Scene management

Scene management is made in several scripts. But combined the functionality would be as follow

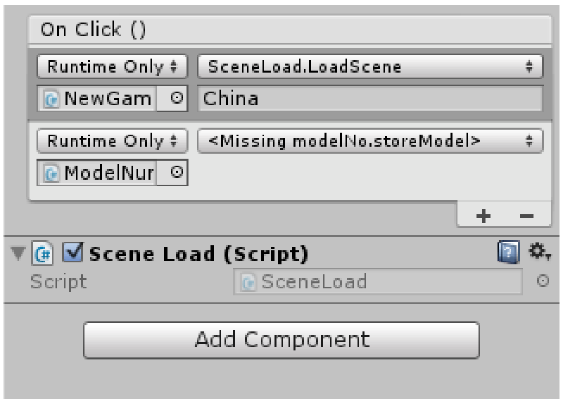
Functionality
· Load scenes from menu/end level triggers
· Handle model swapping

Considerations
For this the considerations have been; how to make the winning condition of the game. Fade screens, camera movement or other effects.
The final selection is to load a new level when we hit a trigger at the end of the level. This will load the next level in the scene by the use of SceneManager.LoadsScene(scene) function.

Output

```
public class SceneLoad : MonoBehaviour
{

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

    public void LoadScene(string scene)
    {
        SceneManager.LoadScene(scene);
    }
}
```

Fig 35. *SceneLoad.cs script*



```
On Click ()
Runtime Only    SceneLoad.LoadScene
NewGam          China
Runtime Only    <Missing modelNo.storeModel>
ModelNur
Scene Load (Script)
Script          SceneLoad
Add Component
```

Fig 36. *Button trigger option*

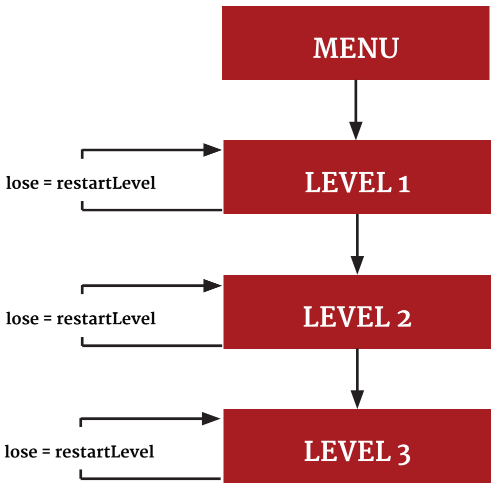


Fig 37. *Scene loading game flow*

5. Testing

Test 1
(Elizabeta Scriba)

One of our collegues tested out the game by using the tuk tuk controls. The output was overall good. The controls were responsive and the game interactive. However, it was laging a little bit and one of the options triggered automatically.

Our conclusion is that sometime, because the error tolerance from the tuk tuk inputs it is high, the values needs to be adjusted and evaluated by using the average operation. A cause of the little bit of lagging might be the delay for transimiting the data sets from one platform to another.

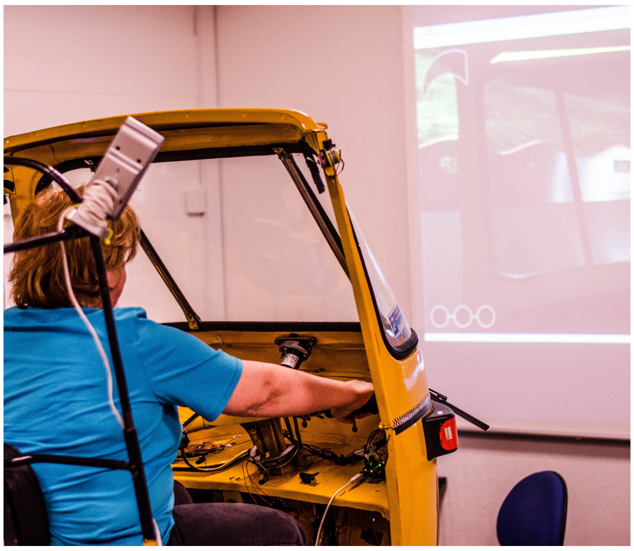


Fig 38. *Test 1 with Elizabeta*

Fig 39. *Test 2 with Dana*

Test 2
(Ana-Daniela Traista)

The second test was performed by an non-student. This time the outcome was a little bit better as the result of fixing attempt from the previous test. However, the gear button switch, because of the lack for grounding, it was oscilating between values 0 and 1023 units. The problem is the current that, the laptopt provided. Therefore, we decided to install a switch button (as safety) that has a grounding pin. After that, the problem was solved, and therefore, we succeded in fixing the major problems.

6. Result

The requirements of the project wast to make the game use Arduino for getting hardware inputs to the game. It was a very challenging part as the group had a lack of experience working with the Arduino. To make a controllable setup together with the TukTuk custom made gears and buttons was made to get the inputs from the devices combine them as a string and then send it to Unity. Moreover as the device is sending the signal as one big string it was separated into smaller parts by a protocol so each individual input from the steering wheel to the buttons are clearly divided so Unity can make a use of it.

One of the requirements of the project was to create menus where the navigation should be done by using the TukTuk controls. It was a great challenge to make it happen as it is assigned to go up or down by pressing a button in the keyboard in Unity, but as we are using a non-conventional method the assigned button had to be "clicked" in script so it triggered the action. This was conqured by triggering events that made a simulation of a button click.

The game features many custom made 3D-models which are specially designed for each scene. One of the problems we experienced during the design part was if the game models are very detailed and there is a large amount of the models in the scene it will affect the runtime of the game by introducing lags to the game. They had to be designed in a way to keep the rich details while keeping the polygon count down.

One of the main features of the game is the escape from the maze. To make it realistic the walls had to be equipped with Box Colliders to stop the player from going through them. Moreover as there is a bomb attached to the TukTuk, so when it explodes the TukTuk flies away. In order to do that a Box collider have been place on the TukTuk and as it collides with other colliders the game automatically pushes the car in an upwards direction giving the effect of blowing up. This was fixed.

In total the game features three scenes: Egypt, China and Rome. By using multiple levels it was hard to pass the information from one scene to another as when the system is changing the level it destroys every object in the previous scenes. To prevent that we created a counter for the selected Tuk Tuk and parsed the value of the counter so we could select the right Tuk Tuk in each scene.

One of the biggest and hardest things was to make the game VR compatible. In our game we use different camera positions for the special abilities. We experienced some bugs when the "Drone" flew up but it didn´t come down again. As well as the UI is not visible on the VR camera. Therefore the game isn't VR compatible in this release.

7. Discussion

At the end of the project we as a group have an overall satisfied feeling about the project. We always strive to do more and better. But time is a limiting factor both in the game and in the real world. If we had more time to develop on the project, we think we could manage to make an even better game.

What would/could we have done differently?
Alex have been spending a lot of time preparing and implementing hardware for the Tuk Tuk. Some time other groups didn't have to use because they could use his implementation. His work power on installing the hardware's could have been used on the game development. We still think this have been a good learning experience for the group to work with the hardware inputs.

Better code sharing
Find a better way to share projects so all can do more work at the same time, without crashing and making the project all messy.

Better work distribution
Having a better plan of who is going to work on what. This will give less time wasted.
We would have liked to implement the finale project with VR(Virtual Reality) but was forced to put it aside due to all our menus not working with the oculus.
Overall we are satisfied with the project and the finale product.

8. Conclusion

Taking a look at the requirements for this project we can conclude we have fulfilled all the must include requirements, all of the should include requirements and one of the could. See the ones included below. Besides the requirements from the project description we have managed to do the following:

Create our own 3D-models
Create our own Materials
Make use of culling (the game engine only renders the watchable units off the camera, increase performance)
Bake light and shadows
Build a full control setup for the Tuk Tuk
Doing an all nighter finishing the game and report

Fulfilled requirements
• The game must make use of the Arduino for input. Standard input (mouse+keyboard or controller/joystick) may be used as a fallback, but the game should be designed with non-conventional input in mind.
• The game must include audio.
• The game must feature menus. Keep in mind that these should be navigable without standard input.
• The game should include imported 3D-models.
• The game should include animation(s) that you have created.
• The game should include RayCasts, Collision events, or other Physics-based events.
• The game could include multiple scenes/levels.

Our conclusion of our final result is: we have a final product we are proud of to present. We have managed to fulfill the necessary requirements and some extra. We have had a learning experience through the whole project, learning new things each day.