Videogame design and development degree

Final degree project's technical report

SolarRacer

Design and development of an educational game for a client

Author: Carlos Hernández Barberà

Tutor: Antonio Morales Escrig

SUMMARY

In this document is exposed the work done in the course Project Exposure (2018) at Saxion University developing an educational game for a client external to the university.

This document is divided in different sections related with the development of the game. Due to the nature of the course the document also talks about the work in group and how different solutions has been developed to satisfy some necessities of the non-programming part of the team during the development time.

Index

1.-Introduction 5
1.1.-Background 7
1.2.-The course: Project Exposure 2018 7
1.3.-The team 8
1.4.-Objective 8
1.5.-Justification 9
1.6.-Initial Planning 9
2.-Game Design 11
2.1.-Game overview 11
2.1.1.-Concept 11
2.1.2.-Game feel 11
2.2.-Gameplay and Mechanics 11
2.2.1.-Gameplay core 11
2.2.2.-Game progression 11
2.2.3.-Challenge structure 12
2.2.4.-Objectives 12
2.2.5.-Actions and controls 12
2.2.6.-Game options 12
2.3.-Interface 13
3.-Art design 14
3.1.-Visual content 14
3.1.1.-References 14
3.1.2.-3D modeling and texturing 14
3.2.-Sound effects and music 14
4.-Functionalities and Technical aspects 15
4.1.- Curve shader 15
4.1.1.-Code details 15
4.1.2.-In-game usage 17
4.2.-Artificial Intelligence 19
4.2.1.-Code details 19
4.3.-Game control 21
4.3.1.-Code details 21
4.4.-Score manager 24

4.4.1.-Code details 25
4.5.-Tile system 26
4.6.-In-game elements (collectables etc) 28
4.6.1.-Code details 28
4.7.-Extra tools 29
4.7.1.-Pre-designed levels tool 29
4.7.2.-In-game debug console 30
5.-Project monitoring 32
5.2.-Feedback 32
5.1.-Planning deviations 33
6.-Results 34
7.-Conclusions 42
8.-Bibliography 43

Table of figures

Figure 1 – Different ways of input allowed 12
Figure 2 – Initial user interface idea 13
Figure 3 – Final user interface design 13
Figure 4 – Asset examples created for the game (done by Claudia Sczygiol and Sebastian Pfeiffer) 14
Figure 5 - Surface shader diagram (Zucconi, 2017) 15
Figure 6 - Vertex modifier in surface shader declaration 16
Figure 7 – Vertex shader declaration 16
Figure 8 – Space conversions in vertex shader 16
Figure 9 – Vertex shader complete 16
Figure 10 - Surface shader 17
Figure 11 – Control loop from the script that controls the curvature of the road 17
Figure 12 – Curve shader result from the side and up-down views 17
Figure 13 – Curve shader Unity's editor interface 18
Figure 14 – Difficulty selection screen 19
Figure 15 – Artificial intelligence main loop 20
Figure 16 – AI's function in charge of check obstacles in road 20
Figure 17 – Touch input management 21
Figure 18 – Input management 22
Figure 19 – Input selector code 22
Figure 20 – Standard touch input management 23
Figure 21 – Game scene before screen touch 23
Figure 22 – Ray cast used to determine which world position is being clicked for the player 24
Figure 23 - Screen side check 24
Figure 24 – Launch event in score method 25
Figure 25 – In red the visual feedback (created for the manager too) of a score event 25
Figure 26 – Road manager in the inspector 26
Figure 27 – Empty road tile 27
Figure 28 – Obstacle tile 27
Figure 29 – Empty and obstacle tiles combined to create a final game tile 28
Figure 30 – Tool interface design 30
Figure 31 – Saving levels 30
Figure 32 – Command class 31
Figure 33 - Main Menu 34
Figure 34 - Number of players selector 34
Figure 35 - Difficulty selector 35
Figure 36 - Starting animation 35
Figure 37 - End of starting animation 36
Figure 38 - Gameplay (Pick-up collision) 36
Figure 39 - Gameplay (Obstacle crash) 37
Figure 40 - Gameplay (Jumping a train) 37
Figure 41 - Gameplay (Final tile and train) 38
Figure 42 - Gameplay (Finishing the race) 38
Figure 43 – Scores 39

Figure 44 – Results 39
Figure 45 - Winner's name selector menu 40
Figure 46 - Scoreboard (after only one race) 40
Figure 47 - Feedback scene 41
Figure 48 - Back to main menu 41
Figure 49 – SolarRacer logo 42

1.-Introduction

1.1.-Background

After three years of coursing the UJI's Videogame Design and Development degree, I decided to do my last year out of Spain to put the learned to test. New ways of working, new professors and new classmates seem the best opportunity to experiment what the life after university could be –and of course, live different cultures and new experiences, something in my opinion fundamental in a creative world like game development.

I decided to do my last term –and in consequence my final project- at Saxion University of Applied Sciences in Enschede, Netherlands. I have been much more interested in the technical part in my past years at UJI and this university had the best option in this direction on the list to choose.

Even having an idea that I would like to develop for this project and knowing that due to the characteristics of the course (see 1.2.- The course: Project Exposure 2018) this idea was not going to be able to be developed I decided to do the exchange term.

1.2.-The course: Project Exposure 2018

Project Exposure is the last course of the last quarter in both gaming related majors (Design and Development) at Saxion University. At this course, the students develop in group a game for an external client following some requirements. In this case, the client has been the *Oyfo Techniekmuseum* at Hengelo(Netherlands) and the requirements have been the followings:

Goal	Increase the awareness of sustainability technology among Oyfo museum visitors • Better insight into strengths and opportunities of sustainable technology • Positive attitude toward the sustainable technology and energy transfer
Method	Interactive experience about strengths and opportunities of sustainable technologies • A serious video game using a touchscreen or an interactive experience for VR
Target	Children, teenagers and adults. • Main language Dutch, optional German / English
Development Teams	From 4 to 9 students. Teams have to been created before the kickoff session.

In addition to these requirements, a list of features and specifications were delivered to us. They are annexed in the *"starting_requirements_annex.pdf"* document for the minimal relevance that they have had in the development of the game.

1.3.-The team

Due to my study contract, I started the course without knowing anyone in that class, so I wrote to the supervisor to find a group. After some days a group of students contacted me looking for a second programmer for their group. With me, the team count with 6 members: three artists, one game designer and two programmers.

The group dynamic has been quite hard. It is important to highlight that the other group members are second-year students and for them is one of the first group project with relative freedom to develop their own idea. This point has made the development a little bit more complicated than my last projects: they still don't have the idea of making a game for a course instead of making their dream game. This created some tensions that have been alive since the first moment.

Due to the characteristics of the team, I decided to adopt a low key profile in the team but a high key profile on the programming team. I would not want to make more difficult the decision making but I wanted to be sure that the programming area was going to work. I tried to have a fluent communication with the rest of the team –specially with the game designer that worked as a kind of bridge between artists and programmers- and give to them all the tools and facilities possible to make them understand what is happening in the game and not convert it for them in a black box. They were not really interested in understanding how the game worked but I thought that this knowledge would make easier understand why some features would need more time to develop or why a certain asset maybe would need to be developed in a certain way to fit the game.

1.4.-Objective

The main objective of this game is to create a serious game to raise awareness about the green energies. The game born from the necessity of transmit information in an accessible way for everyone but especially for kids. Is a game developed for a client so the possibility of creation and innovation are limited as well as the theme or the system where is going to be played.

In addition to this and as a secondary objective, a set of specific tools for that project are going to be created. The tools answer two main necessities that have been found in this project: in the first place the desire of the game designer to be able to remove some of the randomness in the procedural generation of the level in favor of a more pre-designed level. On second place, from the necessity to be able to debug the game in built versions during the different playtesting at the university and at the Oyfo museum. In order to do this two tools have been designed:

- Unity's scene that allows the user to create designed levels making use of the different assets developed for the game.

- In-game console equipped with a set of instructions to enable or disable different debug functionalities.

1.5.-Justification

Two pre-conditions of our project were in the center of our design since the very beginning: the first of this conditions is that the game loop has to be short enough to allow the player to play from start to the end in 5 minutes. The second of the points that we took as one of the main axes of our design was the fact that we were not going to be able to test the touch screen until almost the half of the development time. OYFO museum promised to install a computer with the same characteristics – touch screen included- as soon as soon as possible but they said us that for sure it would not happen before the first playtesting with the museum managers in the 3rd week –from a total amount of 8 weeks of development.

This two conditions guided us to the conclusion that we needed a really simple game based in one mechanic and with a really simple input due to the fact we will not know how precise and reactive would be the screens until we have the game in a relatively advanced phase of development. Knowing also that the potential players are kids and the average play time per game is around 5 minutes we decided to do a game from a known style to make the game easier for the player to understand.

1.6.-Initial Planning

The development of the project in this course is mainly based on iterations. Week after week the professors inspect the game and the game is modified taking into account the feedback given.

Extra to this, some of the weeks have special reviews:

- **Week 3:**
  - Client review
    - In companion with the professors this week some managers from the museum come to the sprint meeting to see the game concept and make suggestions or ask questions.
- **Weeks 6, 7, 8 and 9 (minimum two sessions):**
  - Target public playtesting
    - Playtesting at the museum with kids to have target public feedback.

Even with the fact that a rigid planning is not possible with the iteration system used in the course I tried to structure the course in smooth deadlines taking special attention in the 6th week with the first museum playtesting.

This is the initial planning for the project:

7th May – 13th May	• Kick-off session • Preliminary design
14th May – 20th May	• First gameplay prototype
21st May – 27th May	• Road generator • First GDD and work proposal for Antonio Morales
28th May – 3rd June	• Player control • Pre-designed levels tool
4th June – 10th June	• Obstacles and pick-ups
11th June – 17th June	• Feedback scene and museum playtesting set-up
18th June – 24th June	• Power ups • In-game debug console
25th June – 1st July	• Sound and post-processing
2nd July – 8th July	• Post-mortem presentation at Oyfo museum • Final memory

2.-Game Design

In this section, the Game Design Document of the game is going to be explained. Due to the characteristics of the game and the development process, there is no section dedicated to the game narrative and the art design part is going to be a little bit schematic because I haven't had any impact on it –beyond giving my opinion.

2.1.-Game overview

2.1.1.-Concept

SolarRacer is an educative game scoped around the idea of making attractive for kids the use of renewable energies. In order to achieve this, a famous game pattern has been used: an infinite runner game. In SolarRacer the player has to win a race driving a solar car avoiding obstacles and cloud shadows to increase the solar energy supplied to the car and get a speed boost.

2.1.2.-Game feel

The points on which the game feel orbits are visual clarity and easy control. All the game tries to be as simple as possible but keeping the learning goal in mind. The use of flat colors and saturated colors gives to the game a cartoon style trying to increase the friendly sensation that the game tries to achieve.

2.2.-Gameplay and Mechanics

2.2.1.-Gameplay core

The player has to change between different lanes in a road to, avoiding obstacles, win the race. The speed of the car is not directly controlled for the player and the road is generated procedurally in each game loop so road memorizing is not possible. SolarRace appeals to the player's reflex over strategy or muscular memory following the idea of being a game that is going to be played only one or two times per player.

2.2.2.-Game progression

Due to the nature of the game, the only game progression in the game is inside of each game loop. Is not sure that a player is going to play more than one game so implementing a long-term progression has no sense.

On the other hand, reward good players in each race is more than needed. This is achieved giving a velocity boost for players for each second they stay under the sun charging the solar panel. This creates different levels of difficulty: if a player cannot avoid the obstacles the car is going to be slower giving an easier experience but if the player manages to avoid all obstacles the road is going to increase the velocity offering a more challenging game. The final layer of difficulty is achieved with the different collectibles and pick-ups to obtain a higher punctuation.

2.2.3.-Challenge structure

The unique challenge that the game presents to the player is to avoid the obstacles on the road. This challenge is short and highly isolated from the others due to the different obstacles does not interact between them.

The road generation (see *4.5.-Tile system*) ensures us a non-totally random obstacles structure which ensures that challenges are going to be controlled and adjustable.

2.2.4.-Objectives

Two layers of obstacles are included in the game. The first of them and the principal one is to try to reach the goal before your rival (that can be AI controlled or controlled for a second player). The second of the objectives is try, at the same time that you try to win, to gain as many points as possible to enter in the leaderboards.

2.2.5.-Actions and controls

Talking first about game control is important to make clear that only touch screen input has been allowed for the game. The player has no the option to use a mouse, keyboard or other input devices apart from the screen.

This lead to simplify as much as possible the control but at the same time to allow different kind of inputs to make it as accessible as possible (see *4.3.-Game control*). With this in mind, the only action that players can do is to change the lane on the road but this can be achieved in three different ways: tapping at the left or the right of the vehicle, dragging in the direction that they want to move or using the movement buttons on the screen.

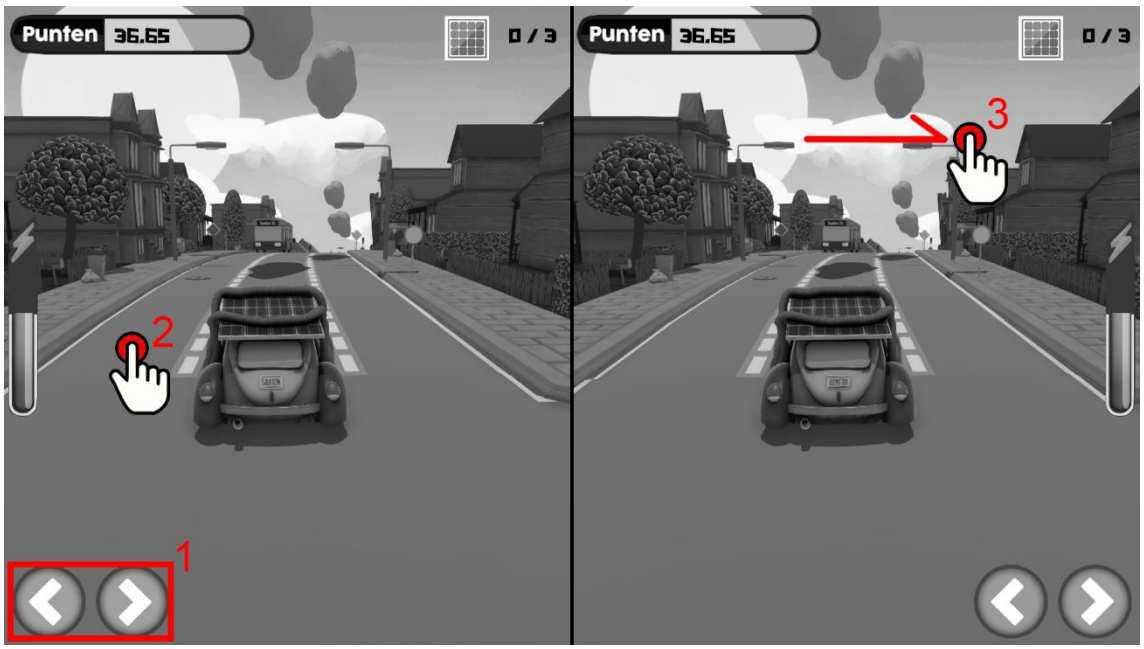

Figure 1 – Different ways of input allowed

2.2.6.-Game options

The player has gameplay options to choose like playing versus another human player or against an artificial intelligence and in this latter case the difficulty of it.

However, graphics options are not allowed in order to ensure the correct working of the game in the museum's computers and sound options are going to be controlled by the client thought the speaker's intensity. Getting away this options from the player is a way to ensure that game is going to work as the client expects.

2.3.-Interface

The design of the interface has meant a challenge. Having a split screen obliged us to duplicate the user interface. However, due to the players are going to play in a relatively small space and side by side does not allow to just apply the same interface in both screens.

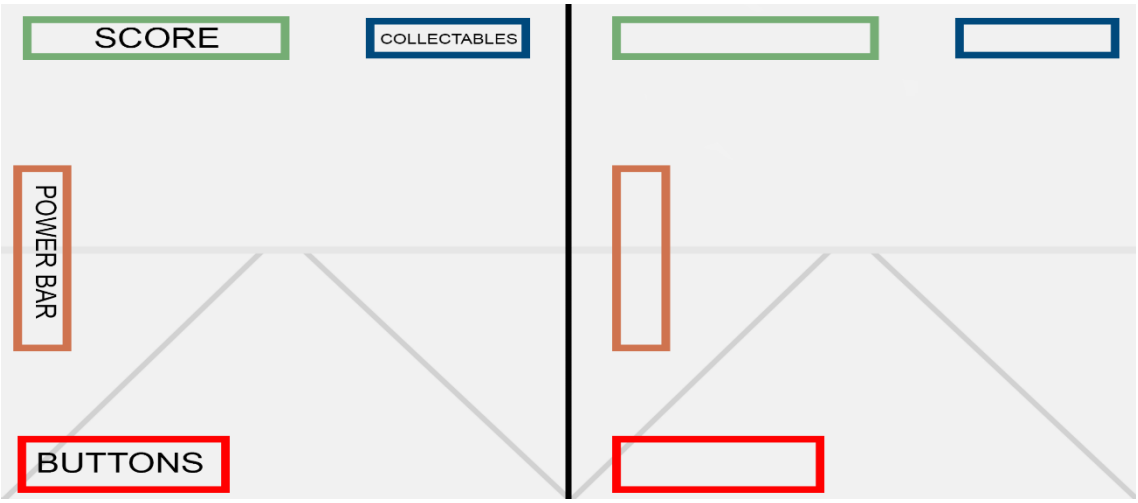

Figure 2 – Initial user interface idea

Apply this interface creates the problem that players obstructed each other when playing due to the proximity between them while they were playing. The solution was reflecting the power bar and the buttons. This solution gives more space for the players meanwhile they are playing. The score and collectibles indications remained in the same position because after some playtesting with them reflected too we realize that for the player two was confusing having the collectibles count before the collectibles.

Figure 3 – Final user interface design

3.-Art design

3.1.-Visual content

3.1.1.-References

Following the kid-oriented character of our game, we decide to use a cartoon style for our game with flat colors and simple geometries giving priority to representation over realism. To achieve this, the reference pool has been focused on television cartoons like Steven Universe or Gravity Falls and video games as RIME or Fornite.

Figure 4 – Asset examples created for the game (done by Claudia Sczygiol and Sebastian Pfeiffer)

3.1.2.-3D modeling and texturing

All the models and textures have been done by Sebastian Pfeiffer, Claudia Sczygiol and Joran Vergoessen using Autodesk Maya and Photoshop. Sebastian and Joran have worked mainly in modeling meanwhile most of the texturing work has been done for Claudia.

The models were created trying to economize vertices trying to minimize as much as possible the impact of loading the models and the screen display in the game.

3.2.-Sound effects and music

Joey Heinze has been the person responsible for the production of the different sounds in the game. Talking about the music the game does not have any kind of background music or similar. The museum managers informed us that the sound level of the speakers in the museum is really low so the possibility of adding music was abandoned to give all the sound attention to the effects. Mixing music and sound effects in speakers with a low volume could cause that music eclipses the effects reducing the information about game events that player receive from the effects – like for example the sound that warns the player that a train is going to pass.

4.-Functionalities and Technical aspects

In this section, the different features developed for the game are going to be showed and explained. Is important underline that all the features have been developed exclusively by me and the other programmer has done only – if needed – small modifications to adjust the different scripts to features that he was developing like, for example, add getters or similar changes.

A final note is important to underline that most of the codes showed here are cut back to increase the legibility. To see the codes without any modification or cut go to the repository with the project (see *6.-Results*)

4.1.- Curve shader

One of the main features implemented for the game is a shader used for hiding the world generation and give more dynamism to the road thanks to a curvature in the road.

4.1.1.-Code details

The Unity's native illumination model works over the output of the surface shader in the rendering pipeline. If a vertex or fragment shader is implemented but the code does not contain a surface shader a custom illumination model is needed to be implemented.

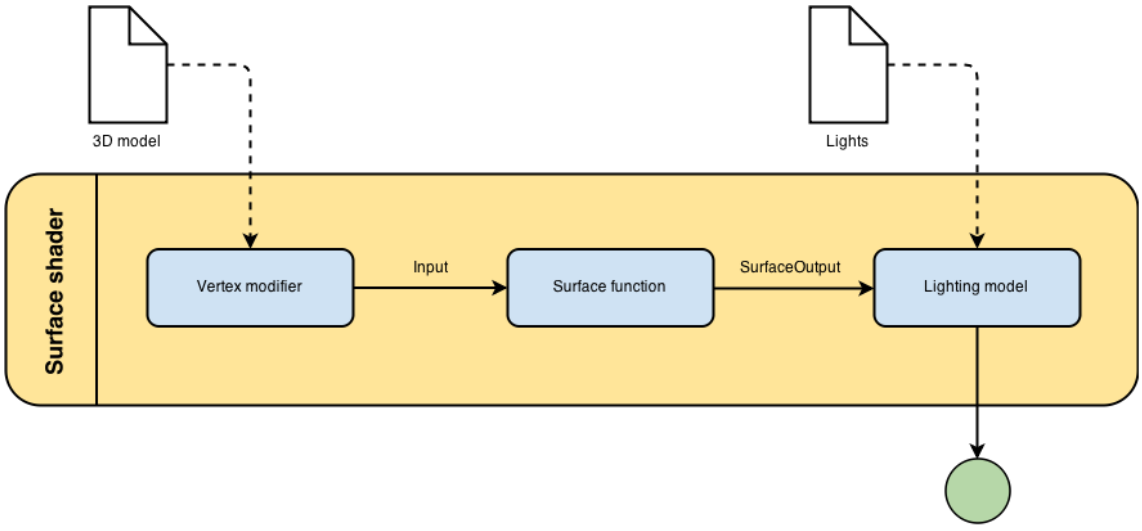

Figure 5 - Surface shader diagram (Zucconi, 2017)

This situation creates a dilemma when a vertex shader has to be implemented in a project: implement a custom illumination model (Phong or Gouraud for example) in a custom fragment shader or try to combine the vertex shader with a surface shader in order to keep the BRDF (bidirectional reflectance distribution function) implemented in Unity.

Seems clear that using the integrated system is going to be, probably, much more efficient and, in the first moment, easier than implement our custom illumination model.

It is possible to use a custom vertex shader to modify the vertex data that is going to be sent to the surface shader. Using the compilation directive vertex:name_of_our_vertex_shader we can have our custom vertex behavior.

```
// Physically based Standard lighting model, and enable shadows on all light
types
#pragma surface surf Standard fullforwardshadows vertex:vert addshadow
```

Figure 6 - Vertex modifier in surface shader declaration

Is important take into account that our vertex shader has to have at least an "*inout appdata_full*" parameter (only declaration lanes included):

```
void vert(inout appdata_full v) {


}
```

Figure 7 – Vertex shader declaration

The second of the points to take into account is the curvature code. The desired behavior is to have a curvature in the desired axis depending on the distance from a vertex to the camera. To achieve this is needed to change the vertex space from object space to world space due to if not we are not going to have a continuous curve if different objects are disposed in a lane but, the same curvature repeated in each one of the objects. After doing the needed transformation returning the vertex to the object space is needed to be able to process it in the surface shader.

```
void vert(inout appdata_full v) {
	float4 vPos = mul(unity_ObjectToWorld, v.vertex);
	//apply desired transformations
	v.vertex = mul(unity_WorldToObject, vPos);
}
```

Figure 8 – Space conversions in vertex shader

The objective of the transformation over the vertices is to achieve a quadratic curve and adjust the vertices to that curve. For achieving that the distance of the vertex to the camera is divided by a distance value that allows adjusting the distance where the curve is going to start to have effect over the vertices positions. The curve is modified with a constant value defined in the Unity's editor called *_QOffset* of type *Vector3* to be able to curve objects in the three coordinate axes.

```
void vert(inout appdata_full v) {
	float4 vPos = mul(unity_ObjectToWorld, v.vertex);

	float offset = vPos.x / _Dist;
	vPos += _QOffset* offset * offset;

	v.vertex = mul(unity_WorldToObject, vPos);
	v.texcoord = v.texcoord;
}
```

Figure 9 – Vertex shader complete

The last point is to do the surface shader. If the variables declaration has been done correctly this shader is quite trivial and assign the different parameters to the appropriate output parameter variable is enough.

```
void surf (Input IN, inout SurfaceOutputStandard o) {
	// Albedo comes from a texture tinted by color
	fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Tint;
	o.Albedo = c.rgb;
	o.Emission = tex2D(_EmissionMap, IN.uv_MainTex) * _Emission;
	// Metallic and smoothness come from slider variables
	o.Metallic = _Metallic;
	o.Smoothness = _Glossiness;
	o.Alpha = c.a;

	o.Normal = UnpackNormal(tex2D(_NormalMap, IN.uv_MainTex));
	//eveything in uv_Maintext because i know that they are going to have the
	shame texture coords
}
```

Figure 10 - Surface shader

4.1.2.-In-game usage

The shader in-game acts over the vertices of the models adjusting them to a quadratic curve independent in any of the axes with the displacement desired. The curve is controlled with a script that changes the curve parameters to adjust it to a new curvature.

```
foreach (Material mat in _materials)
{
	mat.SetVector("_QOffset", new Vector4(0, displacement.y, displacement.x));
}
UpdateDisplacement();
UpdateLerpTimes(Time.deltaTime);
```

Figure 11 – Control loop from the script that controls the curvature of the road

This is the main loop done for update our curvature. *UpdateDisplacements( )* just lerp the *Vector2* variable displacement from the current value to a target value in a certain time. Is important to say that for the game set up the script is giving values only to two of the curvature values keeping the third one straight but if needed we can have a curvature over the three axes.

Figure 12 – Curve shader result from the side and up-down views

Another interesting point about the shader is the fact that it only works over the visual elements of the objects but not over position, colliders or other elements attached to the object. This is really important in order to keep easy the game related mathematics. For our game logic, everything is in a straight lane so, for example, if we need to check if something is on the road we can just make a normal raycast in the desired direction.

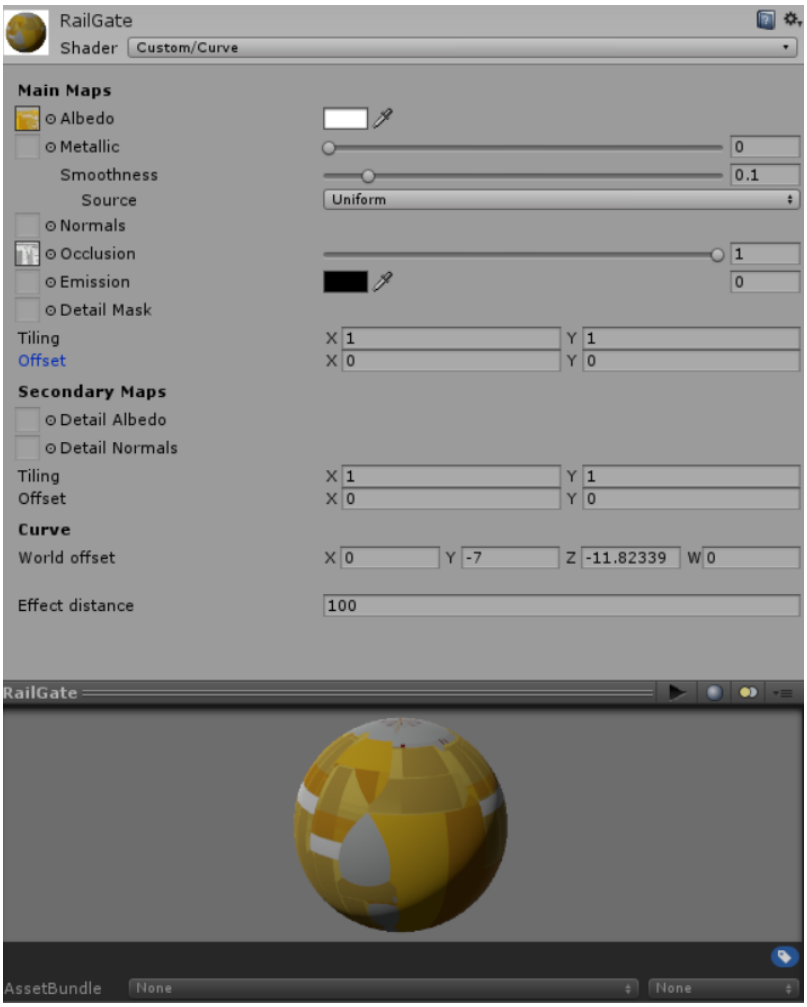

Figure 13 – Curve shader Unity's editor interface

To conclude this explanation and following the design idea of making everything as accessible as possible to the rest of the team (see *1.3.-The team*) a Unity's Editor interface has been developed. With this, designers and game artist can have a familiar interface to be able to manipulate a material with the curve shader applied. Not all the parameters have a linked variable in the shader some of them are just mock parameters in order to make the interface as similar as possible to the standard material of Unity

4.2.-Artificial Intelligence

Trying to keep the race concept in the game even if only one player was playing leads to the necessity of having an Artificial Intelligence(AI) to control the second player if needed.

The client required age-segmented difficulties so a difficulty selector was implemented in the game in case of having a solo player.

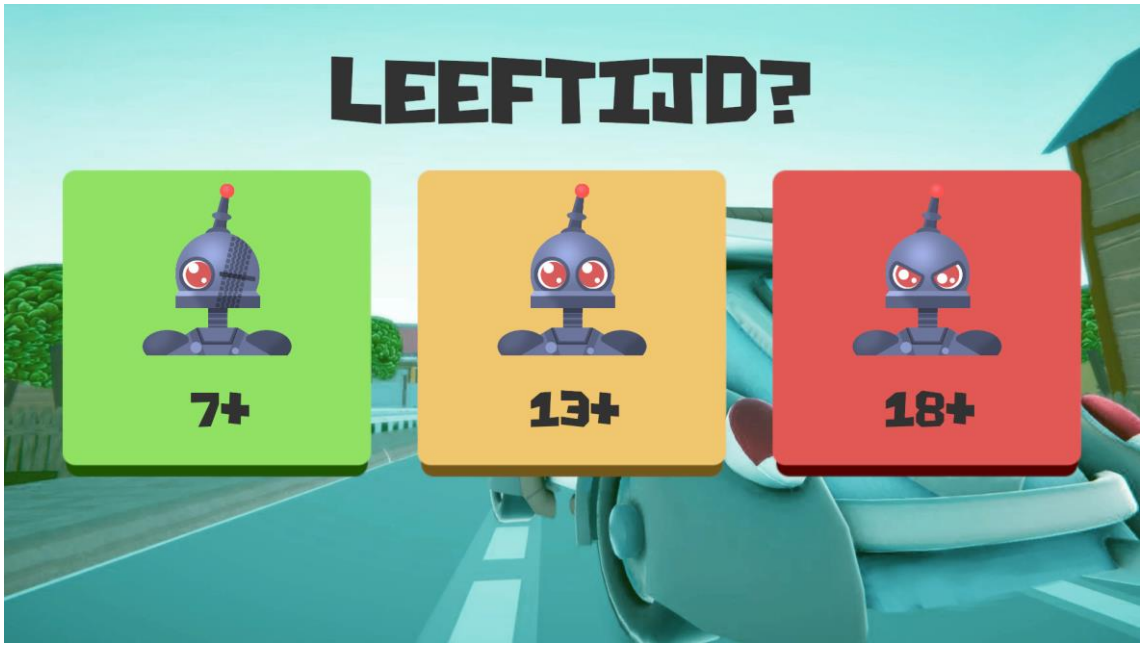

Figure 14 – Difficulty selection screen

The different difficulties change the cool down values and the check distances but not only. The abilities of the AI change between difficulties levels. The easier one just checks in front of the vehicle to decide to which way move but, in harder difficulties also check if they have an obstacle in the position they are going to move to check if is a recommendable position. Last but not least the hardest difficulty also looks for ramps to avoid the trains that cross the road once in a while.

4.2.1.-Code details

The main logic loop of the AI composed in three actions: check for obstacles, decide the best road to go and finally move to the closest road to the target one.

```
void Update()
{
    if (_enabled)
    {
        CheckObstacles();
        _timerLastReaction -= Time.deltaTime;
        _timerLastMovement -= Time.deltaTime;
        if (_timerLastReaction <= 0f)
        {
            Decide();
            _timerLastReaction = _reactionTime;
        }
        if (_timerLastMovement <= 0f)
        {
            MoveToTargetLane();
        }
    }
}
```

Figure 15 – Artificial intelligence main loop

Once again AI in video games needs to be adjusted to be fair for the players instead of perfect in the resolutions. A set of timers for the different actions was added to achieve this objective. Via this is possible to make the AI works clumsy giving to the player some chances to win the race.

Even the three functions inside the loop are quite unequivocal I'm going to do a little stop in this function:

```
private void CheckObstacles()
{
    RaycastHit hit;
    for (int i = 0; i < _lanesToCheck.Length; i++)
    {
        bool hitFlag = Physics.Raycast(_lanesToCheck[i].position, Vector3.left,
out hit, _checkDistance, 1 << 13);

        if (hitFlag == true)
            _distancesToObstacles[i] = hit.distance;
        else
            _distancesToObstacles[i] = 100000000f;
    }
}
```

Figure 16 – AI's function in charge of check obstacles in road

As it was explained in the previous point (see *4.1.- Curve shader*) only standard ray-casts are needed in order to check if the road contains an obstacle in a lane at a certain distance. The hit distance is stored and, in most of the cases, in the following action (decide which lane is the one we have to move) the lane with the bigger distance to an obstacle is going to be selected as target lane.

4.3.-Game control

The game control of SolarRacer presents some peculiarities induced as much for the final system where is going to be played the game as for the fact that the main potential objective players are kids.

4.3.1.-Code details

Screen input in touch screens is managed in Unity in a different way than the mouse clicks. Is possible to do the input management interpreting the touches in the screen as mouse inputs but, in that case, multi-touching would not be supported. The local multiplayer nature of the game makes impossible this input management because the input of one of the player could block the input of the other one.

```
for (int i = 0; i < Input.touchCount; ++i)
{
       Touch t = Input.GetTouch(i);
       if (t.phase == TouchPhase.Began)
       {
            //Input.GetMouseButtonDown equivalent
       }

       if (t.phase == TouchPhase.Ended)
       {
            //Input.GetMouseButtonUp equivalent
       }
}
```

Figure 17 – Touch input management

Looping through the inputs solves this problem and from the moment where an input has been selected the management is mainly as the standard mouse input.

The different playtesting sessions revealed to us that the first idea about how to catch the input from players was not going to work. Some of the kids that assisted to the first playtesting session tried to move the car touching at the right or left side of the car instead of using the buttons at the bottom of the screen.

At that moment different ways of input were included in the game trying to adapt the game to any kind of player. This was marked as a priority in the development trying to reduce as much as possible the learning curve in the game; in a game with a game loop of not much more than a minute the player needs to be able to enter in the game as fast as possible.

Drag input was also included in the game. Due to this kind of input was implemented before being able to try the touch screens the drag input is working with a personal implementation instead of using the *Touch* phase called *Moved*.

The way to combine these three input options is the following: buttons input is isolated and they work with a standard Unity's UI *OnClick* event. The other two options are closely linked but to have a way to distinguish when one or the other is being used is really important. For doing that the movement order is controlled in two times: when the screen detects a new touch in the linked screen side that position is stored. After, when that touch is released the displacement vector is calculated and depending on the

displacement amount the input is processed as a simple touch input or as a drag movement on the screen.

```
for (int i = 0; i < Input.touchCount; ++i)
{
	Touch t = Input.GetTouch(i);
	if (t.phase == TouchPhase.Began)
	{
		_clickPosition = new Vector3(t.position.x, t.position.y,0);
	}

	if (t.phase == TouchPhase.Ended)
	{
		OnClickUp(new Vector3(t.position.x,t.position.y,0));
	}
}
```

Figure 18 – Input management

```
private void OnClickUp(Vector3 _upPosition)
{
    Vector3 _diff = _upPosition - _clickPosition;
    if (Math.Abs(_diff.x) > Screen.width / 2 * _minimumXDisplacementForDrag)
    {
        if (_diff.x > 0)
            OnClickRight();
        else if (_diff.x < 0)
            OnClickLeft();
    }
    else
        ClickOnScreen(_upPosition);
}
```

Figure 19 – Input selector code

In the case that the displacement is bigger than a defined value the only operation to do is check the direction of the displacement and move the vehicle accordingly. However, in the case of a standard touch, we still have to find out the world position. After checking that the touch is not in the space of one of the movement buttons the function *Camera.ScreenPointToRay(Vector3 pos)* is used to determine if the ray obtained from the screen touch collides with the road at the right or the left of the vehicle.

```
private void ClickOnScreen(Vector3 mousePosition)
{
    if (_controlledByAI == false)
    {
       RectTransform r1 = _movementButtons[0].GetComponent<RectTransform>();
       RectTransform r2 = _movementButtons[1].GetComponent<RectTransform>();
       bool res1 = RectTransformUtility.RectangleContainsScreenPoint(r1, new
       Vector2(mousePosition.x, mousePosition.y), null);
       bool res2 = RectTransformUtility.RectangleContainsScreenPoint(r2, new
       Vector2(mousePosition.x, mousePosition.y), null);

        if (res1 == false && res2 == false)
        {
            Ray ray = _camera.ScreenPointToRay(mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit, 200f)
            {
                if (hit.point.z > transform.position.z) //go right
                {
                    OnClickRight();
                }
                else if (hit.point.z < transform.position.z)
                {
                    OnClickLeft();
                }
            }
        }
    }
}
```

Figure 20 – Standard touch input management

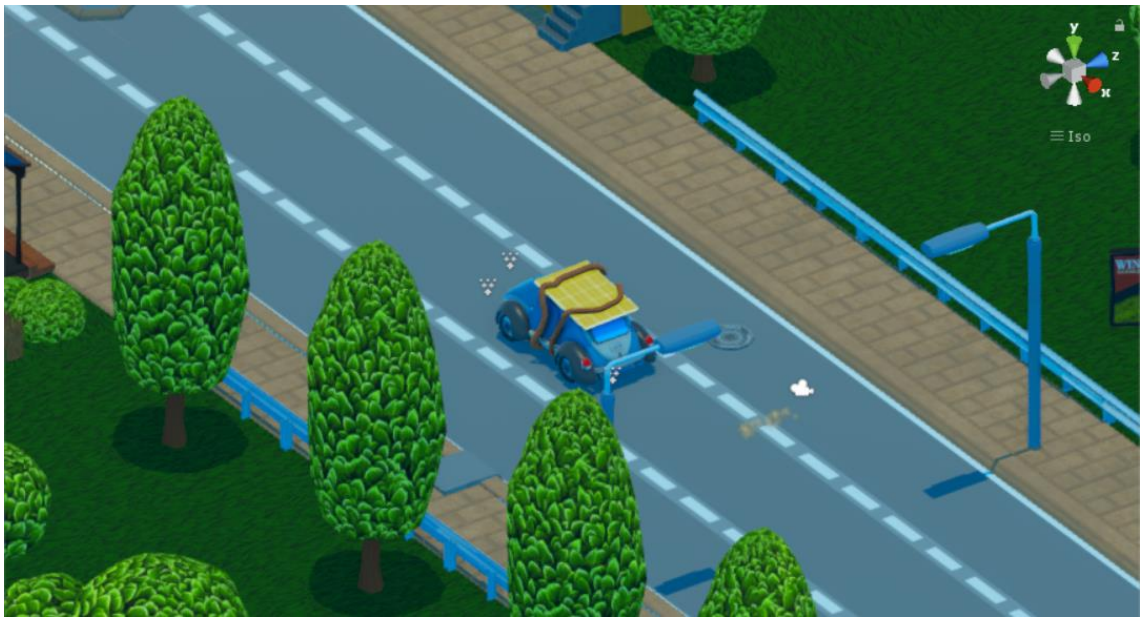

Figure 21 – Game scene before screen touch

Figure 22 – Ray cast used to determine which world position is being clicked for the player

The last but not least of the points to take into account is being able to distinguish with which player a certain input is related. The game screen is spitted into two equal parts and the vehicles have to act only if a certain input is in their screen side.

```
private bool IsThisPointInMyHalfScreen(Vector3 _point)
{
    _inputTimer.ResetTimer();
     if (_point.x < (_camera.rect.x + _camera.rect.width) * Screen.width
         && _point.x > _camera.rect.x * Screen.width)
        return true;
    return false;
}
```

Figure 23 - Screen side check

To do this selection process a touch input is only accepted if it is inside the camera rectangle assigned to the script. The mathematics linked to the check are quite straightforward once known that the width of the rectangle is a value between 0 and 1 and is relative to the screen width.

4.4.-Score manager

Due to the necessity of having a score system easily tweakable for the game designer, a ScoreManager was developed. This system is in charge of manage everything related to the score of both players.

The system is able to increase or decrease the score of a player and also, it can launch score events within HUD visual feedback. Thanks to this system the designer has the possibility to change point values and the other programmer can work with the score in a masked way avoiding to be worry about how it works.

4.4.1.-Code details

This section is going to be dedicated to one of the two public methods in the script. This method is in charge to handle with the score events caused, for example, for the collision with a pick-up. The point to highlight in this method is the call to the function *GetEvent(player, roadPoints).*

```
public void LaunchEventInScore(float points, int player, bool roadPoints)
{
  if (_roadPlayer1.FinishReached == false && _roadPlayer2.FinishReached == false)
  {
      GameObject tmpGO;
      tmpGO = GetEvent(player,roadPoints);

      AddPointsToPlayer(points, player);

       if (points >= 0)
       {
           //green text appear
           tmpGO.GetComponent<Text>().color = Color.green;
           tmpGO.GetComponent<Text>().text = "+" + points;
       }
       else
       {
           //red text appear
           tmpGO.GetComponent<Text>().color = Color.red;
           tmpGO.GetComponent<Text>().text = points.ToString();
        }
    }
```

Figure 24 – Launch event in score method

The distinctive feature of this function is that this script stores a queue of score events instead of creating and destroying them each time. Taking into account that it could happen that various events occur at the same time create a lot of objects at the same time in a short period of time could create a low-performance peak. Instead of that, the cited method checks if the pertinent queue is empty and only in that case a new event is created – creating the visual feedback and adding the score to the corresponding player. When a score event finish it is disabled, enqueued and stays disabled until is required.

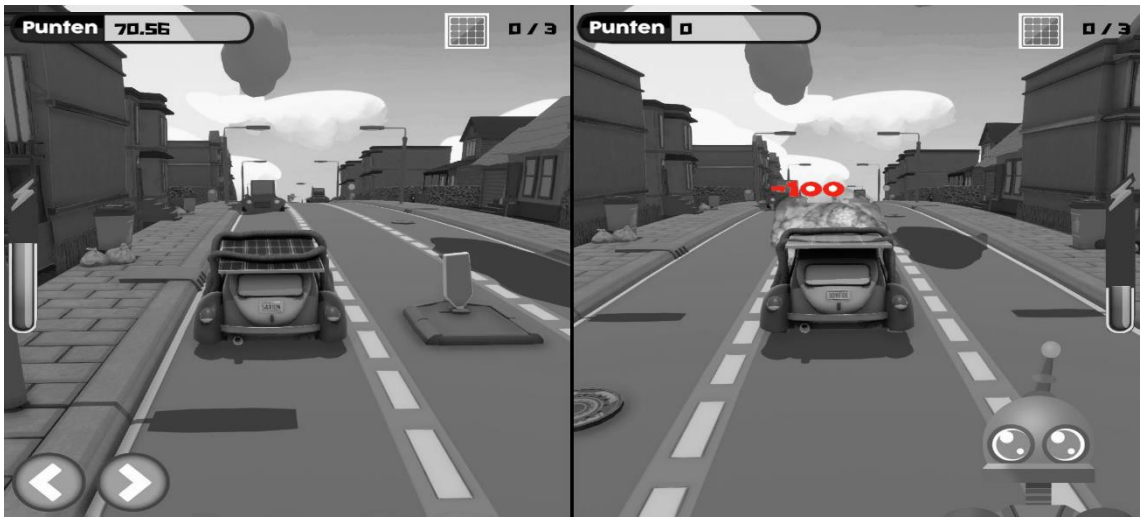

Figure 25 – In red the visual feedback (created for the manager too) of a score event

4.5.-Tile system

Procedural systems gives the flexibility of having a different level each run. However, this kind of system can create unfair, frustrating or directly impossible challenges for the player. In order to solve this the road generation is not totally random generated. Instead of it, the road is generated making use of a group of tiles designed for the game designer of the team.

In the Unity's scene the roads management is divided between two scripts: *RoadManager* and *Road*. The first of this scrips acts as a kind of server where the different tiles, collectables or other elements are stored and the second one that is in charge of the management each one of the roads.

Figure 26 – Road manager in the inspector

The system works with different kind of tiles due to the desire of the designer of having as much control as possible. Is possible to say that there are two big groups –the division is a little bit more complex but for educational purposes is correct- of tiles: road tiles and obstacles tiles. This two kind of tiles are combined, in code, in a random way to create the final tiles that the player is going to find in the road.

Figure 27 – Empty road tile

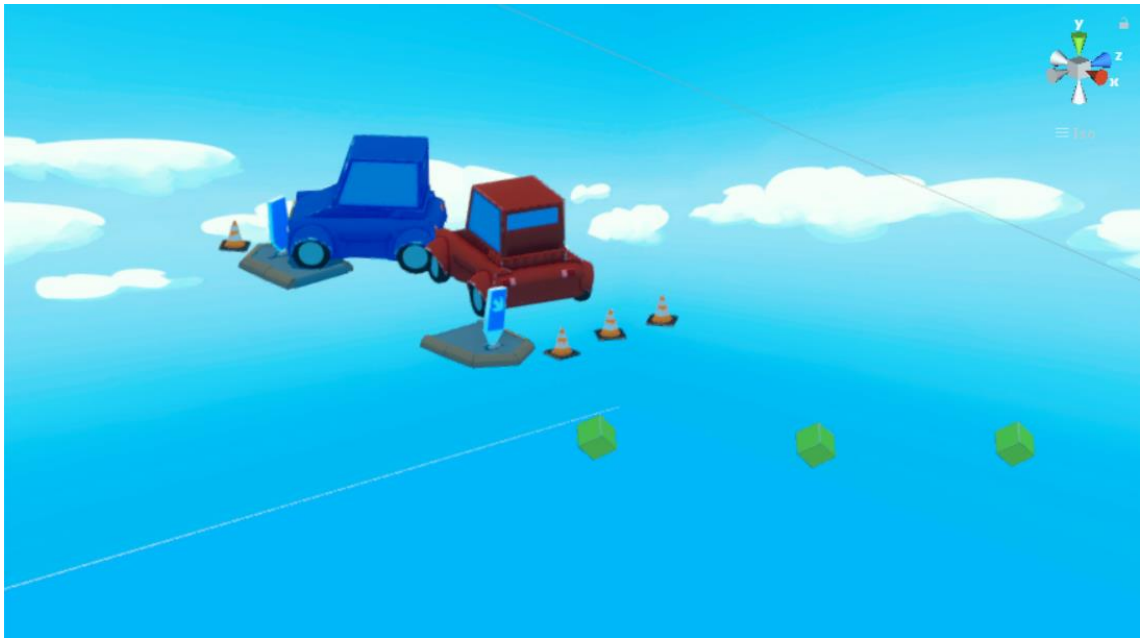

Figure 28 – Obstacle tile

Figure 29 – Empty and obstacle tiles combined to create a final game tile

This tile system is the starting point for one of the tools developed for the game: the pre-designed levels (see *4.7.1.-Pre-designed levels tool*) and allows the adjustment of the game in a really easy way without any change in the code what is perfect to allow non-programmers team members to get more involved in the implementation in Unity of the game.

4.6.-In-game elements (collectables etc)

The behavior of these elements is quite simple but due to his fundamental role in the game, they are in this document. There is two kind of elements that the player can collect in the road.

The first of them are pickups that give to the player a small number of points for each one he can collect. The second one is solar panels distributed along the road; for each group of 3 collected the player receives a relatively big amount of points.

4.6.1.-Code details

The interesting point is that the collectables are not linked to the tile system (see *4.5.-Tile system*). The road manager has a counter that indicates when a collectible has to be spawned. At that moment the function *LookForFreePosiitionInTile* is called.

```
private Vector3 LookForFreePositionInTile(Tile tile)
{
    Transform[] lanes = _player.GetComponent<VehicleManager>().Lanes;
    Vector3 candidatePosition = new Vector3(0, 0, 0);

    int[] laneOrder = RandomLaneOrder();

    foreach (int lane in laneOrder)
    {
        candidatePosition = new Vector3(tile.GO.transform.position.x, 1f,
lanes[lane].position.z); //temporal

        if (tile.GroundObstacles != null)
        {
              BoxCollider collider =
              tile.GroundObstacles.GetComponentInChildren<BoxCollider>();

              if (collider != null && collider.bounds.Contains(candidatePosition)
              == false) //lanzar rayos para asegurar minima distanacia
              {
                     RaycastHit hit, hit2;
                     bool negativeXAxis = Physics.Raycast(candidatePosition,
                     Vector3.left, out hit, 5f);
                     bool positiveXAxis = Physics.Raycast(candidatePosition,
                     Vector3.right, out hit2, 5f);
                     if (negativeXAxis == false && positiveXAxis == false)
                           return candidatePosition;
            }
        }
    }
    return candidatePosition;
}
```

This function looks for a free lane in a tile. For doing this the three lanes are iterated randomly and the function checks if a possible spawn position is not inside of an object – checking if the position is contained or not for a collider - and if the position is far enough from any other object – using raycasting. If both two conditions are favorable that position is chosen as spawn position; if not a new position needs to be found.

4.7.-Extra tools

In addition to the game features a set of tools have been developed to solve some requirements of the team.

4.7.1.-Pre-designed levels tool

The game designer wanted to have the possibility to create levels selecting one by one the different tiles and obstacles attached to each tile. For solve that requirement a level creator tool was developed.

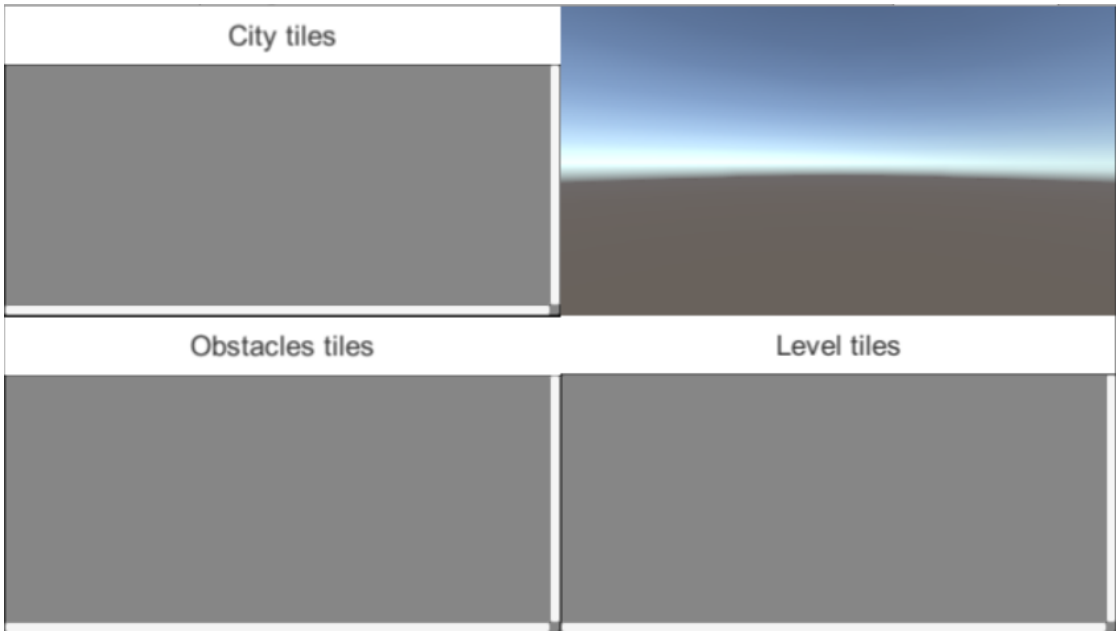

Figure 30 – Tool interface design

In this scene is possible create new levels, delete or edit created ones and select the level to be loaded in the game.

The different levels are created and stored in a .txt file.

```
private void SaveLevel()
{
	//...
	foreach (int[] tileMix in tiles)
	{
		file.WriteLine(tileMix[0] + "," + tileMix[1]);
	}
	//...
}
```

Figure 31 – Saving levels

4.7.2.-In-game debug console

In order to allow to modify some parameters in the built versions at the different playtesting sessions an in-game console was developed. This console allows to change some values in the game – as check distances or others – and enable or disable debug functions like, for example, ray visualization.

```
class Command : IEquatable<String>
{
    List<string> names;
    System.Type type;

    public Command(System.Type t, List<string> allowedCommands = null)
    {
        if (allowedCommands != null)
            names = allowedCommands;
        else
            names = new List<string>();
        type = t;
    }

    public bool Equals(String com)
    {
        return names.Contains(com);
    }
}
```

Figure 32 – Command class

To make it as general as possible a *Command* class has been created where different ways to call the same instruction and the related type of the variable linked is stored. Is possible to add or remove new keywords for an instruction from the console.

5.-Project monitoring

In the first of the points of this document (see 1.6.-Initial Planning) an initial assumption about the possible scope of the project has been made. In the development of the project different feedback, new requirements and problems showed up and in consequence, the initial planning needed to change.

Talking more in detail about the different factors that lead the project to change there's two of them that is needed to highlight: the first one is the team. So many dominant voices in the team provoked a delay that started in the first two weeks of the project. The first idea didn't like to all the team so a whole change was made. The game changed from a puzzle game to an infinite runner. This little bumps between team members have been continuous during most of the project and, for example, do something to change it later has been a constant practice.

Continuing talking about the team, another of the points that have slowed the development has been an unbalanced team composition and the weekly work loop. The team has been composed of six persons and four of them artists. This situation creates a bottleneck each week where a lot of art and features needed to be implemented in Unity at the end of the week. The workflow is normal in all the developments; the art department used to work more at the beginning of each sprint meanwhile the programming team used to have more workload at the end but, due to the unbalance in the team this dynamic was aggravated.

The second of the elements that slowed the project has been, as expected, the feedback received from the client, public and professors. Different features (the curve shader or the AI for example) and extensions of planned features were ordered. This, of course, delayed or forced us to cancel planned features.

Even with these, in part expected, problems and delays the delivery deadlines have been successfully completed and all the client requirements have been developed.

5.2.-Feedback

An important side note is that between the end of the second and the beginning of the third week the game concept was totally changed. The change has the approbation of the professors.

7th May – 13th May	• Puzzles can be little bit boring for kids if do not have a good design
14th May – 20th May	• The puzzle seems to be well resolved but again, is going to be hard to make it really interesting in the context the project has to be developed. • **NEW GAME**: Fits more with the idea that client looks for but the learning goal is much more diffuse. The local multiplayer is something that any other group is going to develop and is really interesting.
21st May – 27th May	• **Client:** Having a solo mode is required because we cannot ensure that all the visitants are going to be accompanied or with someone to play with

	• **Professors:** Cool concept. Needed to find a way to hide the world generation.
28th May – 3rd June	• There's a little bit of delay in the development
4th June – 10th June	• The delay and the problems seem to not been in the way to get solved. Really important to have the game much more advanced for next week.
11th June – 17th June	• **Play testing:** Needed to add more ways to control the vehicle. • In general the game has been well received for the kids and the work ordered past week has been done.
18th June – 24th June	• **Play testing:** Funny but simple. The game needs more things to do (vertical movement, collectables…).
25th June – 1st July	• Sound is needed and more visual feedback to the player. • The final presentation has been really poor. • The game is really interesting but the workflow haven't been the best one
2nd July – 8th July	• **Public:** People liked it. Has been the most played game but not the one with higher grade.

5.1.-Planning deviations

The final work calendar has been the following:

7th May – 13th May	• Kick-off session • Preliminary design
14th May – 20th May	• First gameplay prototype • Game change • Prototype new game
21st May – 27th May	• Curve shader • First GDD and work proposal for Antonio Morales
28th May – 3rd June	• Curve shader
4th June – 10th June	• Artificial Intelligence • Point system
11th June – 17th June	• New input ways
18th June – 24th June	• Collectibles and pick-ups • Ramps and trains
25th June – 1st July	• Visual feedback particles • Final Memory
2nd July – 8th July	• Sound • Museum adjustments • Post-mortem presentation at Oyfo museum • Final memory

6.-Results

The results of the game can be checked in:

GitHub: https://github.com/car1l2os/SolarRacer

YouTube: https://youtu.be/PQh7IpDIu3I

Also, this is one example game loop:

Figure 33 - Main Menu

Figure 34 - Number of players selector

In case of choosing one player (in Dutch: 1 speler) the next screen is going to be the difficulty selector. (FIGURAS)

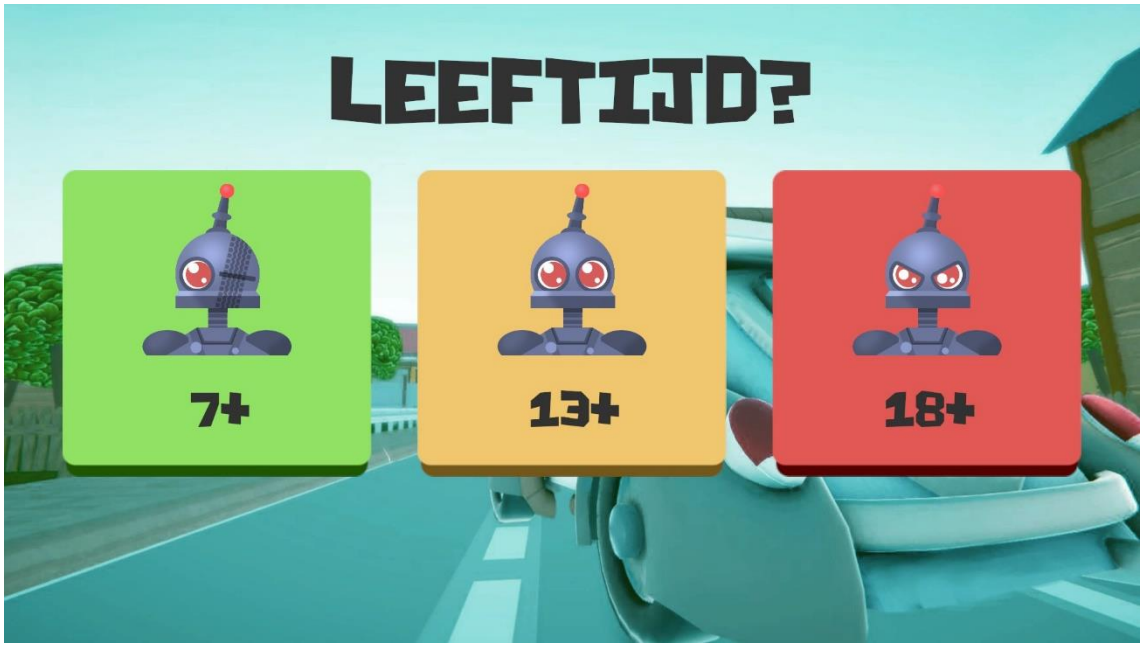

Figure 35 - Difficulty selector

The game starts with a traffic light doing the starting countdown animation (FIGURAS)

Figure 36 - Starting animation

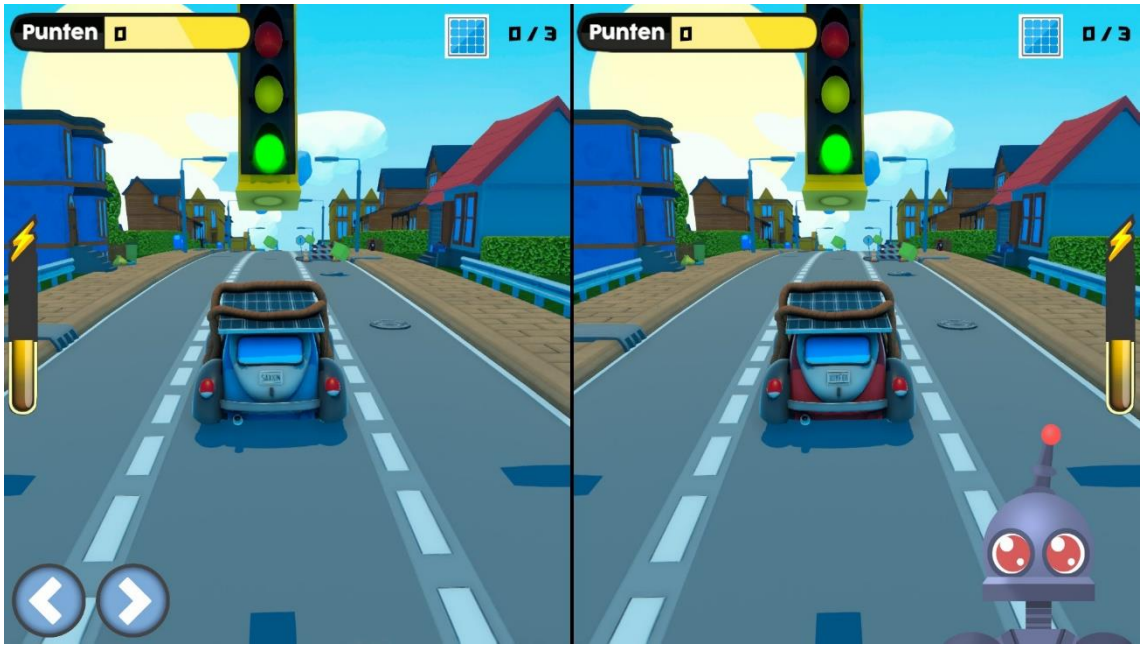

Figure 37 - End of starting animation

Depending of the difficulty selected (FIGURAS) the pertinent robot is going to appear in the bottom right screen of the player 2. If two players mode was selected a couple of movement buttons are going to be instead.

Figure 38 - Gameplay (Pick-up collision)

The AI's vehicle crash with an obstacle, it disappears and the crash particles are played.

Figure 39 - Gameplay (Obstacle crash)

Is possible jump over the trains that cross the road to avoid wait until it totally passed away.

Figure 40 - Gameplay (Jumping a train)

Both players have the power bar in the maximum value and it causes that wind particles appear in the screen. Player's vehicle can see the finish line. AI's vehicle is waiting for a train to pass.

Figure 41 - Gameplay (Final tile and train)

Figure 42 - Gameplay (Finishing the race)

In the score screen is possible to see the score acquired with the different elements in the game and your punctuation with stars -up to three- depending how good you did it in the race.

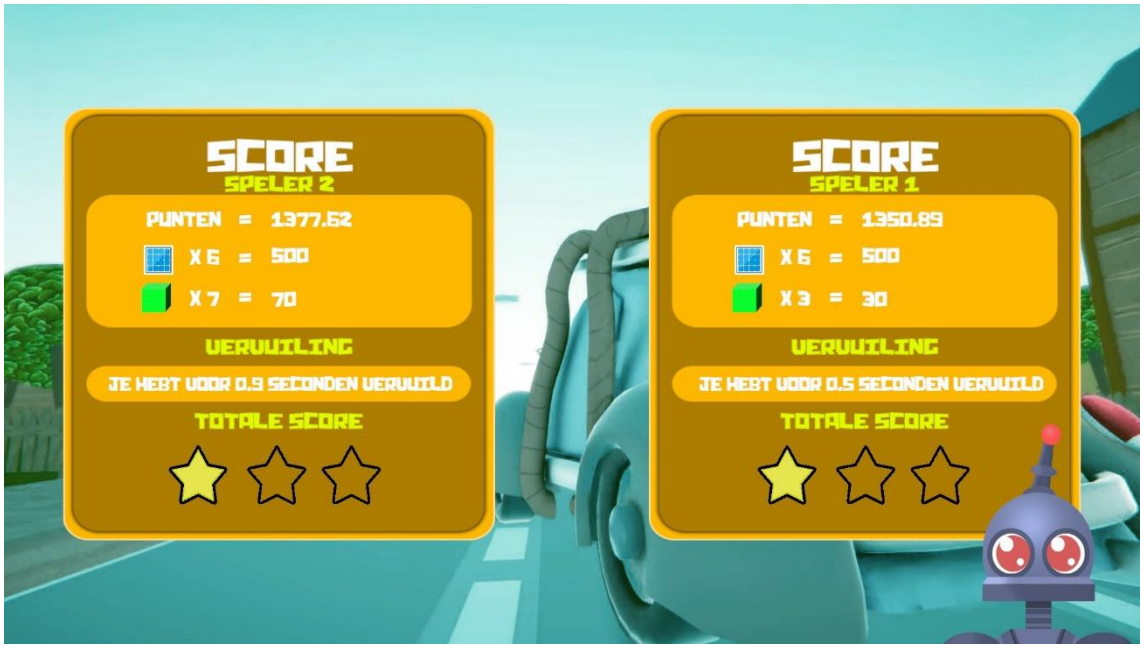

Figure 43 – Scores

The winner and loser are selected. This is based exclusively in who finished first the race.

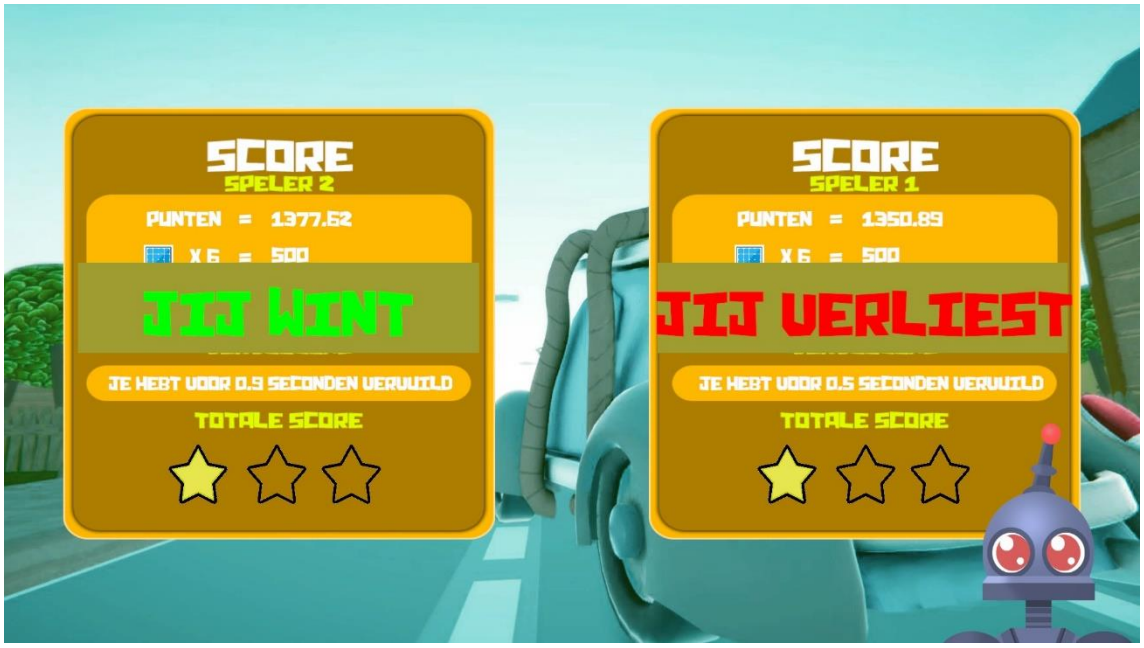

Figure 44 – Results

After the race –and only if a human player won- the game requires the winner to insert his/her name.

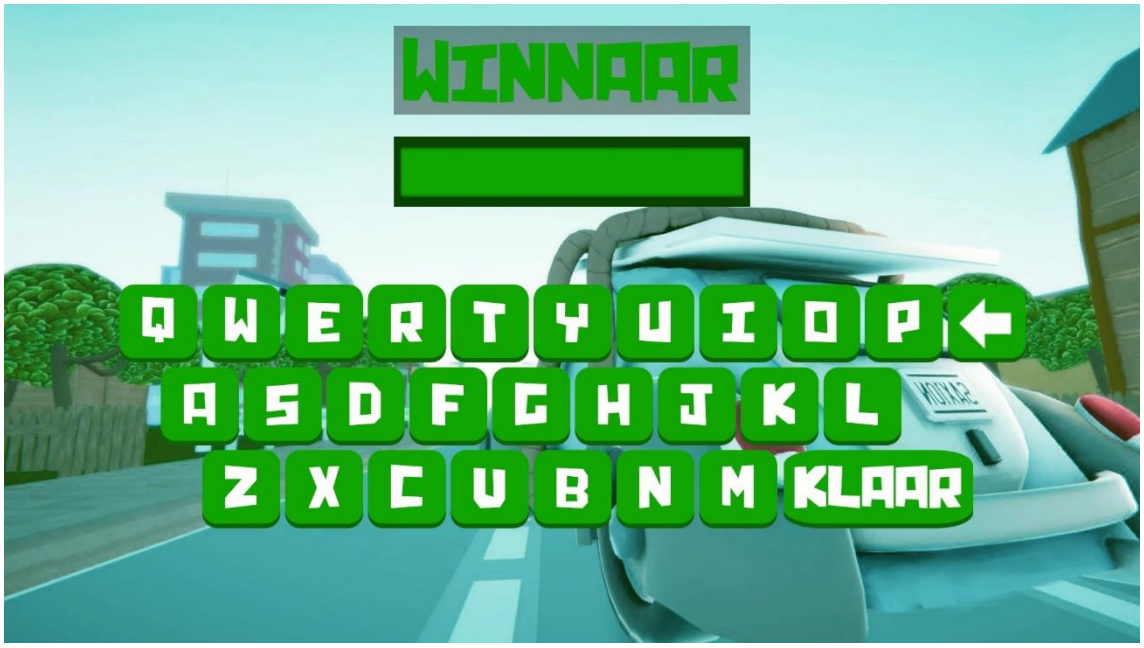

Figure 45 - Winner's name selector menu

After insert it the daily and yearly –VANDAAG and ALTIJD- leaderboard is showed.

Figure 46 - Scoreboard (after only one race)

To conclude a feedback screen is showed where the players can reply to some questions related like “What is your opinion about the solar energy?” and “How much do learned from solar energy with this game”.

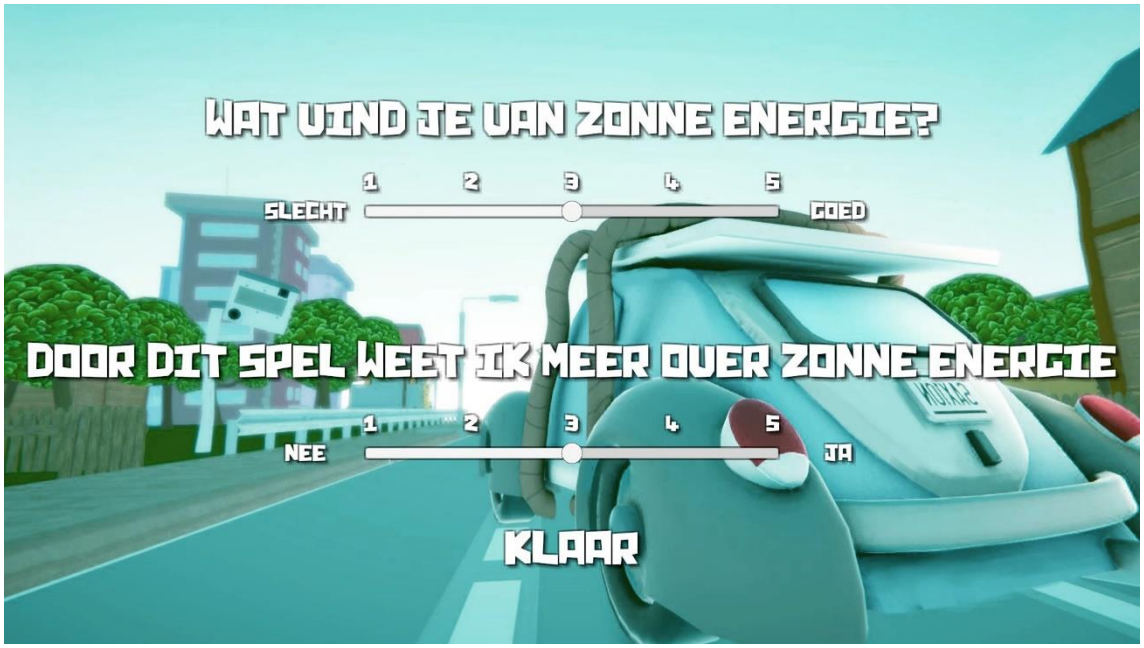

Figure 47 - Feedback scene

After reply the feedback the game returns to the main menu.

Figure 48 - Back to main menu

7.-Conclusions

Developing an educational game for a museum is a subtle task. It has to be appealing but visually clear. Fast but easy to understand. Funny but educational. All the process is walking on thin ice trying to balance different aspects that need to be in the game but not rule it. Dealing with the client is another point to take into account. Clients know what they want to transmit but normally they are not experts about how the process of development of a game or how to make a game experience attractive. Knowing when to cede to a client proposition and when to reject them in favor of the game is something fundamental. Of course in this situation is also really important to be able to express it in the correct way to persuade the client.

Another social factor in the development of a game is the teamwork. Working in a group or individually are really different experiences. On one hand, working in a group allows to develop bigger projects, on the other hand, make the project work can be much more difficult due to the necessity of share decisions and match expectations, work willingness or other fundamental points of the development. This is harder as bigger the team is and for this project, the team has been twice bigger than teams in the UJI's degree.

With all these conditions all the features required have been successfully developed and has passed the client selection and is going to be exposed in the OYFO museum. Interesting and easily exportable systems have been developed and different programming – as graphical computing, artificial intelligence or object-oriented programming – areas have been involved.

Figure 49 – SolarRacer logo

To sum up, SolarRacer has been a really good opportunity to experiment what a real development can be. With a client involved, a new environment and a relatively big team have been a good final to put in practice all the learned at the UJI in the last four years. From the technical part to the communication branch, going through art or game design, all the skills learned has been applied in one or other way. SolarRacer is a finished game susceptible to be published and that is the best of the possible conclusions.

8.-Bibliography

Images

Zucconi, A. (2017). Surface shader diagram. [image] Available at: https://www.alanzucconi.com/2015/06/17/surface-shaders-in-unity3d/ [Accessed 6 Jul. 2018].