Rave Rush

Controls

- N : Nitro
- R : Reset car
- B : Ball camera
- J : Jump
  - J + direction : In the air is double jump, spin towards the arrows pressed
- (WSAD) or (Arrow Keys) : Driving

Unity Overview

Unity is a game engine that excels at developing for multiple platforms. This engine was chose as some members of the group prefer Mac, and others prefer windows.

Unity allows for javascript and C# files, but in this project no javascript was used, only C#. Unity provides built in methods that are called for a variety of events. For example, the Start() method will be called when the object is first created, and the Update() method will be

called every frame. There are a lot more (such as TriggerEntered) but the most important ones to know are Start and Update.

Update is useful for aiding in animations, or for checking certain conditions at every frame. For example, when doing the “3,2,1” countdown UI effect, the size of the text is changed by a small amount at every frame, which is illustrated in the following edited version of the script.

```
private float textScale = 0;

void Start() {
   Transform eventTextransform = eventTextGameobject.GetComponent<Transform>();
}

void Update() {
   if (textScale < 0.015f) {
      textScale += 0.0003f;
      eventTextTransform.localScale -= new Vector3(textScale, textScale, textScale);
   }
}
```

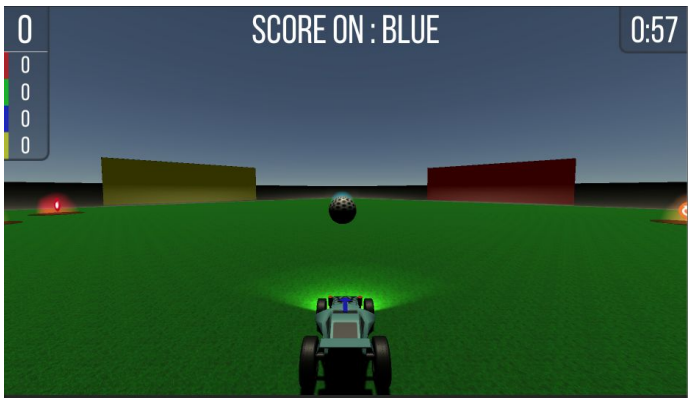
This code snippet shows that the scale of the countdown text is decreased at every frame. Once the text reaches a certain size (0.015f) it will stop scaling the text. In another part of the code, it resets the scale of the textbox and displays the next number to simulate the countdown effect.

In Unity, the GameObject class is the base class for all entities. A GameObject itself does not accomplish much, but acts as a container for components.The Component system is how you get real functionality out of the game object. In the previous example about the countdown event, we can see that the localScale of the event text transform is what gets manipulated each time the update is called. The transform is the component that is manipulated, not the GameObject To get a component of a GameObject, you need to use GetComponent<Name>() as illustrated in the start method of the example.

Gameplay

Overview

The gameplay revolves around trying to put the ball into the specified net. When the game starts, there is a countdown and then the user is allowed to move. The game tells you where to score (As seen on the top, or the colour of the arrow on top of the car).Once the user scores, the ball is returned back to the center, and another random colour is chosen as where to score next. When the timer reaches 0, the game pauses and all motion is slowed and user input is disallowed. The



game gives you 5 seconds to look at your stats before returning to the main menu.

The user can use the ball camera (press B) or use the arrow on top of the car to locate the ball in the scene. The car can also jump and flip in various directions in order to help with hitting the ball. There are obstacles around the scene which affect the car (Spin and jump) or the ball (blows it up). When the user hits the ball hard enough, a sound and custom effect will occur.

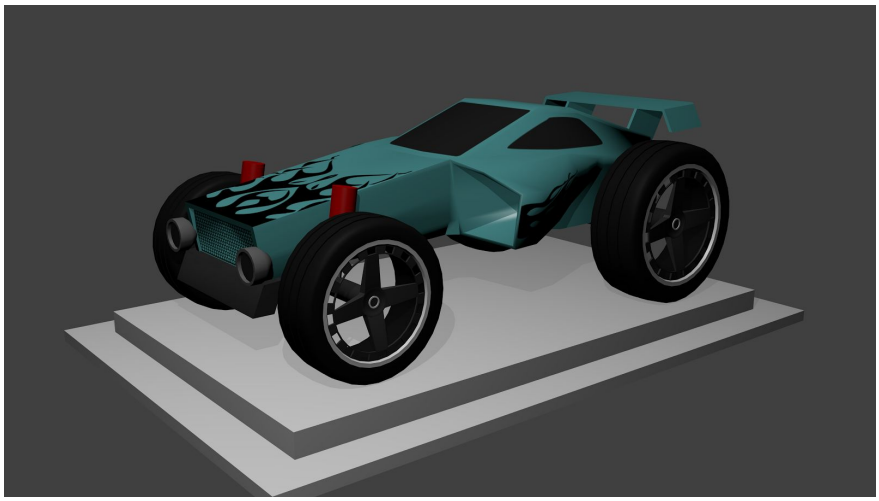
Features

- 3 Different obstacles
  - Spin
  - Jump
  - Destroy Ball
- Car
  - Created in blender
  - Custom physics
  - Can jump
  - Can jump and flip in specified direction
  - Nitro boost
  - Headlights
  - Wheels
  - Working suspension
  - Custom model and texture
  - Arrow which points to ball
  - Can climb walls
  - Sounds
    - Engine
    - Collision
      - Ball
      - Environment
  - Working collision
- Goal Mechanics
  - Custom triggers to handle goals
  - Explosion effect when the ball is scored
  - Ball respawns at center after each goal
  - Scoring custom sound
  - Scoring custom text animation
- Two different camera modes
  - Follow ball camera
  - Follow car camera
- Ball object
  - Textured
  - Uses gravity
  - Bounce physics
- Main Menu
  - Different modes
  - Different game lengths
  - Custom UI
  - Main menu changes based on where the mouse is
- UI
  - Custom UI
  - Custom image overlaid on canvas for HUD
  - Displays time left
- Rave Mode
  - Has different sounds
    - Background noise
      - Had to tweak to loop properly
    - Engine noise
      - More arcadey noise
  - Has custom lights which spin
  - Custom lights determine the colours of the goals rather than static sides
  - Custom arena
    - No coloured sides
    - Different ground texture

Models

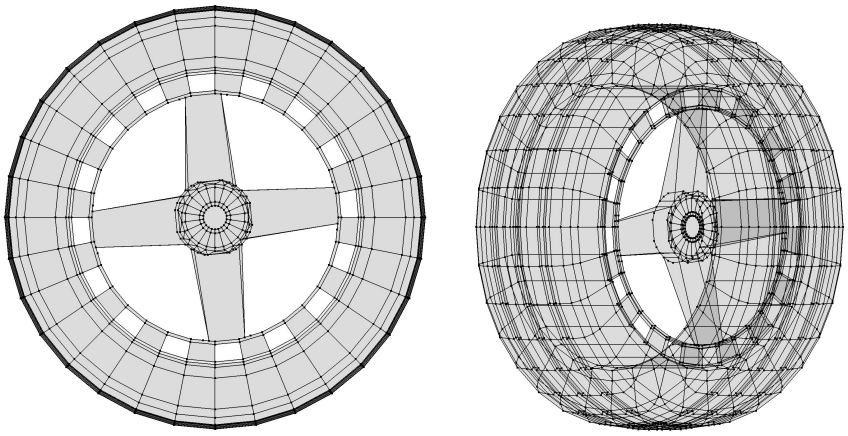
All models were created in Blender which is a free 3D modeling software. The group members had next to no experience with building 3D models and so watched tutorials and self taught ourselves. Two car models and two arenas were created, the first of each was replaced by the second version as the second iterations were better than the original models. The cars took a few days of work with modeling and texturing.

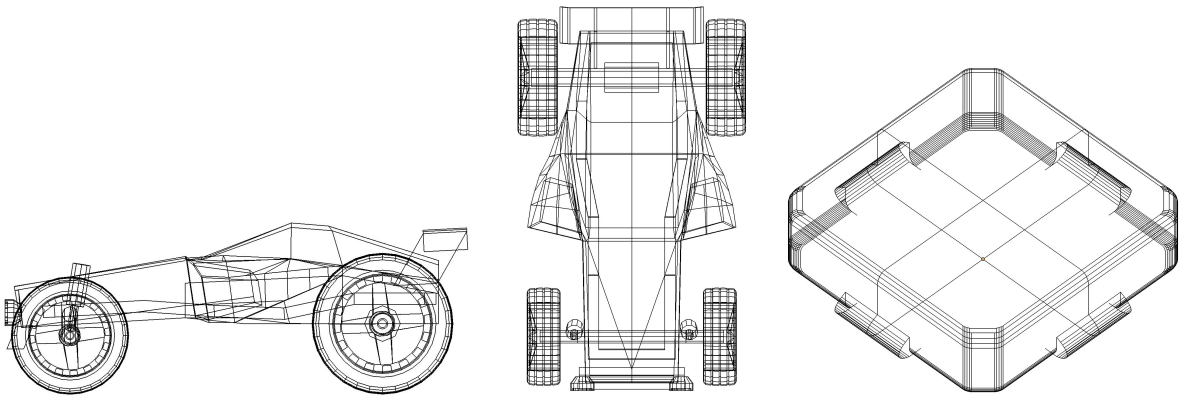
Texturing models had to be done in both Blender and Unity. For complicated models like the car body, the model was mapped to an image inside of Blender, then took that image into photoshop to draw on it. For simple objects, just apply a solid material colour to different faces of the model. When importing the models into Unity the materials and textures for the different surfaces of the model.



must be reapplied or recreated.

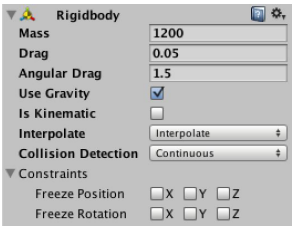
The main arena was created in Blender as one object. Only one side and one corner had to be created and then was mirrored. What was difficult with this, and still remains a problem is the collision between the car and the arena at high speeds. If the car is driven fast, straight at the curved part of the wall, it will collide harshly instead of driving up the wall. The walls of the arena have a transparent, glass like look that allows your to see the sky outside. The curved walls were given an asphalt like texture and smooth shading so the curves look good.





Physics

General

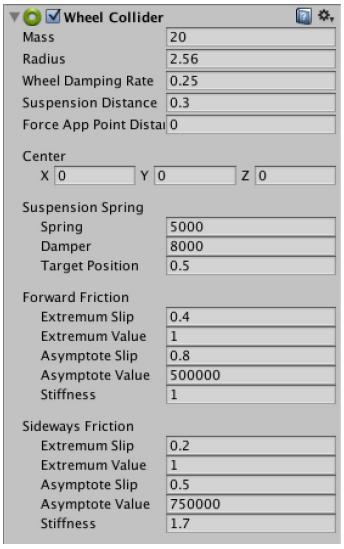


Unity uses Nvidia's PhysX physics engine built into their platform to attempt to have realistic physics. During the testing of the application, the physics was found to be far from realistic.

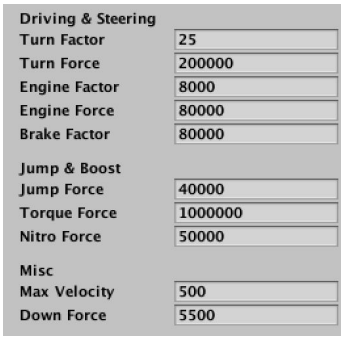
In Unity, if you want a GameObject to be affected by physics then you add a Rigidbody to it. The Rigidbody has mass and drag properties that the physics engine uses to determine its behaviour. However, on its own using the default gravity value and a realistic mass for an object, the gravity seemed very weak. To make the gravity more realistic it was changed from -9.81m/s to -40m/s, which doesn't follow any real life physics.

The second big problem with Unity's physics is the wheel friction. In real world physics if a car wheel had high friction it wouldn't slip and hypothetically if it was extremely high it would be able to grip up walls. In Unity however, high friction causes the car to bounce all over and be impossible to maneuver. Increasing the mass or applying a downforce to counter this odd behaviour did not work.

Car



The car physics was one of the most difficult parts of this project. Unity has a built in WheelCollider that attaches to each wheel and you can then apply steering, engine or brake forces to the wheels. The WheelCollider also has many physics settings that includes mass, suspension, forward and sideways friction. It took a lot of testing, research and tinkering to get the numbers how they currently are, but the it still is not where it should be. The car still behaves strangely and further testing is required



The most difficult setting with the Wheel Colliders was the friction, too high and the

car would bounce all over but too small and it would slide all over and not go very fast. To counter this, for driving and turning additional forces were applied. So when driving forward, along with the WheelCollider engine power, a relative force is applied in the forward direction. The same is done for steering but apply a relative torque force instead of a linear force.

To allow the car to drive up the walls and to help with the friction, a relative constant downforce is applied on the car's physics body. This makes it so that when driving up a wall it would have a force keeping it there similar to gravity in normal conditions.

For the different effects on the car (such as the jump and spin obstacles or pressing J to jump and spin) forces get applied to the physics body of the car.

Ball

The physics of the ball was fairly simply to do. It was assigned a Rigidbody with a mass and then gave the ball a physic material with a factor of bounce that gave it a fairly realistic and fun feel.

Landscape

With the arena, each of nets have a trigger so when the ball goes through it, it triggers an event that is used to increase the score. When a goal is scored an explosion is triggered that acts like the ball has exploded, then the ball is returned to the center of the arena.

There are 5 obstacles on the map that do different things to different objects. There are two spin and two jump obstacles that apply forces on the car when triggered. The bomb obstacle acts only on the ball and it rotates around the arena, when the ball collides with this obstacle it explodes and gets returned to the middle of the arena.

Here's the code for the obstacle that makes the car jump

```
void OnTriggerEnter(Collider other) {
   if (other.tag == "Car")
      car.AddForce(Vector3.up * 70000, ForceMode.Impulse);
}
```

When the ball enters this trigger zone, an up force of 70000 to it (numbers just came through testing, as it depends on the mass assigned to the car).

Sounds

Overview

Timing the sound to work with the animation was one of the main goals during editing of the sound in audacity. This is shown through the background music[2] as it was edited in order to fade in and out properly. It was also edited to loop correctly so the background music will keep playing until the game is over. Timing it to fade in with the initial countdown was a challenge.

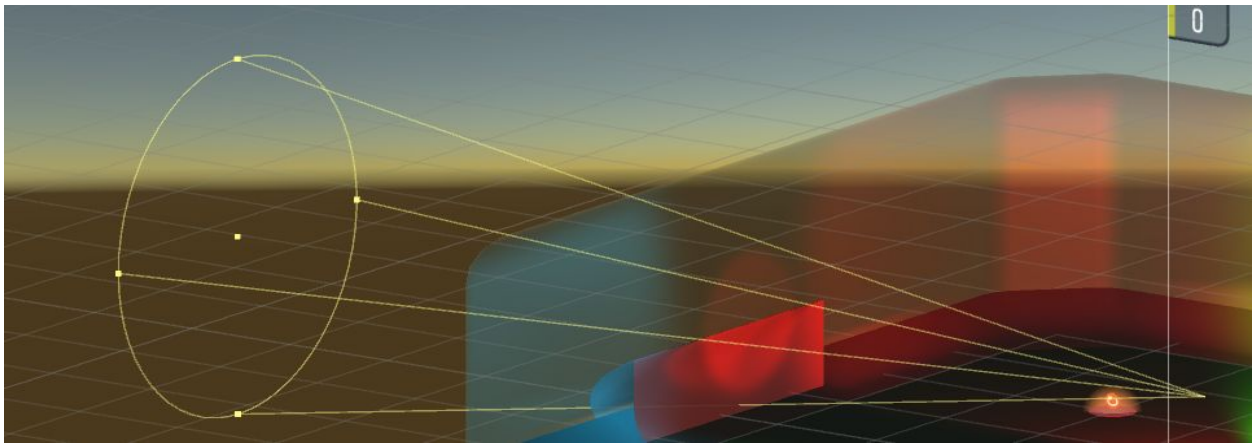
The robot voice effect was created by applying a variety of changes to the initial voice (mainly splitting into two channels and altering the pitch and bass), and the background sounds in the clips were sampled from various sources[3][4] .

Unity requires that a “Music Source” component be attached to an object to act as the source of all sound. In order to play a sound you want, you pass this source a sound file and tell it to play the file after a specific action (Eg. When the game starts the countdown sound is played).

Lighting

Unity provides us with a selection of different light presets. The two used for this project were spotlights and directional lights. This lighting system makes it very simple to set up, as it does all the rendering and calculations for you. The user still needs to come up with the values that will define the light (such as intensity and range)

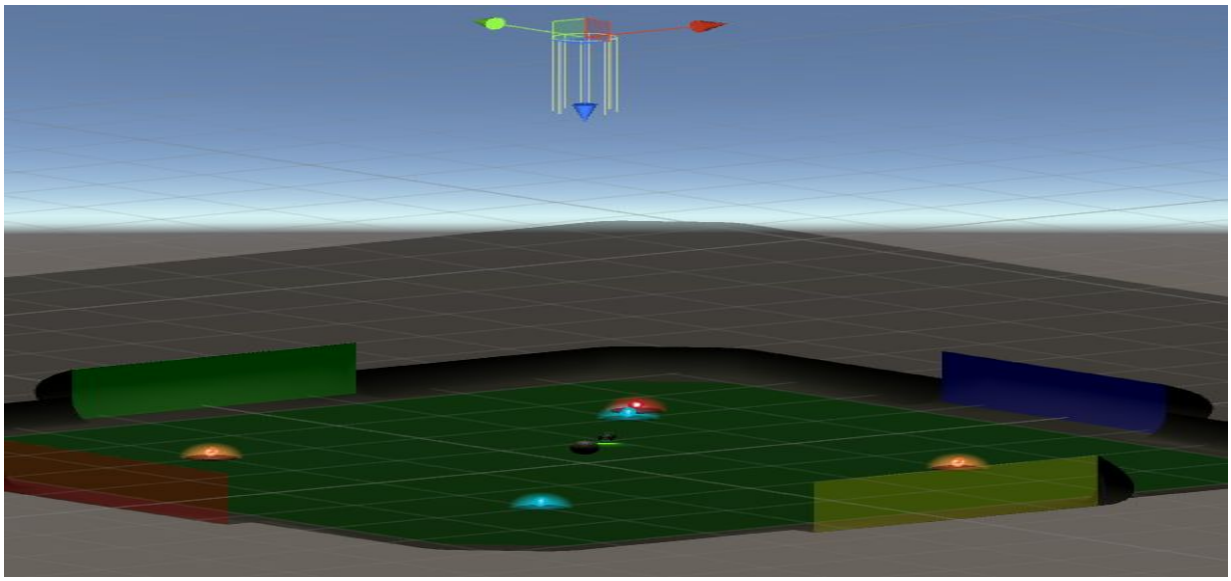
Spot



Spotlights provide a cone-shaped region of illumination which is useful for simulating artificial light sources. Spot lights were used in the Rave mode as well as the headlights for the car.

Implementing the rave mode scoring based on the colour of light shining at the goal was a challenge. The final solution was to assign each of the nets a “Trigger” area, and assign each of the four Trigger areas to one of the four lights. Each net then selects what colour is shining on it (as in which colour goal it currently is)by using the rotation on the Y axis of the attached light.

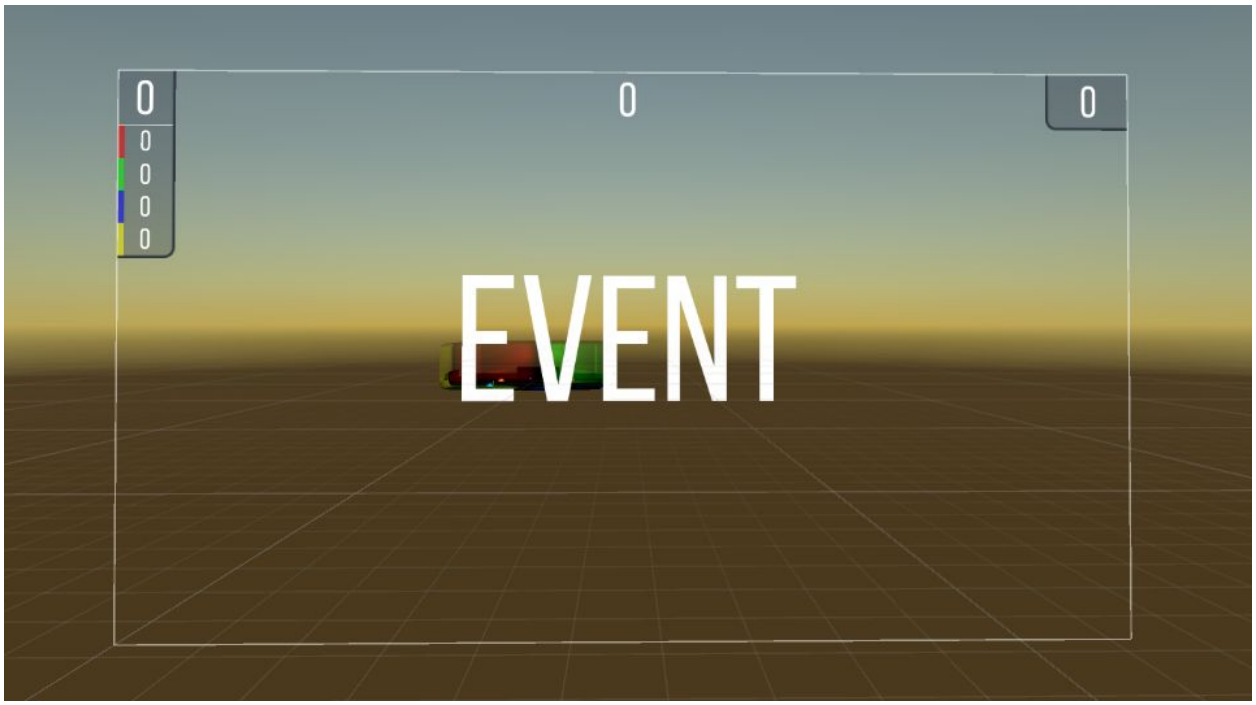
Directional



Doesn't have a direct source of illumination, but acts as if the source is a far away object (Like a sun). This lighting was used in plain mode to simulate sunlight.

UI

Overview



Unity allows for a UI to built on top of a canvas. This canvas object is a 2D rectangle that's overlaid on the scene. A 2D image was created in photoshop which was then placed on the canvas to function as a HUD. A custom font (Bebas Neue) was used to populate the information in the HUD (such as goals scored).

There is a hidden text object used to display "Event" text such as goal scored and the countdown. There are different animations applied to this box when the text is being displayed which are handled by altering the physical properties of the Transform component of the Text object. These events also trigger certain sound files when they are occurring.

Unity does not allow for changing of the text size, as it is an immutable value in the Text object. This was circumvented by changing the Transform element of the Text object. In order for this to work, the text size has to be set to a large value(It has a font size of 200), as trying to stretch a smaller font size rather than scale down a larger size makes the text distorted. The only other option is to load a new font into the textbox on every update, but that approach is terribly inefficient (the goal text animation is 150 frames long, which means it would be loading in a new font 150 times per goal)

Sources

[1] http://docs.unity3d.com/Manual/index.html

[2] Robbie Rivera. *That's a Good Meatball*. 2013. *Soundcloud*. Web. 01 Jan. 2016. <https://soundcloud.com/dimmakrecords/thats-a-good-meatball-teaser>.

[3] Bioweapon. *Eternal Vision*. 2011. *Youtube*. Web. 01 Jan. 2016. <https://www.youtube.com/watch?v=2bQLGaToUdU>.

[4] Bobby Green. *Velvet Queen*. 2013. *Soundcloud*. Web. 01 Jan. 2016. <https://soundcloud.com/bobbygreen/velvet-queen-feat-tiger-la>.