Road Crossing Game Template

Game documentation and HowTo guide.

This document contains:

Package Description and features

Road Crossing Game is a full Unity template ready for release. It is compatible with mobile as well as standalone. It is similar in genre to classic arcade games, as well as modern infinite runners.

How to Play?

Use the keyboard arrows or the mouse to move around, collect coins and avoid being killed by the many deadly obstacles. The game controls also fit mobile and console without need for further coding.

Update history

1.48 (26.10.2017)
- Support for versions of Unity 5.5, Unity 5.6, and Unity 2017

1.47 (14.05.2016)
- Added moving platforms you can walk on (logs floating in the river).
- Support for versions higher than Unity 5.3.
- Better support for UnityAds 5.2 and above.

1.45 (24.12.2015)
- Added support for the **Animator** component, along with an example player.
- You now have an option in the game controller to stop powerups on death.
- Added support for Unity 5.3 SceneManger.
- Uploaded versions for Unity 5.1, 5.2, and 5.3.
- Bug fixes (static speed multiplier, thicker move limits, victory screen).

1.41 (30.11.2015)
- You can now have a victory condition in randomly generated levels too.
- New lane sets including 3x lilypads, 5x lilypads, and the super buzzsaw!
- Continuous tap/mouse control allows the player to be more responsive.
- Added MainMenu/Restart/Resume buttons to pause screen.
- Reorganized JS/C# folders to make sure there are no clashes.

1.36 (24.10.2015)
- Added an option to create custom levels (non-random), with a victory screen and animation. Example scene with custom level also included.

1.34 (12.09.2015)
- Added support for UnityAds along with a quick integration guide.

- Improved controls responsiveness by caching the next move the player
chooses (Similar to how in PacMan you can give the next turn command
before you finish your current move)

1.3 (27.07.2015)
- Added river lanes with lilypads you can jump on.
- Added 2 new characters: A sheep and a duck!

1.27 (10.06.2015)
- Added 3 new characters: A goat, a turtle, and a koala.
- Video tutorial showing the process of creating a new character covering everything from modeling in 3DS Max, UVW editing, texturing, and then exporting and implementing the new player into Unity, and adding it to the shop too.
- When the player dies a ghost spawns which can be moved to a safe area before respawning the player.

1.23 (21.05.2015)
- Added lives to the player. When the player dies he loses a life and. Also added an Extra Life item you can pick up. Dying in the fog pushes the fog a little back.
- Video tutorial shows how to add new enemies to the game, put them in a lane, and make the lane appear in the game.
- Added script to remove some of the objects after a certain distance. This helps improve performance.

1.2 (03.04.2015)
- Added a powerup system. You can collect items throughout the game which give you double coins, slowmotion, etc. You can also add your own powerups to the list.
- Improved drawcalls by setting the receive/cast shadows state on renderers to one or the other, rather than both.
- Improved object drops on lanes, allowing for a list of objects that can also drop in a sequence, for better gameplay consistency (random drops are also an option).
- Set next lane position to be public. You can now see a red line which represents the placement of the next lane in the game.
- Sounds now play regardless of whether there is an animation or not.
- Fixed the minimum gap between two objects in C#.
- Fixed TextByPlatform in C#.

1.18 (05.03.2015)
- Added a shop where you can buy and unlock new characters to play as. The coins you collect during gameplay are stored and can be used to buy new cute animals.
- You can toggle swipe controls for mobile devices. Swipe in a direction to move the player.
- You can toggle the random move direction for the objects on a lane.
- You can also set the width of each lane, so you can now have lanes of various widths.
- Locked movement of player to the grid.

1.1 (12.02.2015)
- Entire project ported to C# thanks to Jordan Swapp. You now get the project in both JS and C#.
- Improved performance and reduced drawcalls.

1.03 (09.02.2015)
- You can limit the speed of the Death Line.
- You can set the minimum gap between moving objects.
- MAX models are now included in the package
- Updated tutorial to cover reskinning 3D models, creating new models, and creating new tiles.

Credits

The sounds are courtesy of the free sound project.

Intro music is a clip from Vivacity by Incompitech

Credits go to these authors for their great sound samples: **mattj99, anechoix, joelaudio, tramppa34, fenrirfangs, zerolagtime, atomwrath, jc144940, greencouch, ERH, sleepdust, kastenfrosch**

Please rate my file, I'd appreciate it 🙂

Overview of the game's library contents

Let's take a look inside the game files. Open the main RCGAssets folder using Unity3D 4.6 or newer. Take a look at the project library, usually placed on the right or bottom side of the screen. Here are the various folders inside:

- **Animations:** Holds the animation clips made with Unity's built-in animation system.
- **FLA:** Holds the object graphics made with Flash CS3. These are vector graphics than can be easily scaled without loss of quality and then exported as PNG to be used in Unity.
- **Fonts:** Holds the font used in the game, AGENCYB.
- **Prefabs:** Holds all the prefabs used in the game. These are distributed to various folders for easier access, Buttons, Enemies, Objects, etc
- **Scenes:** The first scene that runs in the game is MainMenu. From this scene you can get to the Game scene.
- **Scripts:** Holds all the scripts used in the game. Each prefab contains one or more of these scripts.

- **Sounds:** Holds all the sounds used in the game. Jump,Item, etc
- **Textures:** Holds all the textures used in the game which are used as sprites in Unity.
- **UI:** Holds all the canvases the game which are used to hold buttons and other UI elements.

Customization Guide

Getting started

Road Crossing Game (RCG) is considered a complete project, and as such is supposed to work as the starting point of your planned game, rather than an addition to an existing project. That said, you may of course pick and choose some of the scripts/models to import into your existing project, but RCG works best as a starter kit which you can customize any part of to your liking.

The Game Controller

The Game Controller is the main prefab that controls all the progress of the game from start to finish. It controls the UI of the game, creates lanes, and also makes the camera chase the player. The Game Controller is also used to increase the difficulty of the game after collecting a set amount of coins.
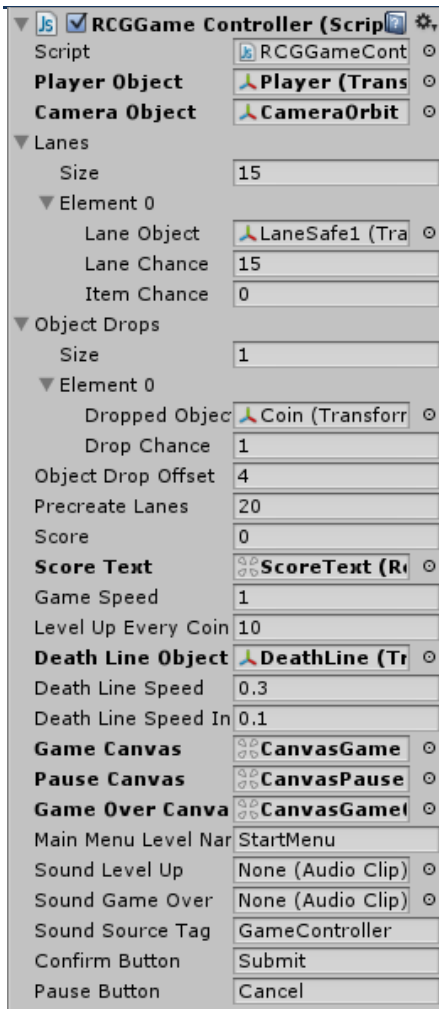
Player/Camera Object – These two must be assigned from the scene to make the game playable. The GameController makes the camera follow the position of the player as it moves around.

Lanes – This list contains all the different lanes that are created as the player goes forward in the game. You can set the chance for this lane to appear, as well as the chance of an item to appear on this lane.

Object Drops – This list contains all the items that may drop in a lane. You can set the chance of a certain item to appear.

Object Drop Offset – This is how far along a lane an item can be dropped.

Precreate Lanes – How many lanes to create at the start of the game.

Changing Game Difficulty

The GameController also allows you to change the difficulty of the game through several variables, making the death fog line get gradually faster with as you rise in the levels.

The Death Line:

The death line is the object (The fog) that constantly chases the player as he advances through the level.

You can control how fast it advances toward the player by changing the **Death Line Speed** field, and you can also set how much faster it becomes **(Death Line Speed Increase)** after collecting a number of coins **(Level Up Every Coins)**.

So based on the settings below, the speed of the death line increases by 0.1 after collecting 10 coins, and keeps doing so until it reaches the maximum speed of 1.8.

Level Up Every Coins – Is the number of

coins you need to collect to level up.

Death Line Object – Is the object that keeps chasing the player.

**Death Line Speed –** The movement speed of the death line.

Death Line Speed Increase – The increase in the speed of the death line.

Death Line Speed Max – The maximum speed of the death line.

Notice that the death line will always catch up to the player whenever he is advancing in the level. This is a game design decision I made. If you think it should be made otherwise, tell me.

If you don't want the pressure of fog behind you, you can simply clear the **Death Line Object** field. Of course you can have all kinds of death lines instead of the fog, like a set of spinning spikes for example, or a horde of blobs. Just ideas!

Another way to increase overall difficulty in the game is to change the attributes of the lanes. Here you can change the number of moving objects that appear in a lane, and the movement speed of the objects.
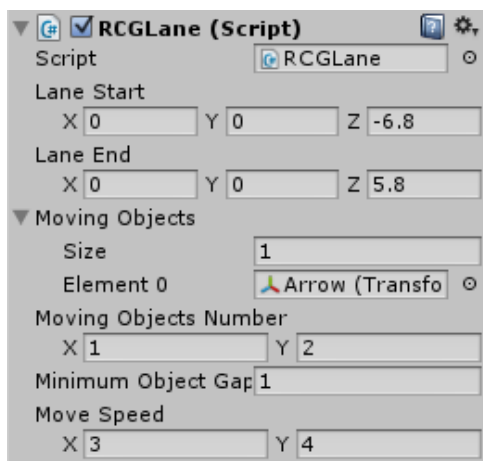
Lane Start/End – These are the points that objects move through.

**Moving Objects –** This is a list of the objects types that will be created in a lane.

Moving Objects Number – A random range for the number of objects that will be creates in the lane. The objects are scattered along the lane with some randomization in position.

Minimum Object Gap – The minimum distance between two moving objects in a lane. This is used to prevent any objects from overlapping each other.

Move Speed – A random number between these two values is chosen as the speed of all the moving objects in this lane.

Editing the Player

The player object has several variables that can change how it behaves, mainly the move speed death effects.
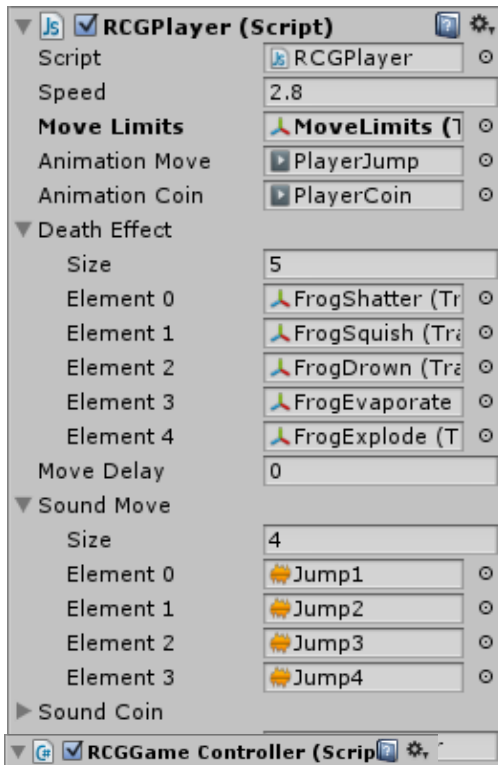
Speed – The movement speed of the player.

Move Limits – This object is assigned from the scene, and it has colliders that blocks the player's movement to the sides by bouncing it back.

Death Effect – A list of effect objects that are created after the player object is removed. This is decided by the object that kills the player.

Sound Move – A list of sounds that are played randomly when the player moves.

Sound Coin – A list of sounds that area played randomly when the player gets a coin.
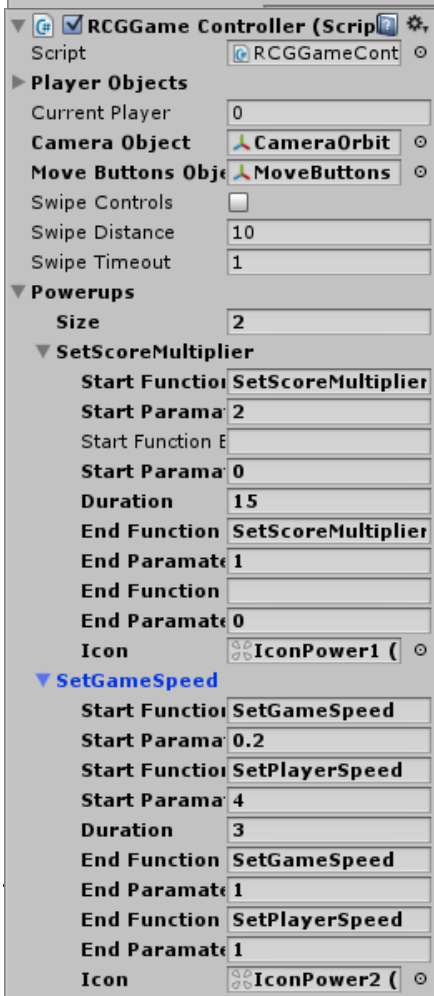
Editing Powerups

A new addition to RCG 1.2 is the ability to add powerups you can pick up in the game. The list of powerups you have is defined in the game controller. When a powerup is activated it activates up to two functions in the gamecontroller, using SendMessage (Similar to how blocks and enemies work).

Let's see how the powerups are listed. The first power up doubles the score from collecting coins. It does so by sending a **SetScoreMultiplier** function command with a parameter of 2. Then the game counts 15 seconds before sending a **SetScoreMultiplier** function again with a parameter of 1, making the score multiplier return to normal. Finally we assigned an icon to represent the power up

duration while it's activated. You can see what the icon looks like below.

Another powerup we have slows down everything around the player for 15 seconds. This is done by sending a **SetGameSpeed** function command with a parameter of 0.2 (so the game is 80% slower). Another command we send is **SetPlayerSpeed** with a parameter of 4 which increases the player's speed to 400%. We did this so that the player can still move relatively fast while everything else is slowed down. To end the power up duration we wait 3 seconds and then call the same functions but with a parameter of 1 for each, making the game speed and player speed both go back to 100%. Note that we set the duration of the slowmotion to 3, because the game is already set to 20%, so 3 seconds will last 15 game-time seconds. 3*5 = 15 seconds.
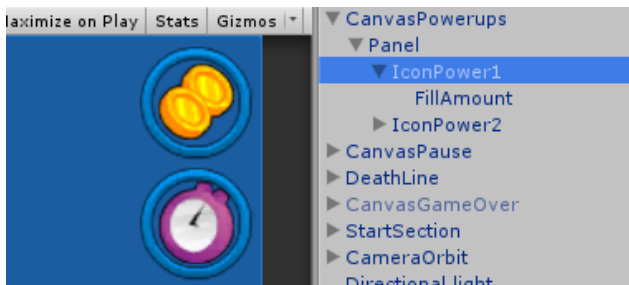
This is the powerup icon, made of an icon and a fillamount circle which shows the duration of the powerup.

Now that we listed the possible powerups, we need to put items in the game which activate those powers when picked up. We also do this in the game controller.
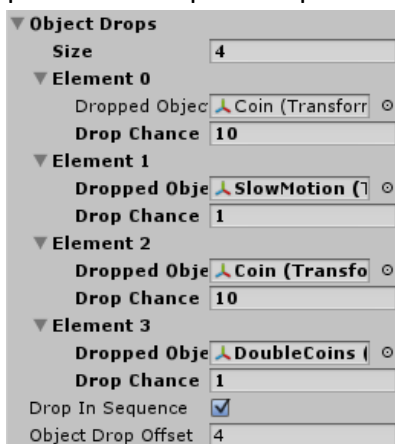

Note the **DropInSequence** toggle at the bottom. This switches the drop method from random based on **DropChance**, to sequential making objects drop one after the other, and making **DropChance** decide the number of objects to be dropped before moving on to the next object type in the list.

So in our case we started with 10 **Coins**, then dropped 1 **SlowMotion** item, then 10 more Coins, and finally 1 **DoubleCoins** item. This sequence is then repeated from the start.

If you don't like the drop sequence, you can tick it off and then objects will drop randomly with a higher drop chance if you set **DropChance** to a bigger number. In this method you only need to set each type of drop in the list once (no need for Coin to be listed twice).

Let's take a look at the items we pick up in the game and how they activate a powerup. Go to the **Prefabs > Items** folder and drag **DoubleCoins** and

SlowMotion items to the scene. The component in those prefabs is the same as the component in the blocks, enemies, coins, etc. (RCGBlock)
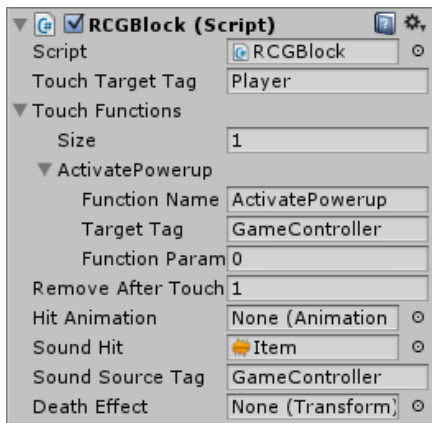

Click on the DoubleCoins object and you'll see the component, in which we called an **ActivatePowerup** function command from the gamecontroller, with a parameter of 0. This activates the *first* powerup in the powerups list in the gamecontroller, which is the **DoubleCoins** powerup. If you take a look at the **SlowMotion** object you'll see that the parameter we sent is 1, which activates the *second* powerup in the list.

Victory in Randomly Generated Levels

In the package there is an example scene (**CustomLevel**) showing how to create a custom level with predetermined lanes, which end with a victory lane with a sound and animation followed by a victory screen.

You can have a similar victory condition in randomly generated levels too. To do this you must set a **Victroy Lane**, **Lanes To Victory**, and a **Victory Screen**. When these three are set, you'll get a random level which ends after a set number of lanes.
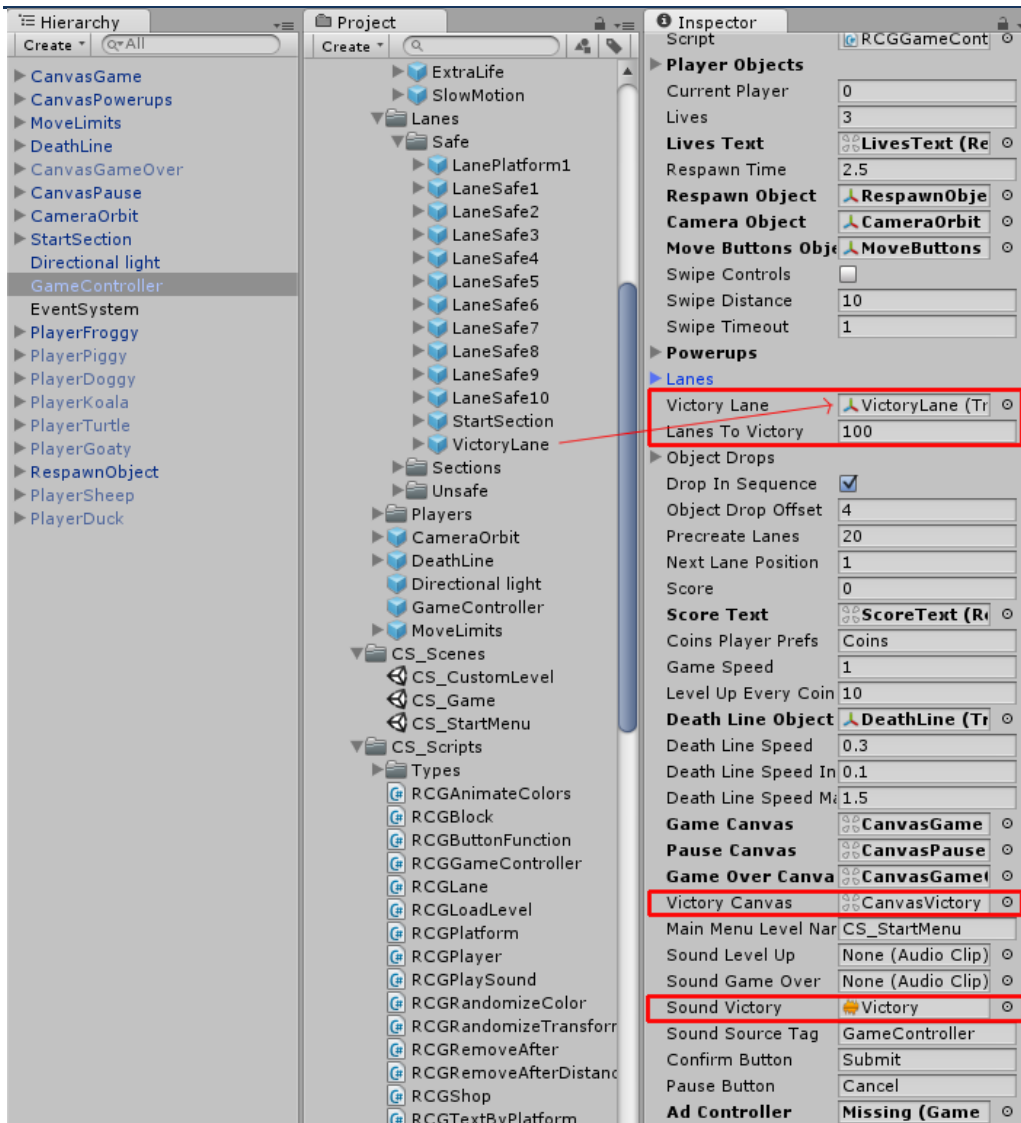
The **Victory Lane** should be dragged and dropped into your victory lane slot in the game controller. This lane contains a block which calls the victory function in the game. Make sure you assign the victory lane with the correct script type (JS or CS).

The number of lanes to victory must be more than 0 to work. For example if you set it to 100, you will get a victory lane after 100 regular lanes have been created.

The **Victory Canvas** is the screen that appears after you win the game. This is assigned from the scene itself, similar to how the **Game Over** canvas is assigned.

You can also assign a special **Victory** sound to play on victory.

You'll also notice that the player has a special animation when winning the level. This is assigned from the player object, like this:
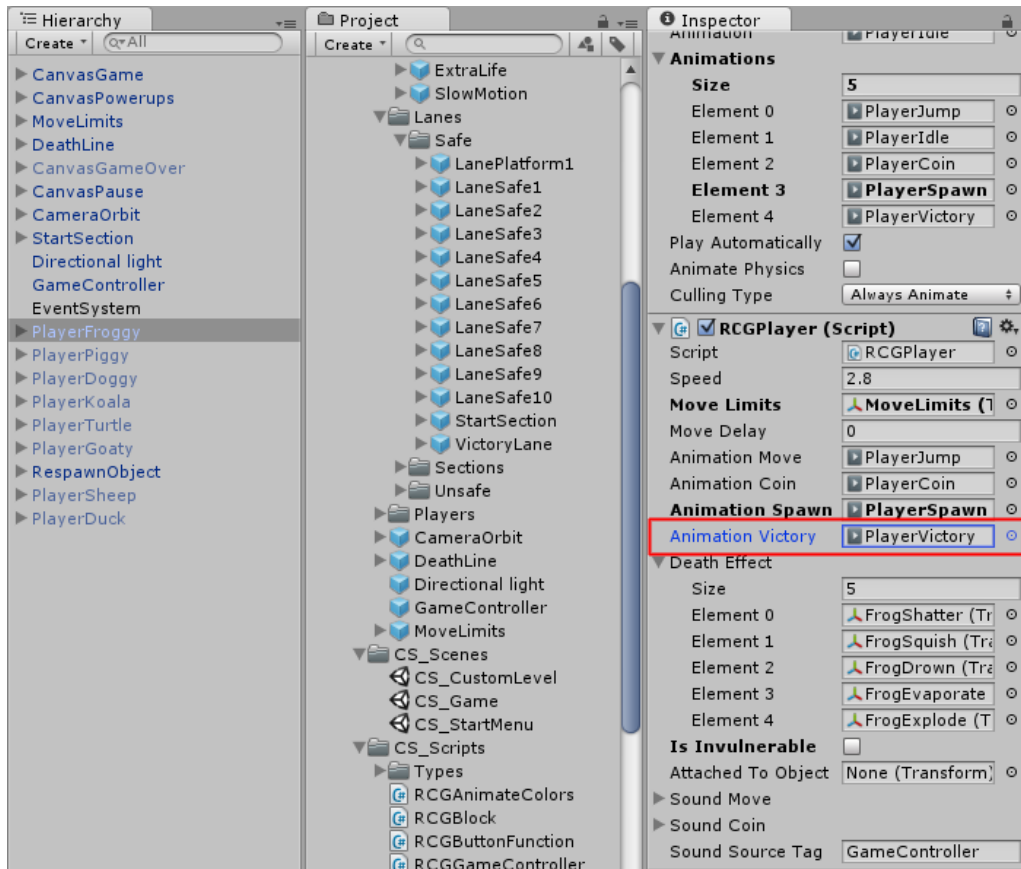
Changing Graphics

Graphics in Road Crossing Game are 3D. They are modelled in 3ds Max and then exported to FBX to be used in Unity. I included the MAX files in the package; you can find it in ExtraAssets.rar. The reason for having it inside an archive and not in a simple folder in the project is because unity has trouble dealing directly with MAX files, so when you have any MAX models in unity the software will hang up for a while in order to accommodate the new objects. So my recommendation is to keep these files just outside of your project folder, like this:
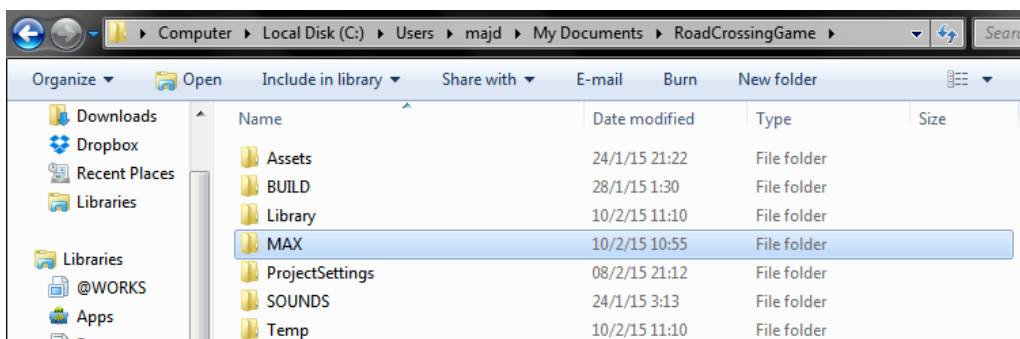
The textures are made using Flash CS3, and then exported to PNG. I do this because Flash allows me to control the quality of the textures easily since it is vector based graphics.

The various animations are made in unity itself and are assigned to generically named parts of the model, so that you can easily switch models without while maintaining the animation on the object.

Reskinning 3D Models

Here's how you can put your 3D models instead of the original ones, using the "reskin" method. Using this method you can simply "reskin" your model with another model without having to worry about creating new prefabs and assigning meshes.

- The package includes a MAX folder that contains all the models in the game. You can open this in 3DS Max 2014 or above. Open the file Models.max. This contains all the models in the game.

- Now you can edit each model individually, and check how it scales relative to other objects. Once you are done, you can export the whole file as FBX to be used in unity.
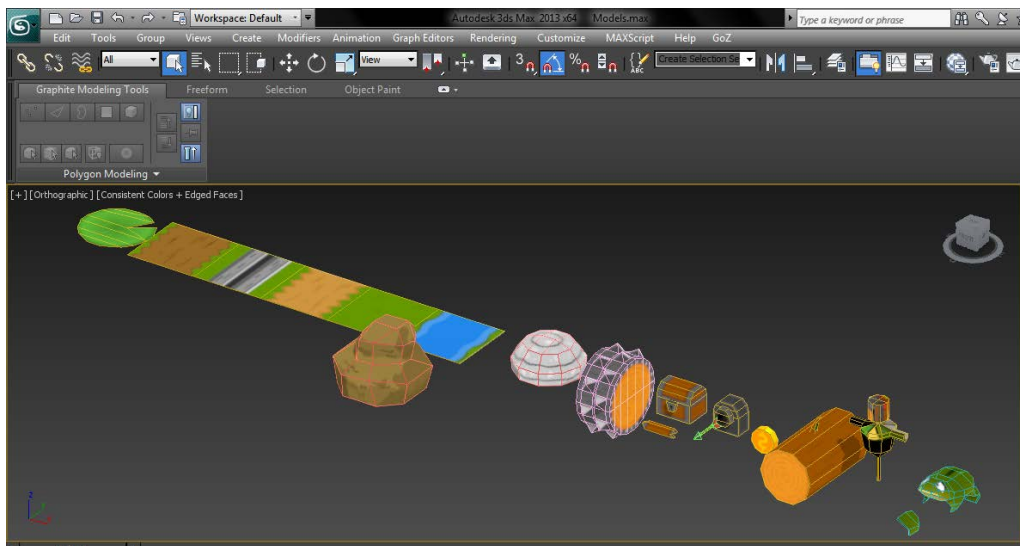
- For our specific example I will replace the frog model with a blob model. First I shift+drag the blob object to create another copy of it.
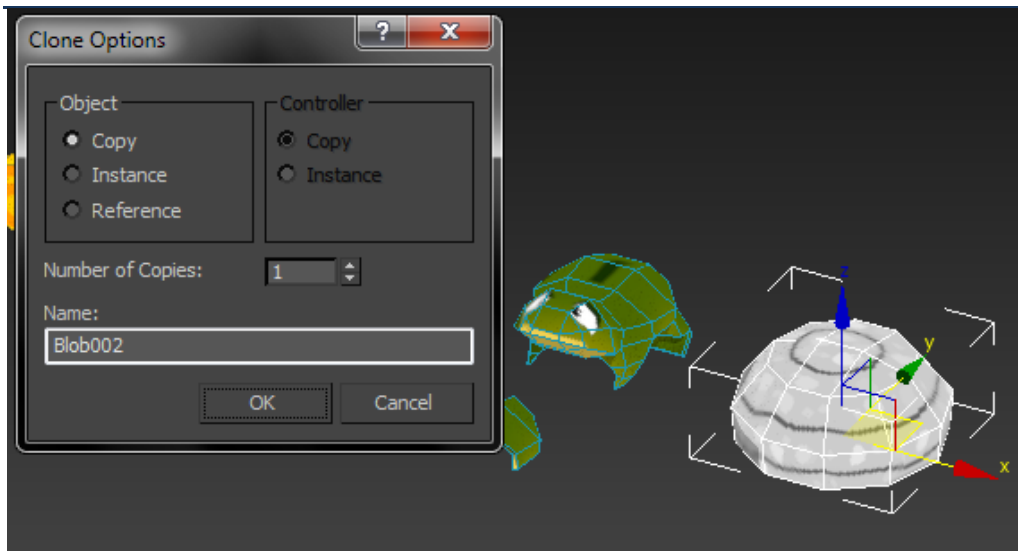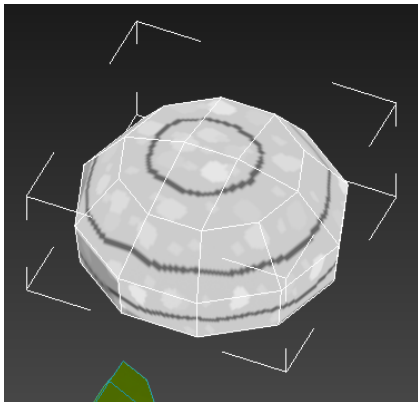
- Then I select the frog model and **Attach** to it the blob model. Click the frog, click attach, and then click the blob. Now the blob is part of the frog model.
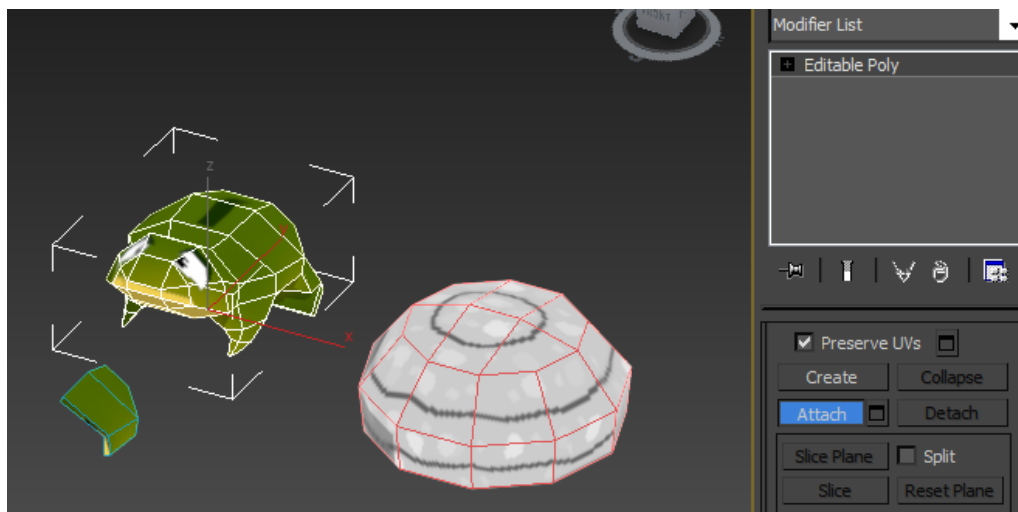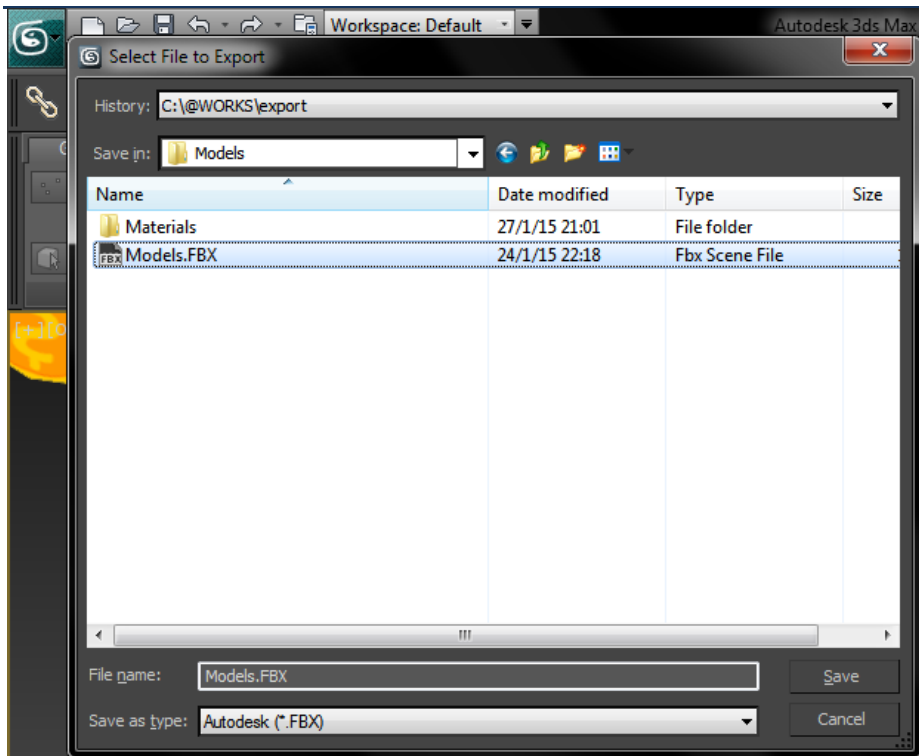

- Now switch to element mode, select and move the blob to stand right over the position of the frog.
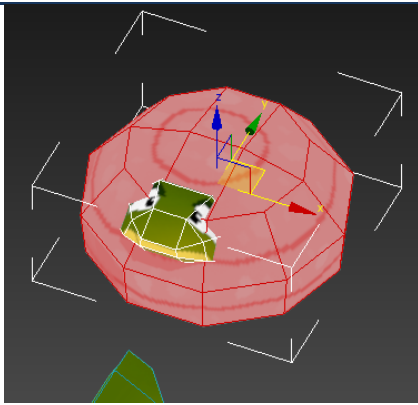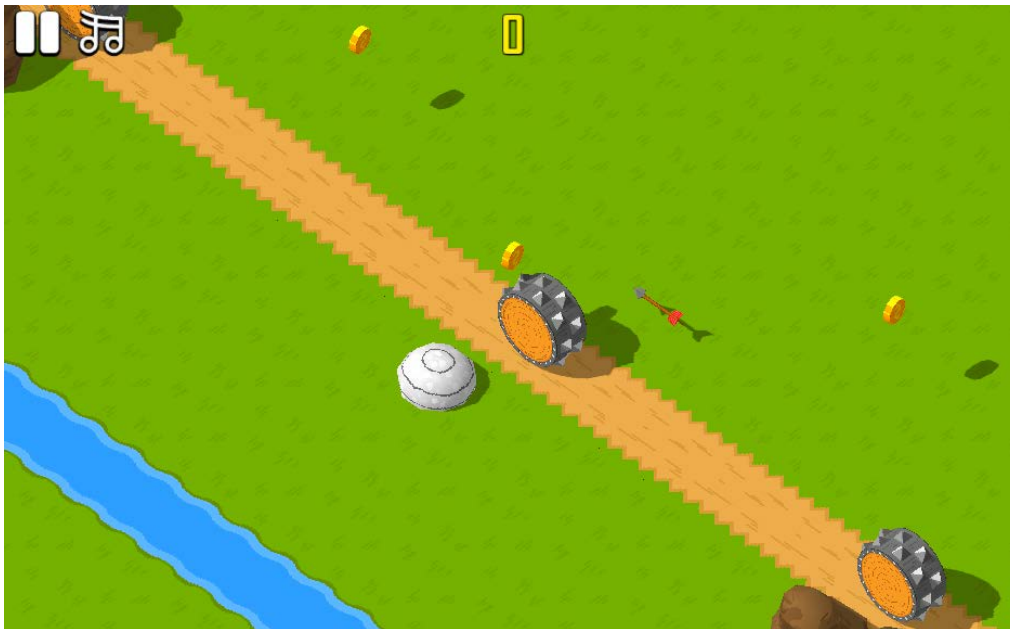
- Next we select the frog element and delete it. Now we are left only with the blob.

- To complete our work we will export our models into FBX for use in Unity.
- Click **Export** from the menu, go to the folder **Models** inside your game project and overwrite the file **Models.FBX**

- And here is the result:

You'll notice the "blob" player has the same attributes of the "frog" as before because we only replace the mesh and nothing else. This is the most straightforward method for reskins, but you can also of course create new models and use them in the game without having to replace a current model.

Creating a new model

To create an entirely new model, just model it in 3ds max inside **Models.MAX**,

give it a unique name and then export to **Models.FBX** again.
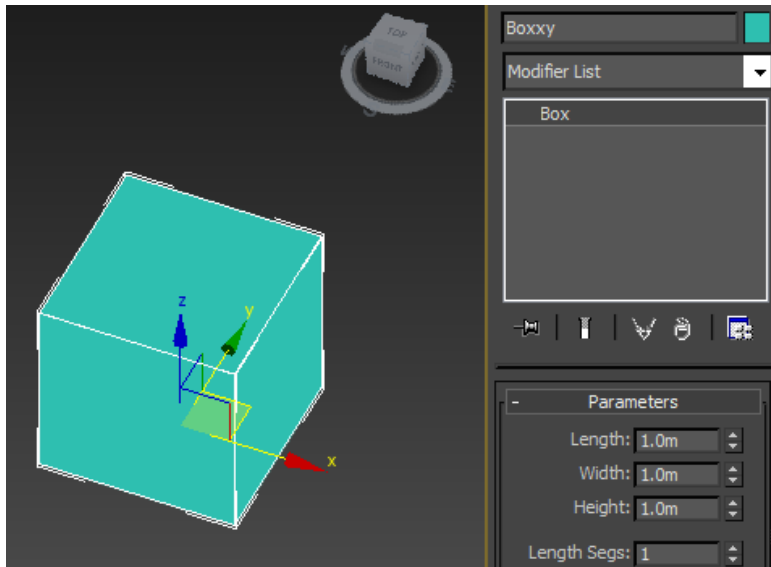
- Note: When creating a model in 3ds max for unity, make sure the Z (blue) axis is looking at the front and the Y (green) axis is looking up. This will align the model properly in unity.

- To do this, select your model and go to the Hierarchy pane, then choose Affect Pivot Only, and rotate the axis 90 degress so it faces the proper direction.
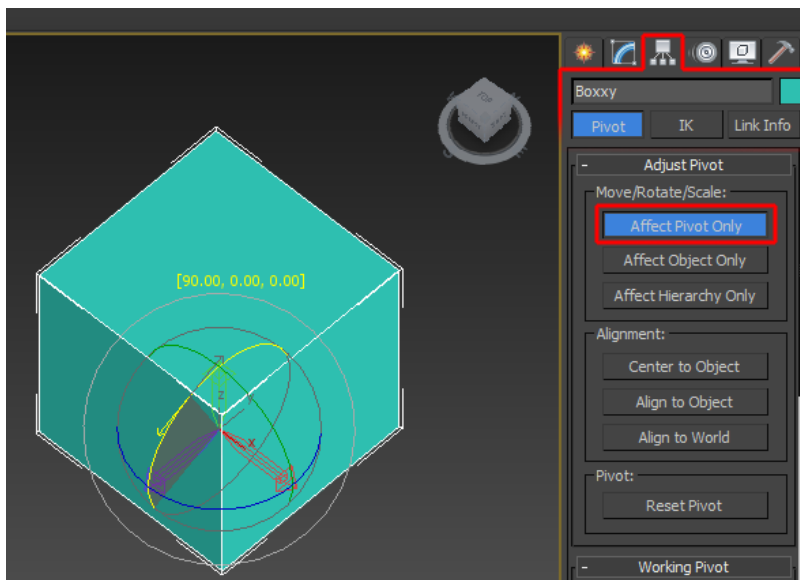
- Click **Export** from the menu, go to the folder **Models** inside your game

project and overwrite the file **Models.FBX**
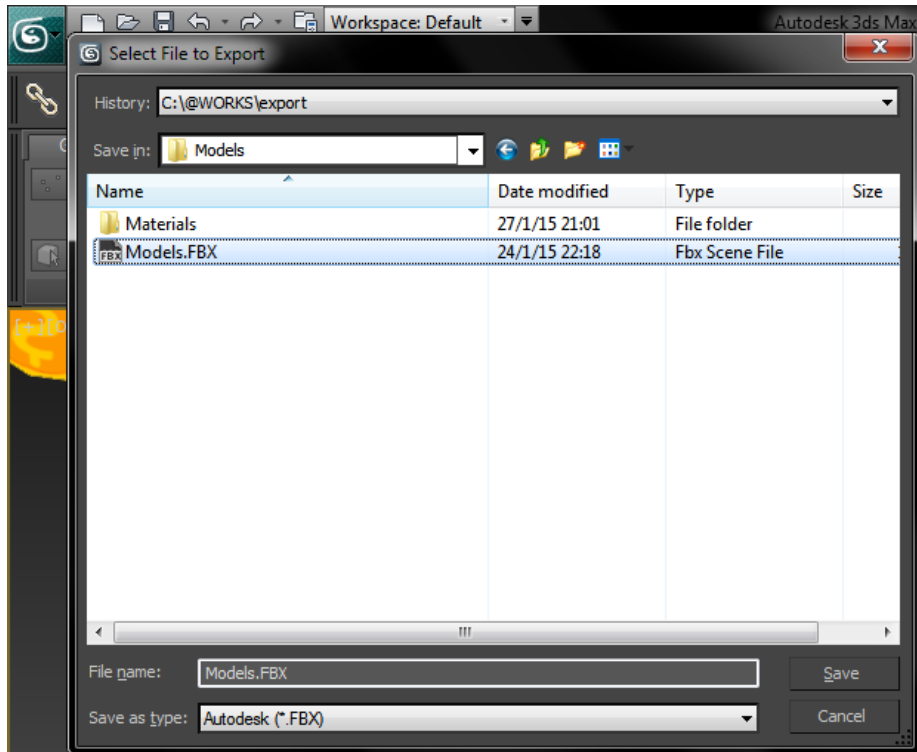
- Now when you go to Unity and check the file **Models.FBX** inside **Models** folder, you'll find a new model named **Boxxy** which you can use in your game!

Creating a new tile

Here's the quickest way to make your own quad:
- For our example I want to make a new tile with a brick texture. I keep all the textures in the same sprite sheet to keep size down.
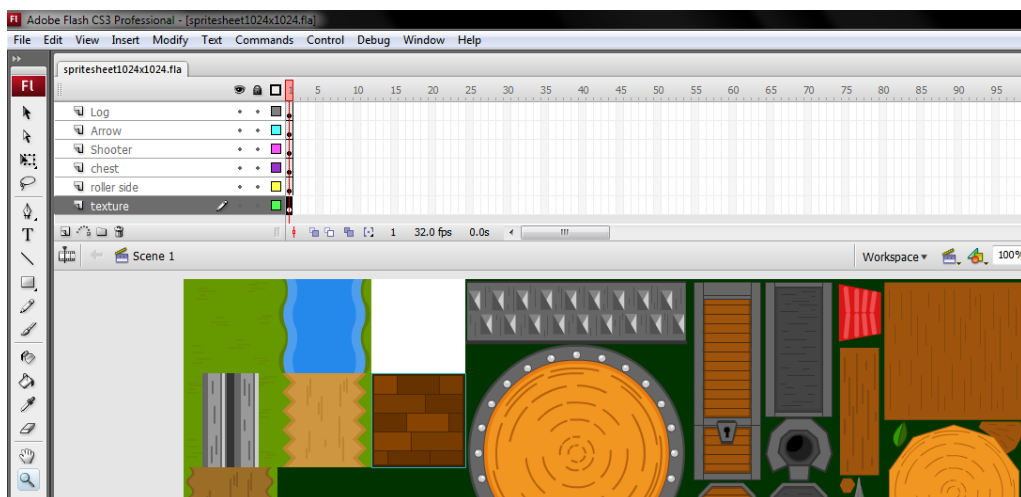
- Note: The package includes a MAX folder that contains all the models in the game. You can open this in 3DS Max 2014 or above. Open the file Models.max. This contains all the models in the game.

- In 3ds max I duplicate one of the tiles and rename it, then edit it UVWs, by just dragging them from the previous tile to this new one's texture. If you can't see the texture, choose it from the menu on the top right (where it says Map#0 (spritesheet1024x1024)).
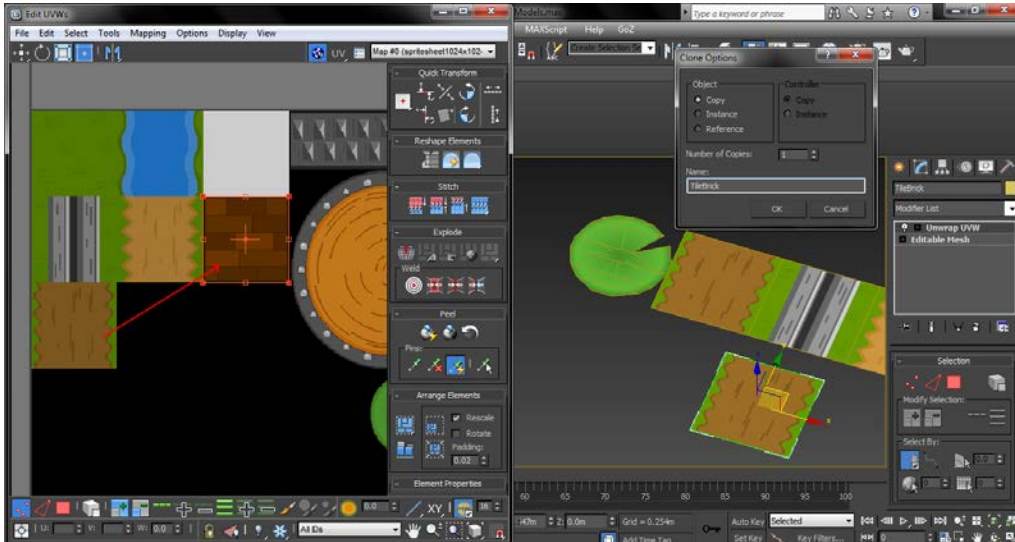
Now just export and overwrite the models.FBX file in your project.

Video tutorial: Adding new enemies

This 6 minute video shows the process of adding new enemies to the game, putting them in a lane, and then adding the lane to the lane list. For the

purpose of the tutorial we used car models from the package Zigzag Infinite Runner but you can use any FBX/OBJ car model you like.

Watch the video on youtube:

https://www.youtube.com/watch?v=fM3dWFoQ8p0

What we did:

1. Drag the car models and animation into our project.

2. Extract the cars we want to use, and assign the correct textures/materials to them.

3. Duplicate one of enemies and use it as the basis for our car enemy.

4. Refit the animation of the enemy using the car animation we imported.

5. Duplicate one of the lanes and use it as the car lane, adding all car variations we made to it. Set speed and number of cars in the lane.

6. Add the new lane we made to the list of lanes created in the game.

Video tutorial: Creating and adding a new player

This 1 hour video shows the process of creating and adding a new player to Road Crossing Game, starting with 3DS Max where we duplicate the doggy model and modify it into a new Goat model, while also editing the texture in Flash CS3 to fit the new character we created. Finally we import it into Unity,

add it to the list of players and to the list of characters in the shop, and also create several death effects for it.

Watch the video on youtube:

https://www.youtube.com/watch?v=CZjPgQdFOQI

What we did:

1. Duplicate doggy model and move it over, then apply a UVW Unwrap modifier to it, and move its UVW to an empty space on the spritesheet. Save the UVW info to a PNG so we can edit it in Flash.

2. Open the spritesheet in Flash, and drag the UVW PNG to it. Align the new UVW element into the rest of the UVW.

3. Create a new layer for the goat texture and draw it based on the UVW info we have. Then export the spritesheet to its original location in the project.

4. When we open 3DS Max again we can see the new texture applied to the duplicated model.

5. Now we can start to reshape it into a goat. Remove the ears and extrude horns from them, and also reshape head.

6. Since we created new polygons for the horns we need to edit the UVW again to unwrap and put the horn UVW in an empty space in the spritesheet so we can draw over it.

7. Same process is repeated for the legs as we want to give them more texture detail than the legs of the dog we duplicated from.

8. Keep alternating between shaping the model, editing the UVW info, and drawing the texture in Flash until we get the result we want.

9. Once we have the model ready we can export it as FBX to Unity.

10. In Unity we duplicate one of the player prefabs and replace the **Mesh** of the object **Part1** into our new model.

11. Add the new player to the list of players in the game controller.

12. In the Shop we duplicate one of the buttons and give it a new name. Then we add it to the RCGShop script. You can set the price of the character unlock here.

Important: Make sure you assign the correct index number in the OnClick() function of the button we created. Index 0 means the 1ˢᵗ character is unlocked, index 1 refers to character 2, etc
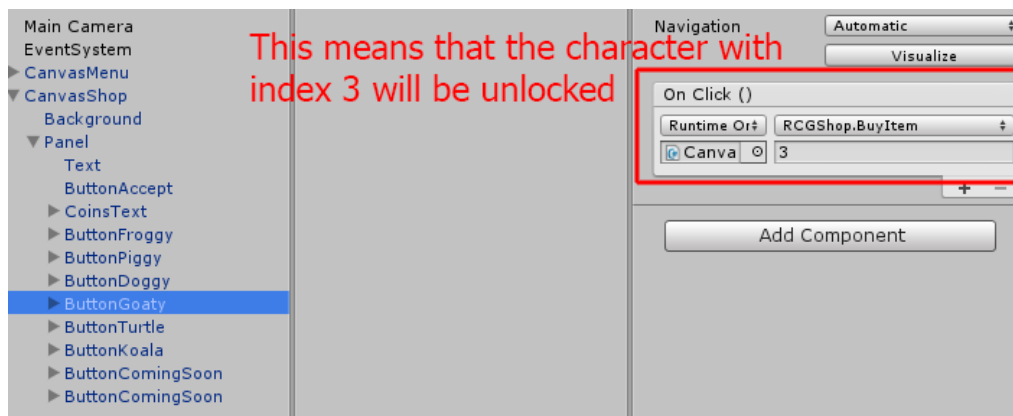
13. We also take a screenshot of the new character and add it to the icons in Flash. Then we assign this new icon to the new button we created.

14. Finally we duplicate some of the existing death effects for other characters and fit them to our character. Then we assign the new prefabs to the new character.

That's it!

UnityAds Integration (Unity 5.2 +)

Since Unity 5.2 UnityAds integration has been simplified, here's how you can have full screen video ads in your game.

This video shows a quick process of integrating UnityAds into your project. In the example we used one of my templates, but it works on all my other templates too.

https://www.youtube.com/watch?v=EQNTgfV35DU

Here is what we did in the process:

1. Sign in to your Unity account in order to allow Unity Services such as UnityAds to be activated.

2. Open Build Settings and switch the platform to one of the supported ones (iOS, Android).

3. Download Puppeteer's UnityAds package from: puppeteerinteractive.com/freebies/PUPUnityAds.unitypackage

4. Drag the downloaded package into your Unity project, and import it. This UnityAds prefab can be used to display ads every several minutes.

5. Drag the prefab into any scene where you want ads to be shown. Make sure to save changes.

6. The time check is shared between all prefabs in all scenes, so you will never show too many ads.

7. The final step is to activate UnityAds services and get your unique project ID.

8. Open the services window and choose your organization, then click create.

9. Choose UnityAds from the list and turn it On.

10. Choose age group for your project (Will affect the nature of ads shown), and save changes.

11. While working on your project keep Test Mode activated. But when you are ready to release the final project, switch Test Mode off.

12. That's it! Now when you start the game, an ad will be shown after 3 minutes. The ad will never appear during gameplay or post-game

screen. Instead, it will wait until the next level load (restart, main menu, etc) and then show the ad.

Before releasing a game, make sure you uncheck **Enable Test Mode.**

For more info about integrating UnityAds read this:

http://unityads.unity3d.com/help/monetization/integration-guide-unity

Integrating UnityAds into your project (Unity 4)

Adding support for UnityAds into your current project is simple and shouldn't take you more than 5 minutes. Let's start:

First we need to create our game entry on the UnityAds website. Go to https://unity3d.com/services/ads and create a new game. If you already have your app set and your GameID noted, just skip this part and go straight to

importing the UnityAds package into the game.
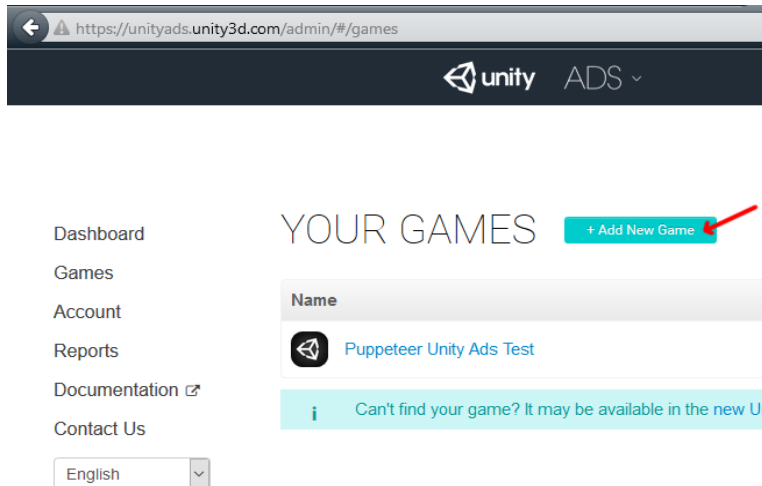
Now we need to choose the platform. The process is similar for both iOS and Android but for the purpose of this tutorial we'll choose Android. If you have an app on Android, enter its name to find it. If you don't have an app, click below where the red arrow points in order to enter the name of the app that has not been added to the store yet. This way you can test the app before it goes live.

After you created your app in the website, make note of the Game ID that appears. This will be used to link the ads to your app.
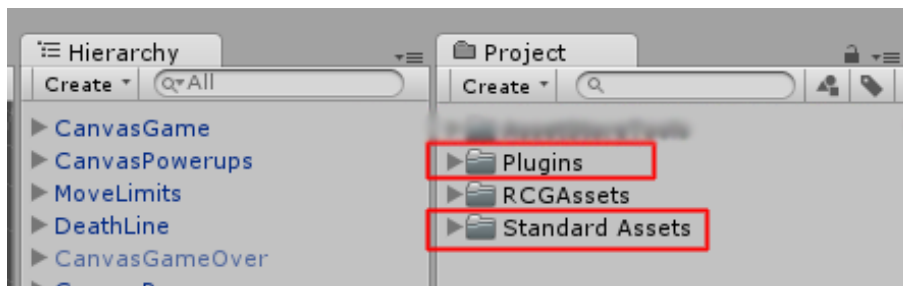
In Unity Editor

Now we need to import the UnityAds package. Open the Unity Asset Store and download the UnityAds package. Import it into your project.

(https://www.assetstore.unity3d.com/en/#!/content/21027)

After import you should have two additional folders in your project.

Now we need to bring in the code that integrates the ads into our game. Click on the **PuppeteerUnityAds** package in your project to import it into the game, or choose **Assets > Import Package > Custom Package…** from the top menu and navigate to the **PuppeteerUnityAds** package in your project to import it.
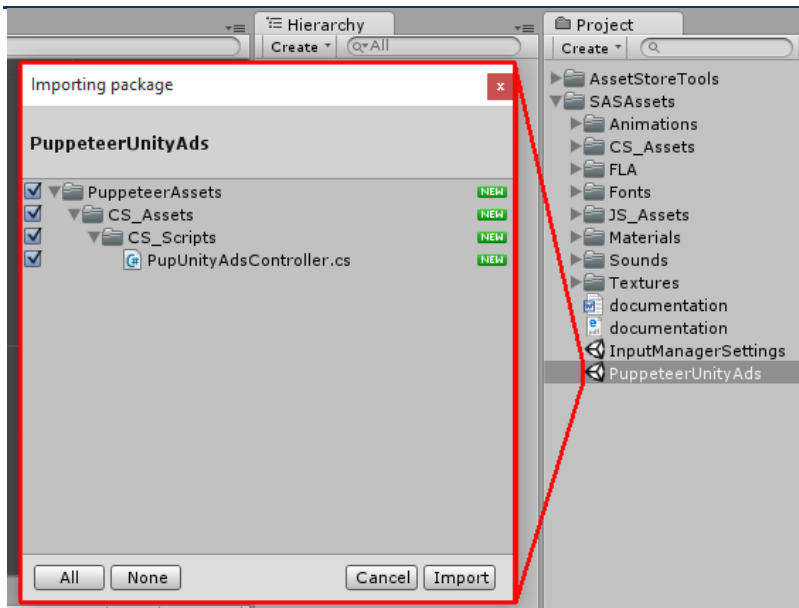
PupUnityAdsController.cs is the main script that links your app to the unityads system. Drag it into your game controller. Now when you look at it you see you can set the GameID of your app, and how often the ads appear. The ad is checked when the level is loaded. "**Show Ad Every**" decides how many times the level needs to be loaded before an ad appears.
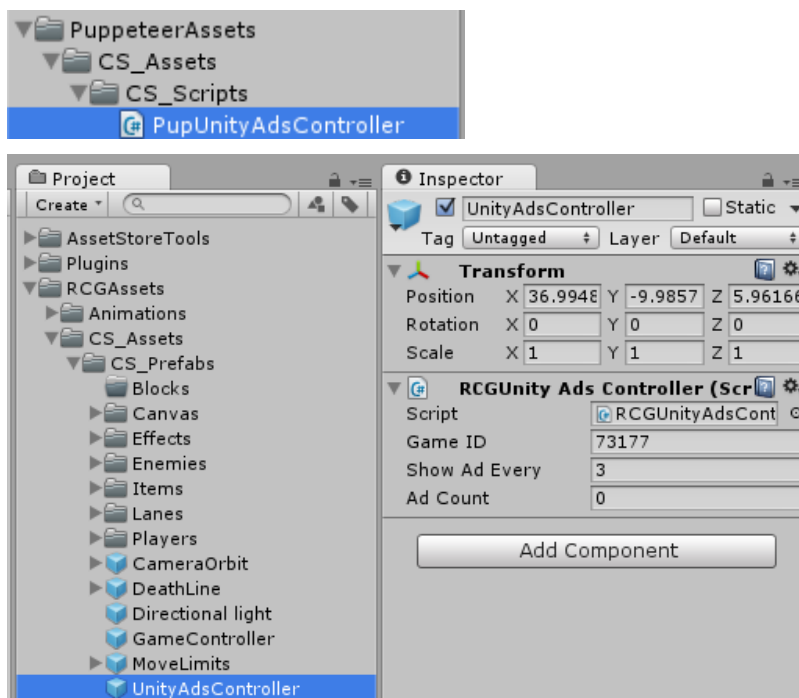
In order to test the ads, we need to switch to the Android platform.
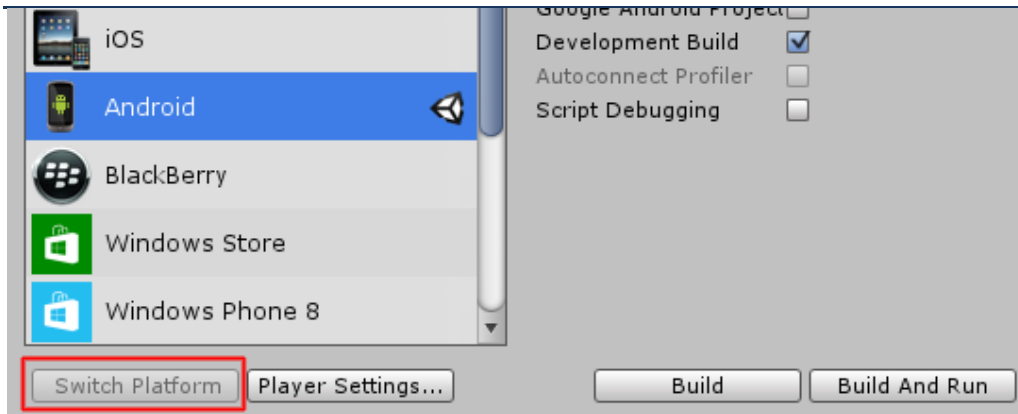
That's it! Now start a level and restart it 3 times, then you should see a blue screen showing the ad system has been activated correctly. If you build to Android you should see an actual video ad appear after 3 level loads.

Does this package work on mobile?

Yes, this package has been successfully tested on both Android and iOS devices. The scripts for each lock type include controls for mobile that are detected automatically based on the platform it's built on.

My sprites are not showing on iOS

Sprite-based textures made with the new Unity 4.3 can sometimes disappear when working on the iOS platform.

You can notice this by opening a scene playing it. When you switch from your current platform to the iOS platform the sprite textures become invisible.

To solve this we must change the texture compression format for iOS. Follow these steps:

1. Click on a texture in the project view.

2. Click on the override for iphone button on the right side.
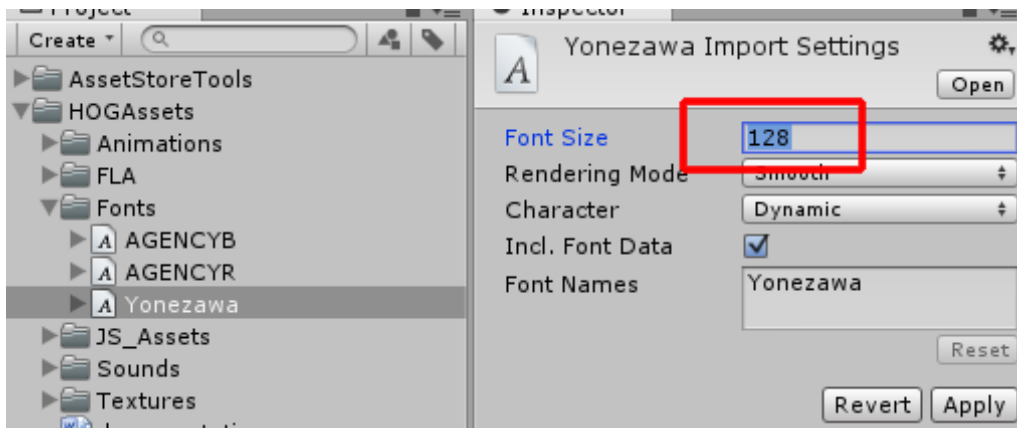
3. Change the format to 16bit.

4. Click Apply.

How to change font in the game?

To change a font in the game do the following:

Find a font you like and drag the .ttf file over to the Fonts folder in your game.
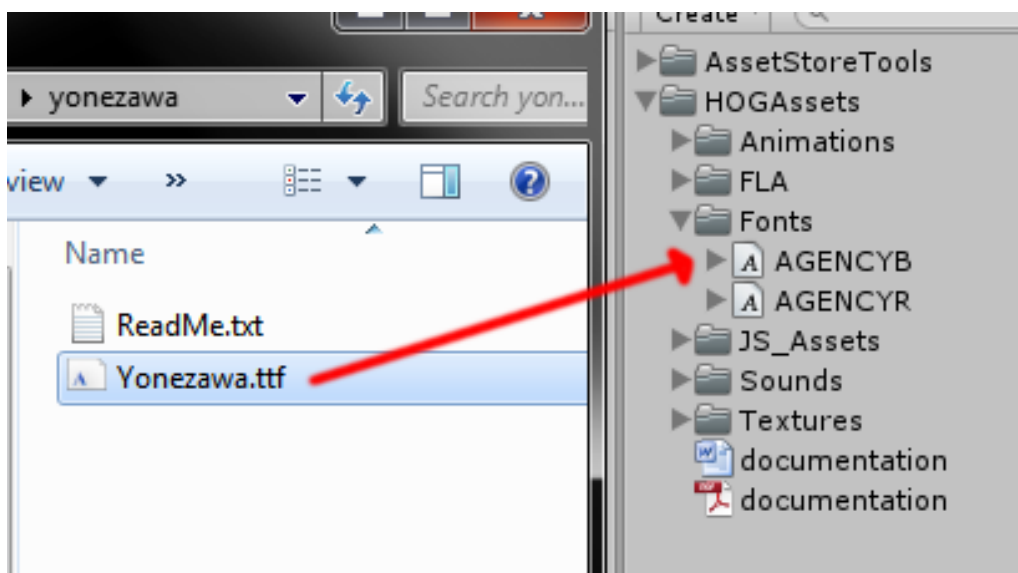
Click on the font you added and edit its attributes. I personally set all my fonts to a high number (and then scale the text object down) so that they look crisper in-game.


Select any text object in the game and change its font to the new font you have. Sometimes the text might disappear, but it's normal. Just write something in the text box above and it will refresh. Also, make sure you change the text for the shadow; you can select both the main text and its shadow and edit them together.
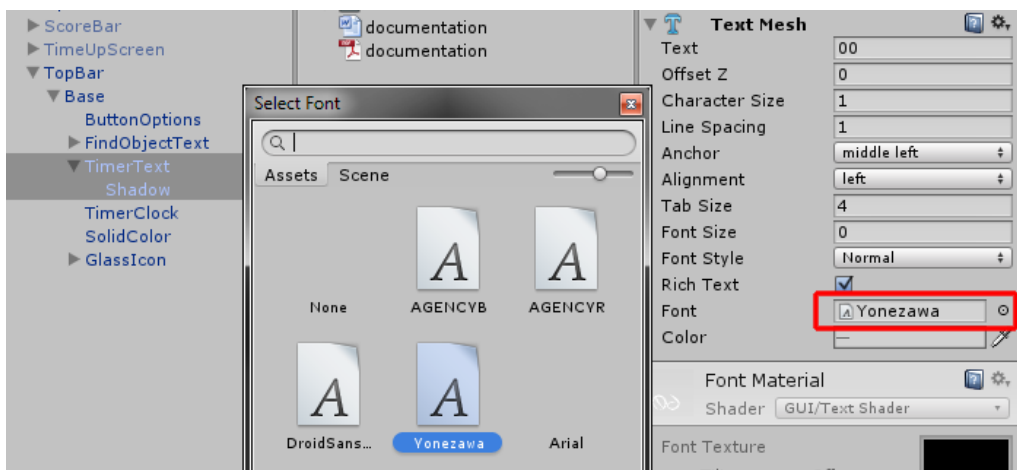
More games by Puppeteer

Click here to see the full catalogue of Asset Store files!
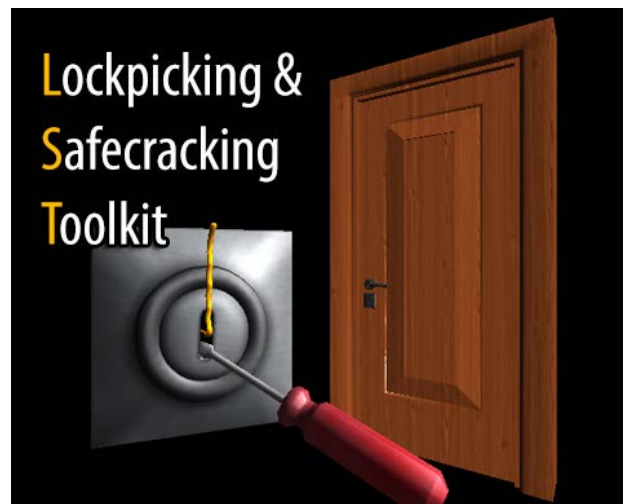
It is highly advised, whether you are a designer or a developer to look further into the code and customize it to your pleasing. See what can be improved upon or changed to make this file work better and faster. Don't hesitate to send me suggestions and feedback to puppeteerint@gmail.com

Follow me on twitter for updates and freebies!

Good luck with your modifications!