Adventucation README

James Isnor, October 2/2015

Note: This README was made with Unity 5.1 & 5.2 in mind.

This Package

This package is designed for teachers who want to make their own simple 2D educational platformer. It is the intention of the authors to create a platformer framework from which to work. It includes the materials for a player (a monkey who can walk, jump, run, climb and move through platforms), animations, platforms, ground, collectibles, and doors (from which to advance to the next level) for **six** different environments already completely done. All the teacher needs to do is make a level with the assets we've provided and determine the objective of that level.

Game Summary

The package containing this document is called Adventucation, which is a 2D platformer for young students (ages 4-6 was the original plan). The idea is for kids to have a fun game that they enjoy playing which contains educational components.

The original idea is based around the concept of "indirect learning", whereby the player has to use their skills/knowledge from school to advance in the game, but it is done in such a way that the player does not realize they are being tested. Often students get bored with repetitive excersizes and tests in the classroom (or for homework) and this leads to disliking school and, in the worst case, disliking learning as a whole.

The game is, like many, a "collectithon", in which the player has to collect certain objects to progress through the current level and on to the next. The objects they collect all relate to each other as well as the "end of the level objective". For example, if the end-of-level-objective, displayed on the canvas (a sort of overlay on the player's screen, representing the HUD), is a door made up of four red squares, then the player must collect four red squares. Not 3 red squares, not 4 blue squares and not 2 purple, 1 green and 3 red squares.

The game also includes coconuts, which are collectibles for the player to pick up that have no bearing on the objective itself. They are there for a few reasons:

- So that the player has more to do
- Collecting things is often fun for young students
- Collecting all of the coconuts will require the player to explore the entire level and is thus more challenging than just collecting the level objectives
- Collecting all of the coconuts for a level rewards the player (in whatever way you want it to). This could be done with achievements, leaderboards or in-game incentives (non of which are featured in this demo)

Game Components

Player

Located in the *prefabs* folder, "testPlayer" is the name of the GameObject that the user controls when playing the game. The player can walk left and right, walk up inclines, move up/down through platforms (if the platform allows it), jump and climb ladders. It has 4 scripts:

1. Controller2D – handles the player's movement
2. Player – Deals with the player's movement upon receiving input: movement speed, jump height, climbing ladders, player direction and animations
3. PlayerHealth – Handles events in which the player loses health or dies
4. CollectionController – Handles picking up collectible items

In order for the player to be able to drop down and jump up through platforms, the player's RigidBody2D must have its CollisionDetection set to Discrete.

Collectibles

Collectibles are any GameObjects that are on the "Collectible" layer. When they are touched by the player, they are collected and added to the inventory. The inventory is implemented as a singleton with a dictionary (more on this in the InventoryManager script & doc). The inventory is located on the Canvas prefab and is present in every scene. To the player, it is docked to the left of the game screen.

Platforms & Ground

There is a prefab for a section (or "bar") of ground for each environment. It contains a parent object with several children, all of which are sprites. The parent object contains a box collider that surrounds the children sprites, as opposed to each individual block of ground having its own collider. The parent object is on the *Ground* layer. Please keep in mind:

- Drop-through platforms must belong to the "Platform" layer
- Ground (floor) objects must belong to the "Ground" layer

Doors

Each door must have a DoorController script attached to it. In the Inspector, one can add objects to the DoorController's array, *Collectibles Required*, which specifies the objects that the player must have picked up in order to advance through the current door to the next level. For example, if a particular door's *Collectibles Required* array had three elements, 1 pink triangle, 1 blue triangle and 1 green triangle, the door would not open unless the player had those three objects, and only those three.

Doors *must* have the tag "Door" and *must* be on the "Door" layer. Also, in accordance with the DoorController script, doors must have both *Animator* and *Collider2D* (any Collider2D) components. The Collider2D, whichever one suits a particular door's shape, must be added before the DoorController script.

Hazards

Hazards are GameObjects who hurt the player, briefly stunning them and dealing damage. They are any GameObject on the Hazard layer (if you wish to make an object that hurts the player, it must be

on the Hazard layer). GameObjects on the OutOfBounds layer are similar, except they will kill the player instantly (say, if the player falls into a pit/off of a platform into the abyss).

Generally, unless your level contains no possible way for the player to "fall off the edge" (not likely), you will need at least one object (that has a collider with isTrigger checked) on the OutOfBounds layer for the player to fall into and hit, killing them. There is a prefab created for this kind of object in Resources/Prefabs/Triggers. If your player is allowed to fall off of the edge and not die, they will continue falling forever, never respawning.

Canvas

The Canvas is how we setup and display the HUD (heads-up display). On the HUD, from left to right, is:

- The Inventory, pinned to the very left of the screen, implemented as an empty parent holding 6 transparent images
- The load and save buttons, underneath the inventory
- The coconut counter – displays the number of coconuts collected in the current level
- The Pineapple counter – displays the number of Pineapples collected in the current level
- The Level Objective Indicator – displays the objective for this level (tells the player what they need to collect to advance to the next level)
- The player's health – three images of the monkey, who the player plays as, each image representing one unit of health

There is a prefab for the Canvas. In each scene, there should be **one and only one** canvas. You can put a canvas in your scene by opening the Project panel, navigating to Assets>Resources>Prefabs and dragging the "Canvas" object (it looks like a blue cube) into the scene. Nothing else needs to be done. This will create a large box at <0,0,0> in world space that can largely be ignore. We recommend building your level below this large box (the canvas).

Music

There is "level music" playing in the background of each level. In addition, many enemies/objects make sounds; these are managed by the *PlaySoundWhenNear* script. If you wish for the player to hear a particular sound when they are near a particular GameObject in the scene, *PlaySoundWhenNear* may work for you, and you can simply add it as a component.

Player Settings

Managed as a singleton, the Settings class gives us a way to save user preferences. This is used only once, in the title scene, and serves only to save the user's language choice (French or English) and current level. It is later used, along with SerializableSettings, in the LoadAndSave script so that the player can quit the game whenever they want and come back to the same level/language (it does not save the items or location of the player by default).

Creating Your Own Scene

Unity Basics

Game Objects

Before we talk about creating your own level with the Adventucation Unity package, we will talk about the basics of Unity. If you are already familiar with Unity, we invite you to skip this and move onto the next section.

When one first opens Unity (or creates a new scene, either with File>New Scene or ctrl/cmd + N), there isn't much; a blank "scene" with a *Main Camera* object (and a *Lighting* object if your project is 3D – for this package, it should not be. If it is, remove the lighting object and click the 2D button, which should be located in a pane underneath the "Scene" tab).

Find the tab that says Hierarchy. In this pane, you should see all of the GameObjects in your scene. Again, the only thing there right now ought to be a *Main Camera* object. Ignore it. If you right click anywhere in the pane (not on another object, though), you'll see a menu. With it, you can create new objects by clicking "Create New" (the "Create" drop-down menu underneath the *Hierarchy* tab can also do this). This will make a new, empty GameObject, which we can examine with the *Inspector* pane. Go ahead and make a new object and then view its components in the *Inspector*. You should see only one component: *Transform*

The *Transform* of an object is simply 3 vectors: the position, in 3D (the z is 0 for all 2D objects), the rotation, and the scale. Adjusting the position will change the location of the object in the scene, adjusting the rotation will rotate it and adjusting the scale will make it bigger or smaller (if the scale is less than 0, it becomes smaller, greater than 0 it becomes larger).

By now you've likely stopped reading this doc and begun playing with Unity. Either way, you may have noticed the *Add Component* button in the Inspector. With your new, blank object selected, go ahead and add a BoxCollider2D. Colliders are how Unity handles GameObjects touching one another. What one object does when it touches another is handled with that object's scripts. More on this later.

Note: be sure you've added a Collider2D and *not* a Collider. For example, there is both a BoxCollider2D *and* a BoxCollider component. The “2D” suffix indicates it is for 2D GameObjects, the lack of one indicates it is for "regular" 3D objects.

You may be thinking "why can't I see my object in the scene?" With the collider added, you should see a green outline of the object, which represents the dimensions of the collider. Blank objects don't have anything to show, or *render*, and so your object seems invisible. There are several different types of renderer components in Unity, but the one we use in 2D games is called *SpriteRenderer*, which render *sprites*. A *sprite* is what we call an image in a 2D game (as opposed to 3D games, which use *models*). If you'd like, you can add a *SpriteRenderer* to your game object the same way you added your collider: with the Add Component button.

The GameObject should still be invisible, but if you drag any image into the "sprite" box in the SpriteRenderer component, that image should now represent your GameObject (be it a player, a box, an enemy, etc). For now, we will ignore the rest of the SpriteRenderer. Find the *Project* panel. In the Sprites folder, there is an image called *X.png*. Click it, and you should see a bunch of information about the image in the Inspector panel. Notice the sprite mode is "single". For now, ignore the rest.

Going back to the Hierarchy panel, click the GameObject you created so that you can see the SpriteRenderer component. Now, from the Project panel, drag the X.png into the SpriteRenderer's "sprite" field. In the scene view, your GameObject should now be displayed as an X.

This is how easy it is to create GameObjects in Unity. For an even simpler approach, if you already know the image you want to represent your sprite, you can simply drag a single image onto the scene, and Unity will create the GameObject *and* SpriteRenderer for you! Go ahead and try:

1. Find the X.png again: In the *Project* panel, go to Assets>Resources>Sprites>X.png
2. Simply drag the image from the Project panel into the scene
3. There should now be a new GameObject that is represented by an X. Looking in the Hierarchy for the scene, we can see a new GameObject called X

GameObjects can also have children GameObjects. To pair two GameObjects with this relationship, simply drag the intended "child" object onto the intended "parent" object *in the hierarchy of the scene*. The parent will now have a small arrow pointing towards its name in the hierarchy panel. Clicking this arrow expands the object, showing all of its children.

Go ahead and play with some more components. Remember, the Unity Manual and Scripting API are your best friends. The manual is the high-level overview of what components do and how they interact, whereas the scripting API gives a way to understand writing scripts with a particular component. If you are not comfortable writing code, that is fine. This package contains many prefabs – prefabricated GameObjects to be used in the scene – that already have all of the components & scripts created and attached to them. All you need to do is drag them into a scene and create a level.

Scripts

Scripts are what make the GameObjects do what they do. If you think that you can create a game without any packages or programming (that is, with just Unity and some sprites), I'm sorry to tell you that simply isn't possible. Fortunately for you, you've downloaded our assets package, which has several scripts written for you, tailored for educational game design! For example, there's a script that lets the player pick up collectible objects, one that lets the user move the player around with the arrow keys, one that manages the player's health, etc.

You attached scripts to GameObjects like any other component (they *are* components): the Add Component button. Scripts can be written in two languages: C# and JavaScript (which is *not* Java). For this package, we've written everything in C#, but if you are making your own scripts, JavaScript is fine and will work together with the scripts in this package. Without a focused discussion on programming, which is not what this document is for, we cannot really talk about how to compose your own scripts, and if you're already familiar with programming, the API is all you will need.

Building

For the game to function as a standalone executable (for desktops), an app (for mobile) or to be played in a webplayer, the game assets (GameObjects, scripts and scenes) have to be compiled together in a process known as *building*. It's a pretty simple process:

1. In Unity, open up the File menu at the top of the window and select Build Settings (or ctrl/cmd + shift + b). Here, you can chose the scenes that you want to be in the build (if your scene is not listed, then close the build settings, open up the scene you wish to add, open the build settings back up and click the "Add Current" button).
2. Choose the platform to deploy to. On the left, there is a list of platforms Unity can create executables for. For now, just build for the PC, "Mac and Linux Standalone". After a few minutes, Unity will have created an executable and you can play and share your game!

Creating a Level

We hope that creating a level with the assets we've provided is fairly simple. To start, open up a new

project (or new scene in your existing project). If you haven't already, you can import our package with Unity's menu: at the top, click the Assets drop-down menu, then select import package>custom package. Browse to the folder containing the Adventucation package and select it. Alternatively, if you have the project that you wish to import our package into open, you can simply double click the Unity Package File (wherever you saved it on your disk) and Unity will open it and attempt to import into your current project.

Once you have our assets imported into Unity, creating a level isn't difficult (though good level design is!) Check out the Prefabs folder. In here is everything you'll need to create an Adventucation level. If you want a reference or place to start, we have added some demo scenes for you, as well as a base template scene called "_blank". We recommend opening up _blank **and immediately saving it as a new scene** with whatever name you want to call your level. You can do this with the file menu. Use *Save As*, and not *Save*. Once you've created the new scene (the window should be titled something like, "Unity Personal (64 Bit) – *NewSceneName* – ProjectName – build_platform"), you're ready to start building your first Adventucation level!

You will notice the template has broken down the level into several different GameObjects, many of which have a small arrow pointing towards their name. This means that they are *parent* GameObjects; clicking them will expand them, showing all of their *children* (who are also GameObjects). In order to keep things organized, we have a parent for all of the platforms, ground, background images, hazards, and collectibles. This means that any time we create a new object in the scene, we place it wherever we want it to be on the level, but in the hierarchy view, it is placed appropriately (e.g. if we're placing 6 collectibles, they are all children of the Collectible GameObject). This may seem strange, because the parent objects do not have any components and, for intensive purposes, don't exist in the game, but this allows us to keep everything clean.

For more support, one can e-mail me at jisnor@stellarls.com