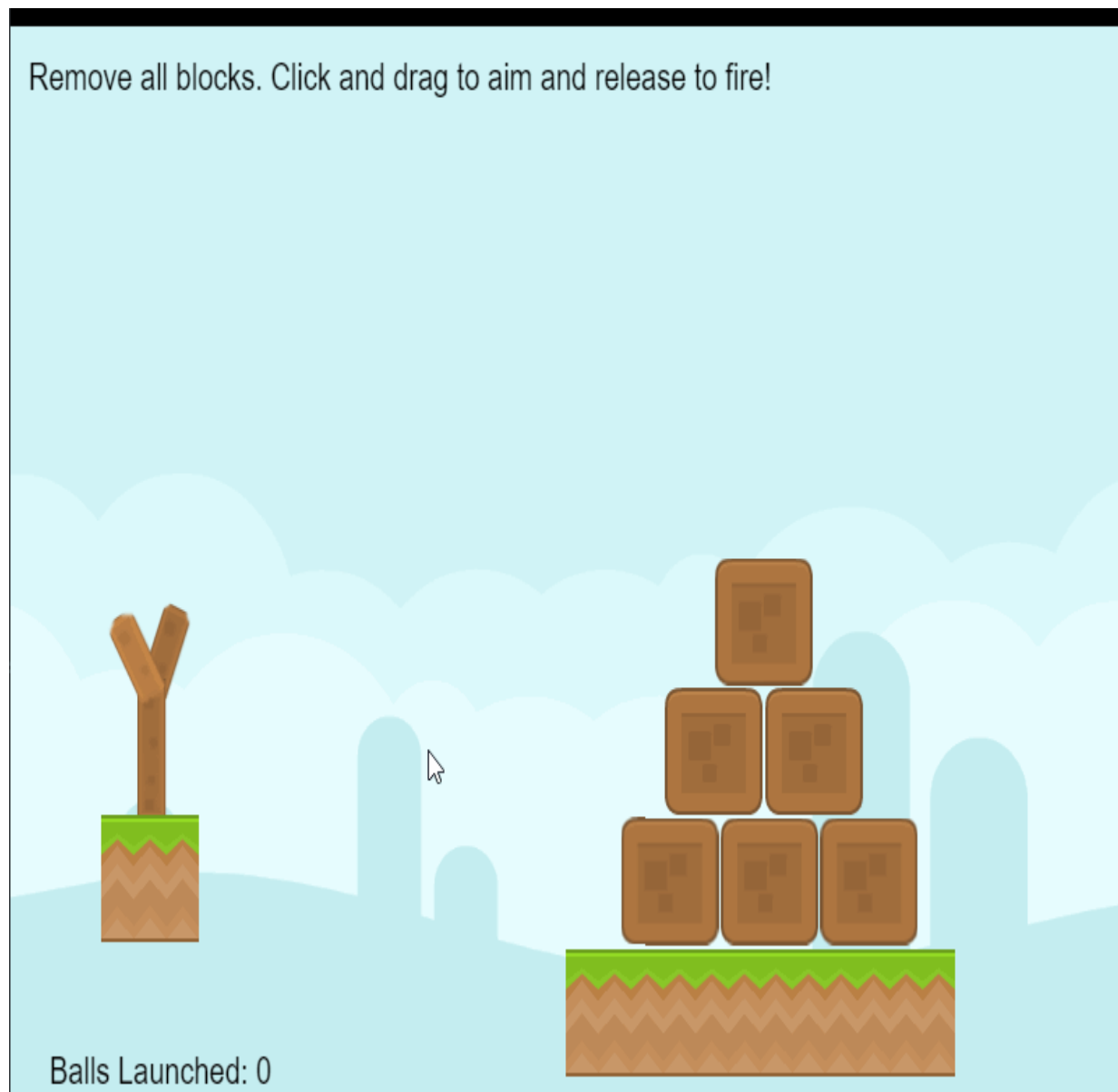


GDevelop Physics Launch Tutorial - Part 1

Posted on: [October 14, 2021](#) Last updated on: October 14, 2021

Categorized in: [GDevelop Tutorials](#) Written by: [TwistedTwigleg](#)

This tutorial will show you how to make a physics game project similar to that of Angry Birds. At the end of the tutorial, you'll have a little project that has everything you need to extend and create the next hit mobile game. Here's how the finished project will look:



The finished prototype project this tutorial series is based on! Download link included

If you want to peek at the finished project, you can find the download for the finished prototype this tutorial series is based on right here: [Download Link](#). We will be making almost every aspect of the prototype throughout this tutorial series!

Tutorial Introduction - What is GDevelop?

GDevelop is a game engine that is designed to be no code friendly, making it easy for beginners to game development to jump into making their own games with necessarily having to write any code. GDevelop has a powerful and intuitive system for making games without relying on code, instead using visual programming with “actions” and “conditions”, which are similar to the programming concepts with the same name.

GDevelop is a 2D focused game engine and can export to HTML5, Android, iOS, and desktop platforms. In addition to the visual programming, you can also write code in Javascript. GDevelop includes a downloadable editor and an online version you can use as well, making it quite flexible and easy to get going.

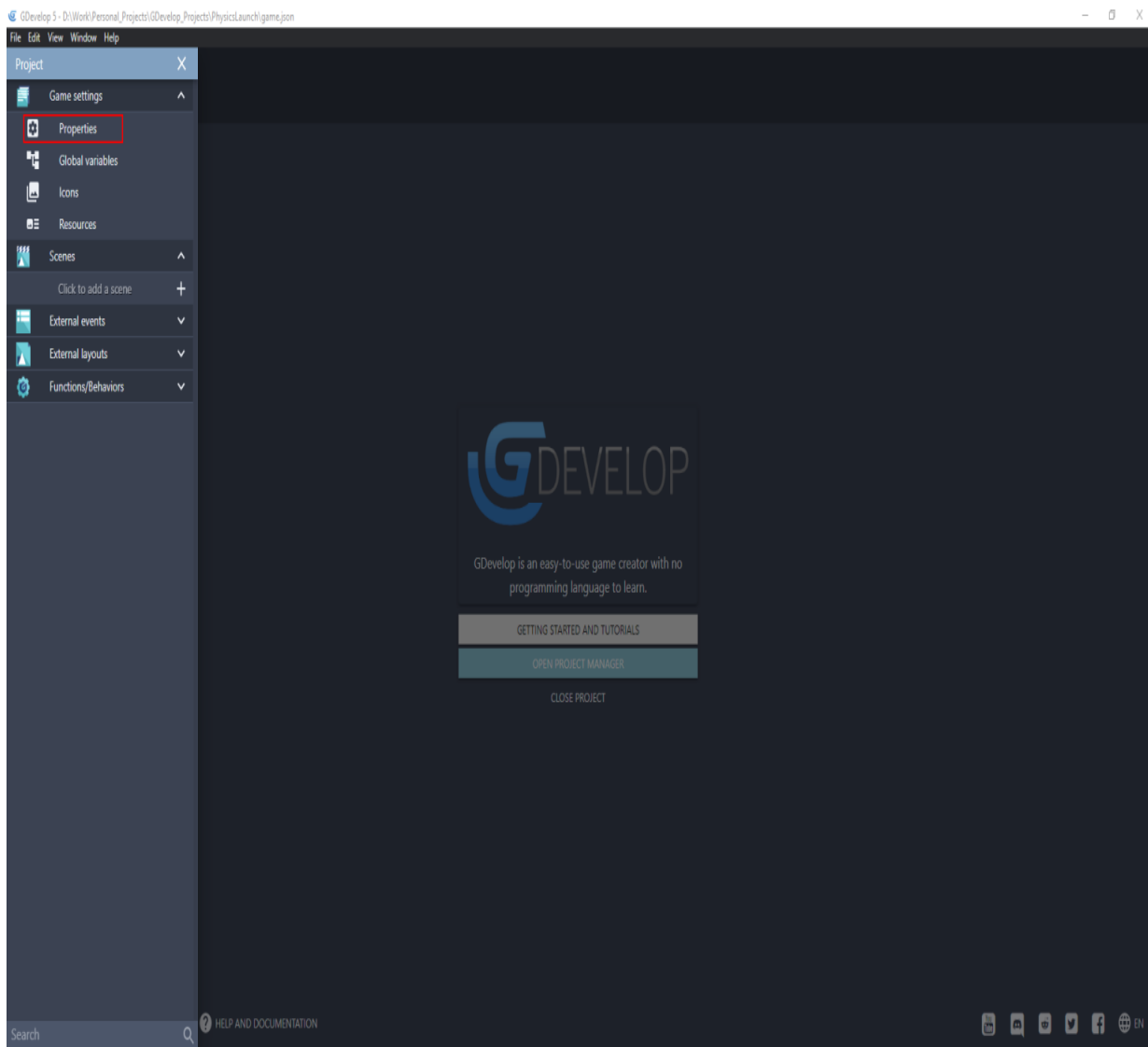
Today, we’ll be making a physics game where you need to use a sling shot to launch balls into wooden crates. Once all the wooden crates are off screen, the game will progress to the next level and this process repeats until you reach the end of the game. Let’s jump right in!

Part 1 - setup

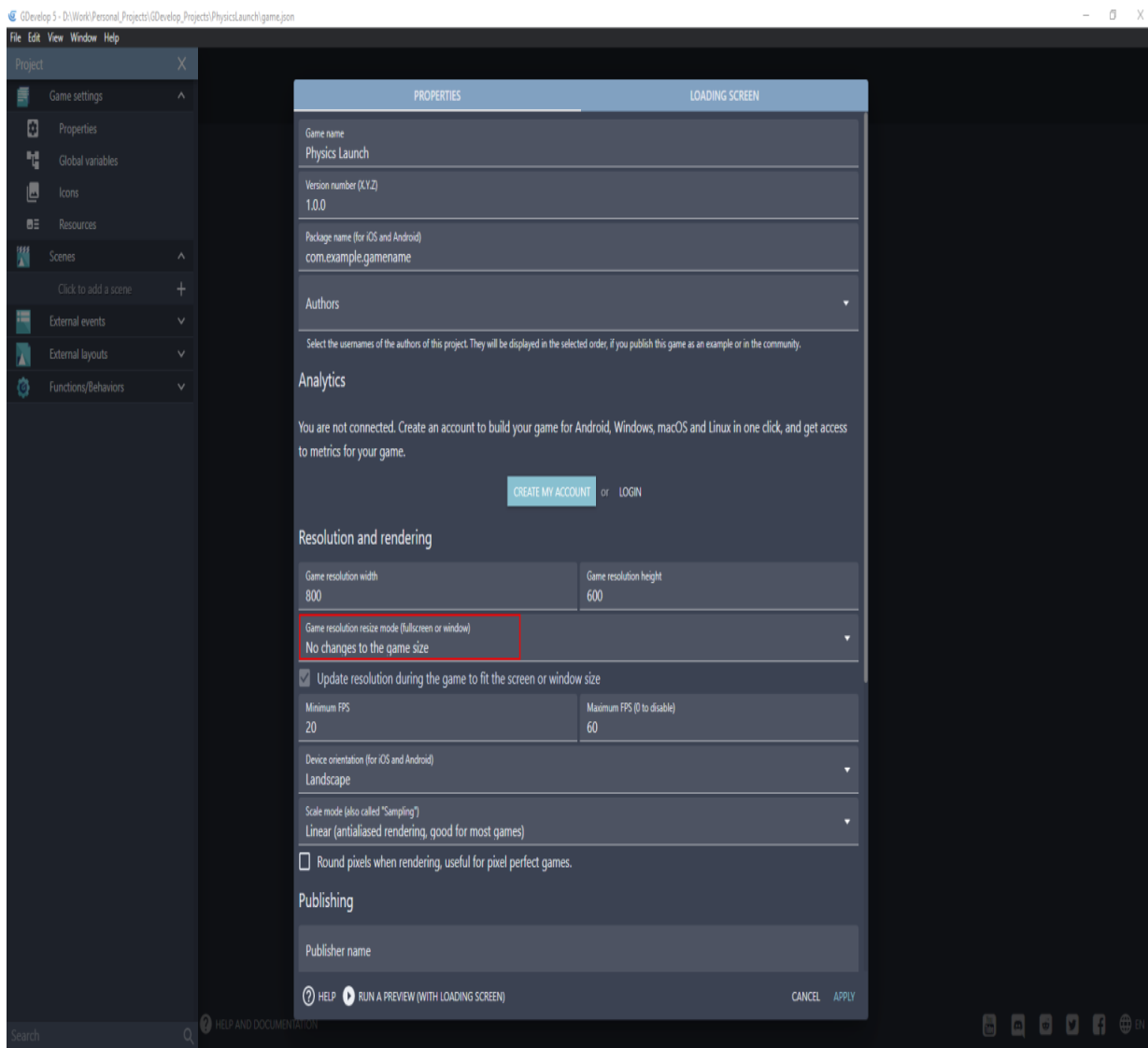
I will be using the downloadable version of GDevelop 5 for this tutorial, so if you are using a different version or platform (like online) you may need to make minor alterations. Additionally, you can download all the starter assets needed for this tutorial right here: [Starter Assets](#).

In this tutorial we will be using some lovely assets from [Kenney](#), who publishes CC0 assets on his website and I would highly recommend taking a look if you want to find some quality assets for your games and prototypes.

First, let’s open GDevelop and make a new project. I will be naming the project and folder “PhysicsLaunch” but feel free to call it whatever you want. You can change the name of the project anytime by opening the project view and going to the “Game Settings”.

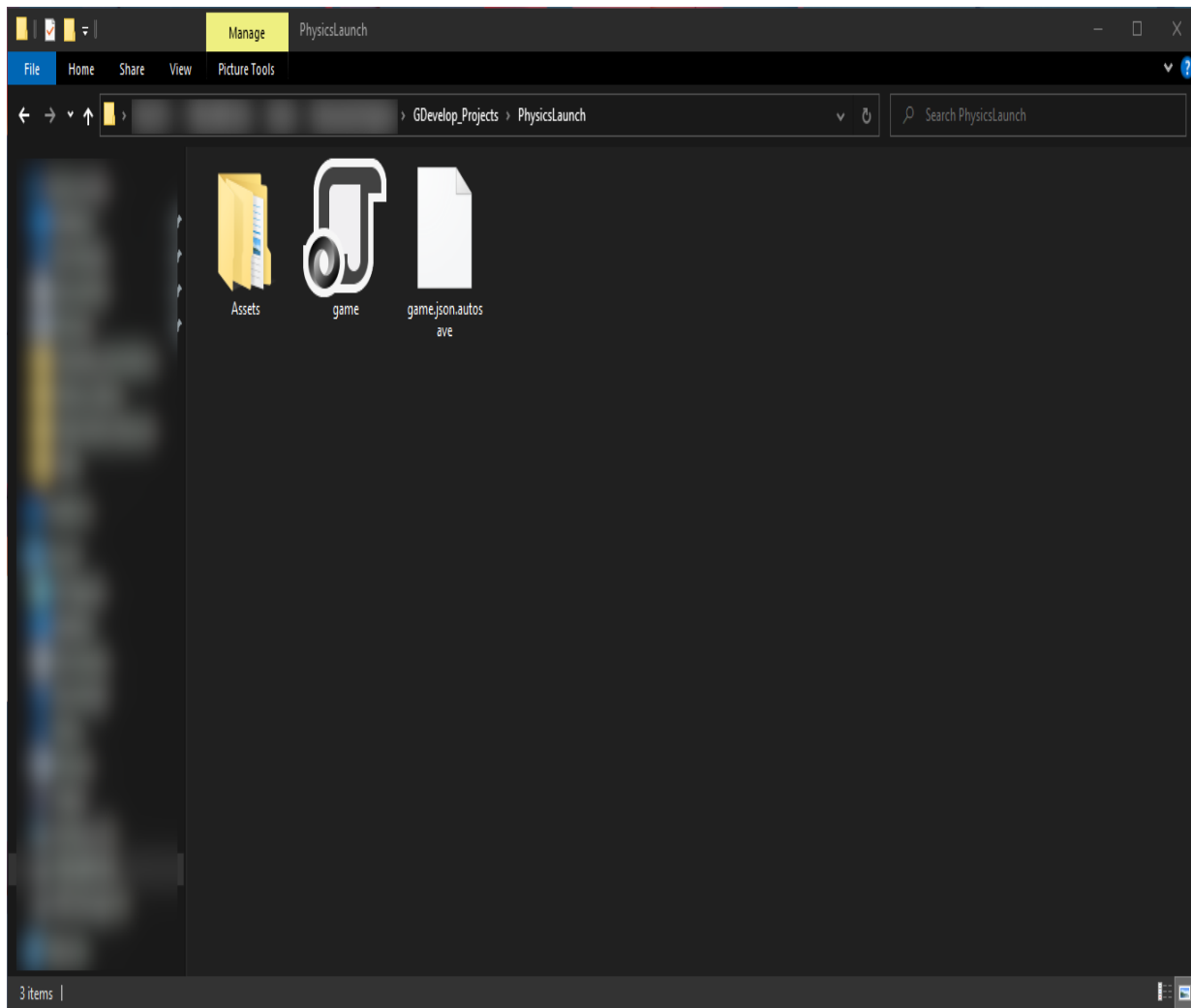


One thing we're going to want to change is how the game window resizes, so the content we place is always in view. Open the game settings and change the "Game resolution resize mode" to "No changes to the game size".

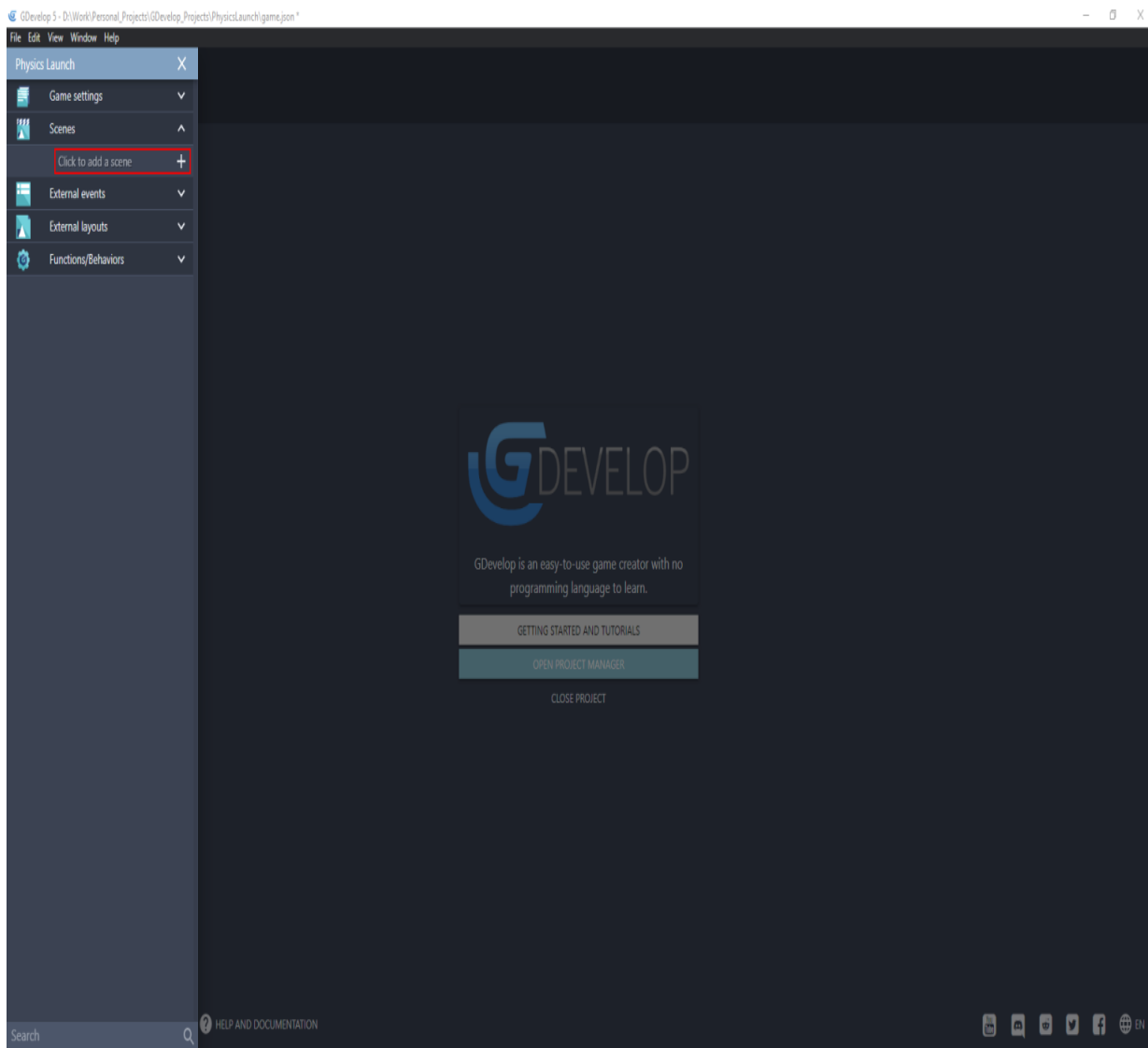


This will ensure that whatever is visible in the camera will be all that is visible, which will make things a little easier as then we don't have to worry about adding extra padding to the sides for players who have smaller or larger screens.

Once you have the project created, navigate to it in the file browser and extract the "Assets" folder from the starter assets into the newly created project folder. Once done, it should look something like this:



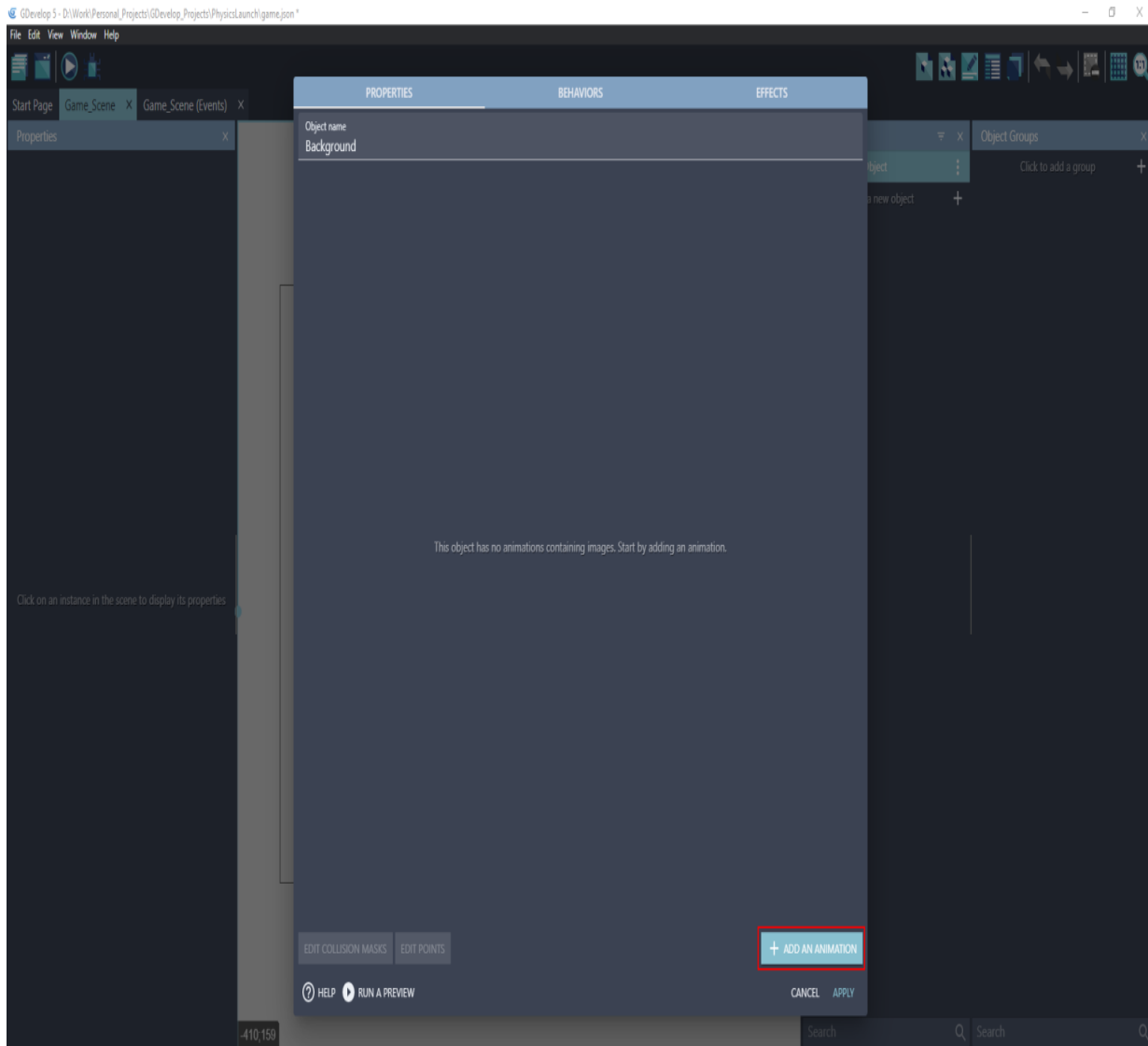
Next, let's create the main game scene for our game. This scene will be where all of our game logic will reside, as well as the objects that compose our level. Select the project manager in the top left, open the scenes drop down, and then select the "+" next to "Click to add a scene".



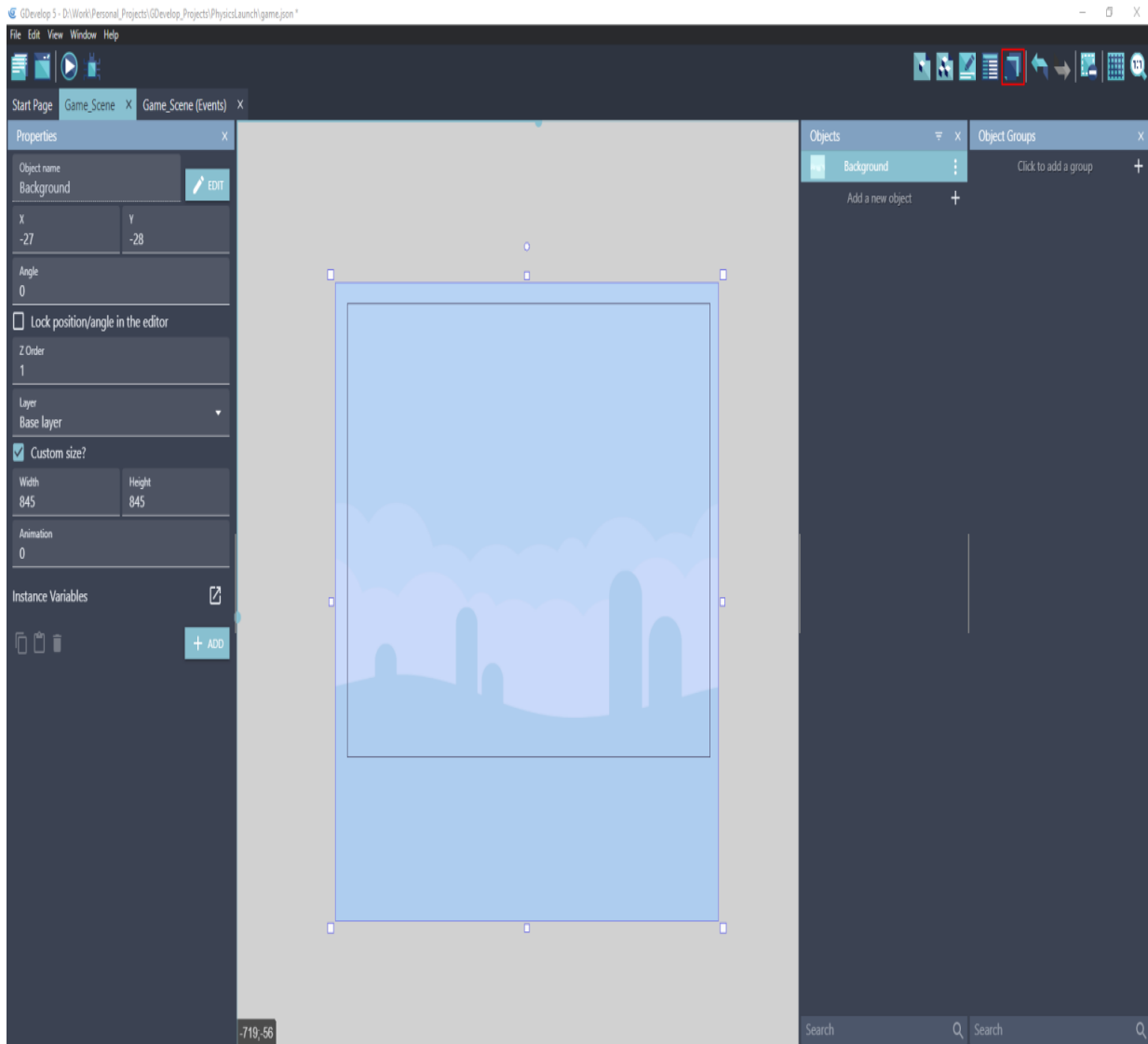
Once you have created the scene, click the three dots on the side and rename the scene to “Game_Scene”. With that done, go ahead and click it to open the scene in the main view.

Setting up the scene objects - Background

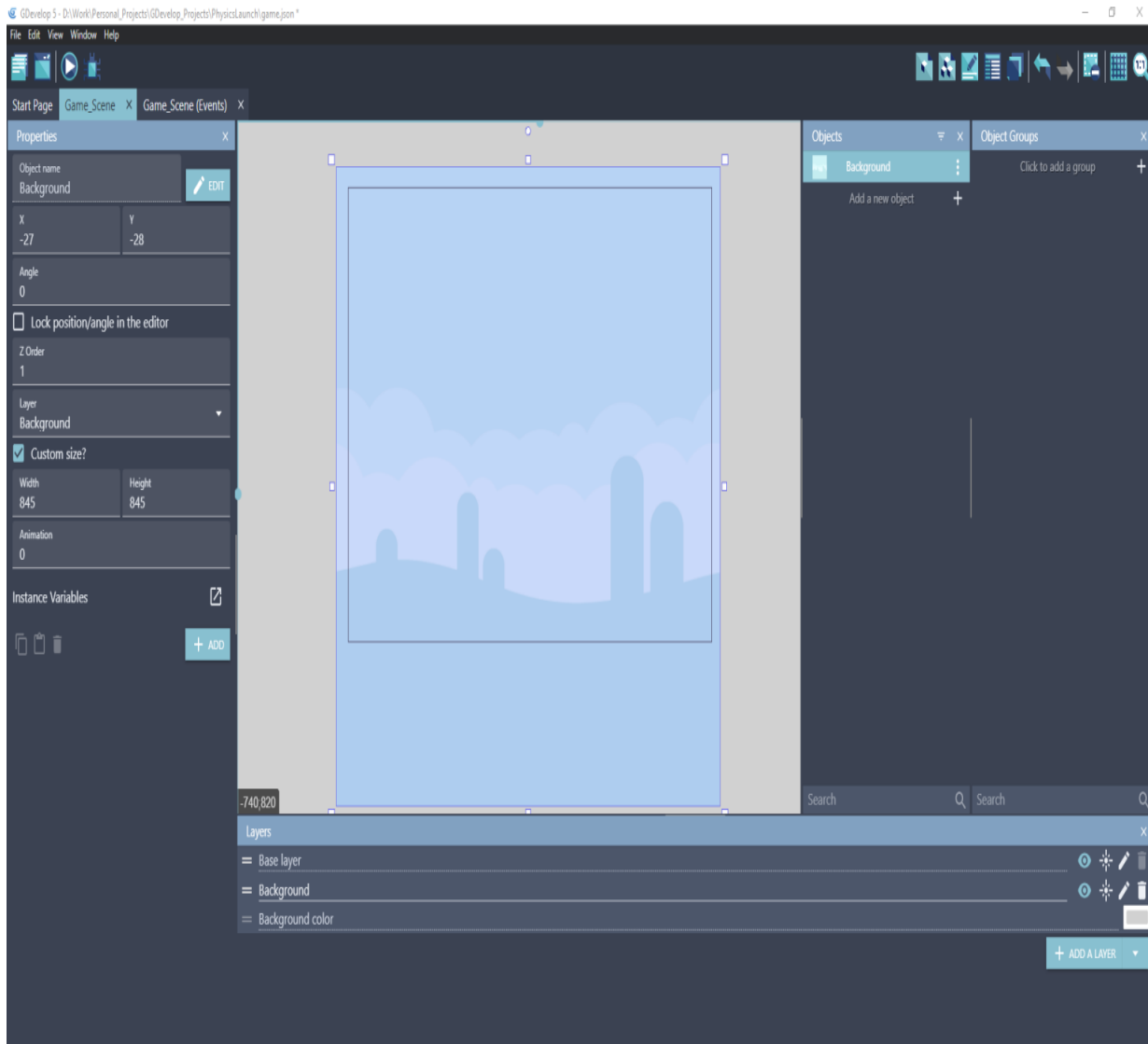
Let’s define some of the object’s well need for the game. In the “Objects” view on the right side, press the “+” next to “Add a new object” and choose “Sprite” from the “New Object From Scratch” menu (should be open by default). Name this object “Background” and then click the “Add an animation” button in the bottom right of the new object window:



Then click the “+ ADD” box and navigate to “Assets/Backgrounds/blue_grass.png”. Select the image and then hit the “Apply” in the bottom right corner of the popup. From the “Objects” menu you should now see your newly created Background object. Go ahead and drag and drop this into the main view and position it inside the camera. You may need to scale it down, which you can do with the scale handles. Once you have it nicely positioned within the camera where you like, go ahead and click it. Now, since this is the background we want to make sure it’s drawn behind all the other objects, so we need to add a new rendering layer to the game. This is easy to do, just click the layer button in the top right to bring up the layer panel:



Then click “Add a layer” and name the layer “Background”. Then drag the layer down so it’s below the “Base Layer” included by default. It should look something like this:



The last thing we need to do is change the layer of the background object we just added to the scene. Click the background and in the properties tab on the left, find the “Layer” drop down and set it to “Background”.

Setting up the scene objects - Slingshot

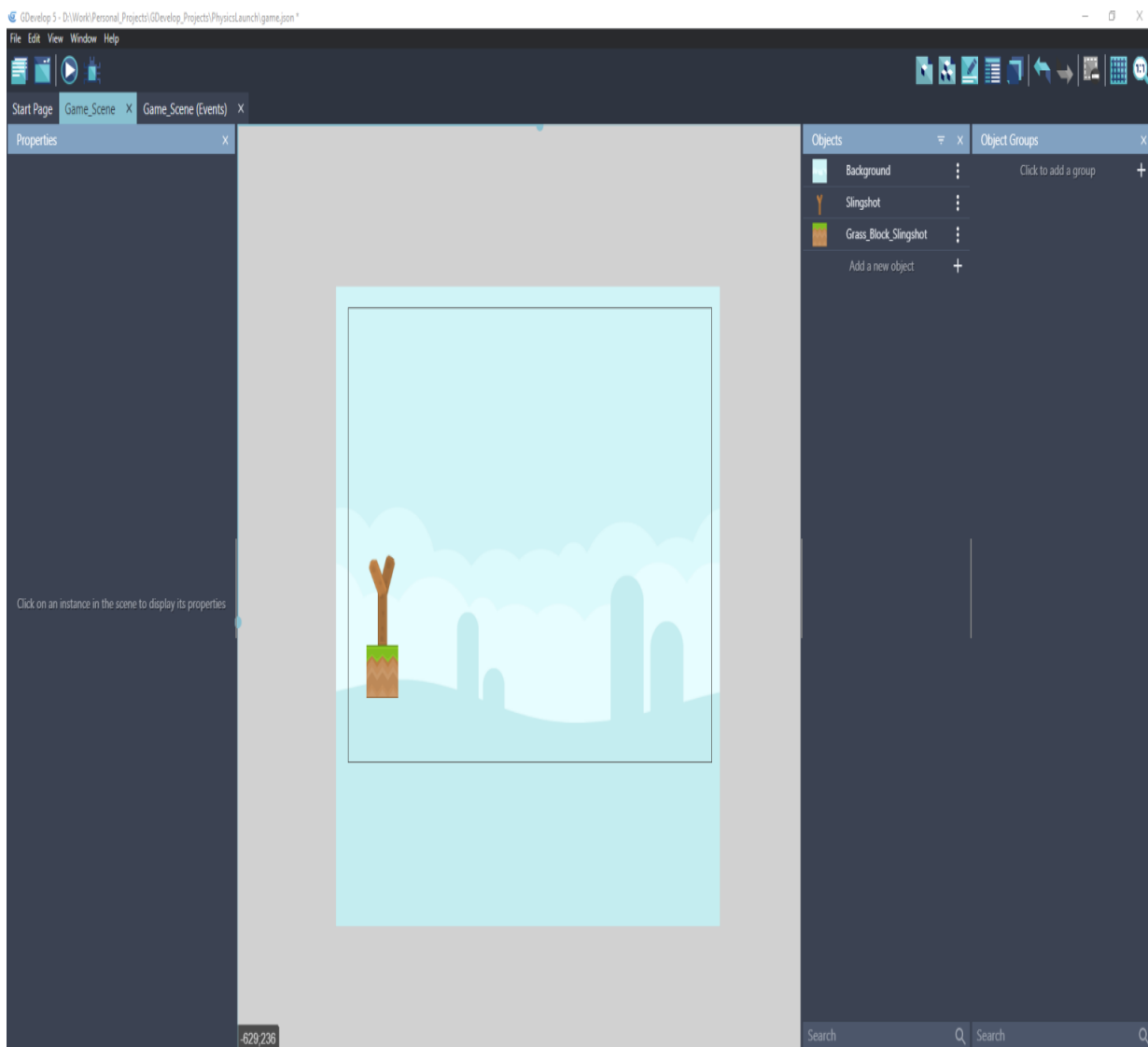
Next let’s setup the slingshot. First, create a new object from in the objects tab by pressing the “+” button. Select “Sprite” and name the sprite Slingshot. Add a new animation and then set the image to the image at “Assets/Custom/Slingshot.png”. Then go ahead and press apply.

Before we drag it into the scene, let’s get the grass block it will rest on setup first. Make another new object by pressing “+” and select “Sprite” again. This time, call the object “Grass_Block_Slingshot”, as this grass block will only be used for the slingshot. Next, add a new animation and set the image to the image age “Assets/Other/grass.png”. Once done, go ahead and hit “apply”.

Tip

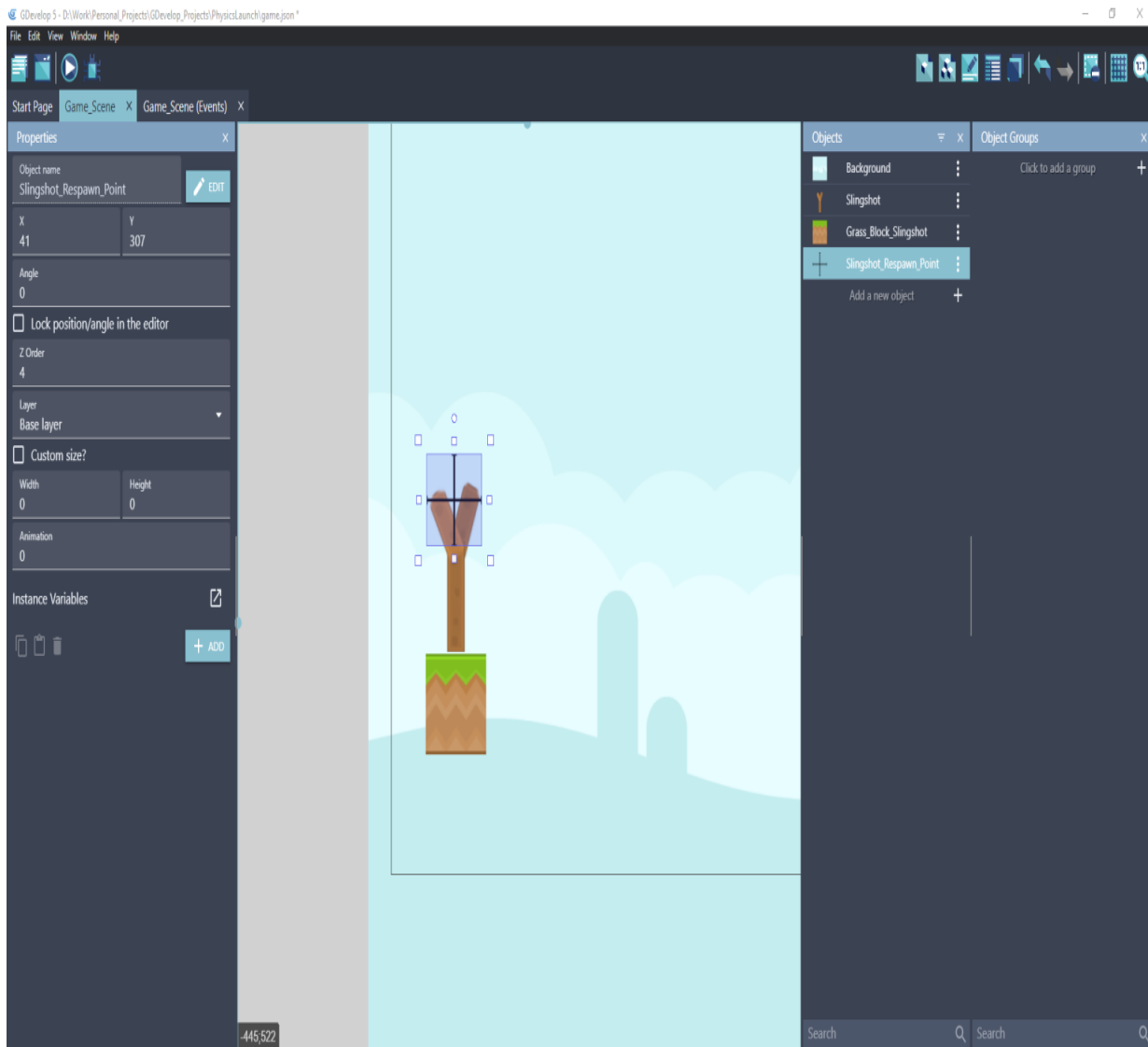
The reason we are making the grass block for the slingshot separate is because later in the tutorial we will add different levels, but we don't want the slingshot grass block to be removed when we are removing the objects when changing levels!

Alright, let's add the new objects to the scene. Go ahead and drag the "Grass_Block_Slingshot" object into the scene and position it to the far right of the camera view, near the bottom. Then place the "Slingshot" object into the scene and position it so it looks like it's on top of the grass block. When finished, it should look something like this:



We are almost done with the setup, we have just a couple more objects to make and then we can start the first steps in making the game itself. First, we are going to need to make a object to know where to spawn balls that we launch from the slingshot. Make a new object by pressing the "+" button, select "Sprite", and then name it "Slingshot_Respawn_Point". Create a new animation and set the image to "Assets/Custom /Respawn_Graphic-1.png", and then hit "Apply". Next, drag it into the scene and position it so the + shape is right in the middle of the curve of

the slingshot so it looks something like this:



We'll be using this object to tell the system where to place the balls we're going to launch. In the actual game, this object will be invisible, so the image we're using here is only for positioning purposes.

Finally, let's create the object the slingshot will be shooting. Create a new object by pressing "+", select "Sprite", and name the object "Player_Alien". Create a new animation and assign the image to "Assets/Aliens/alienGreen_round.png".

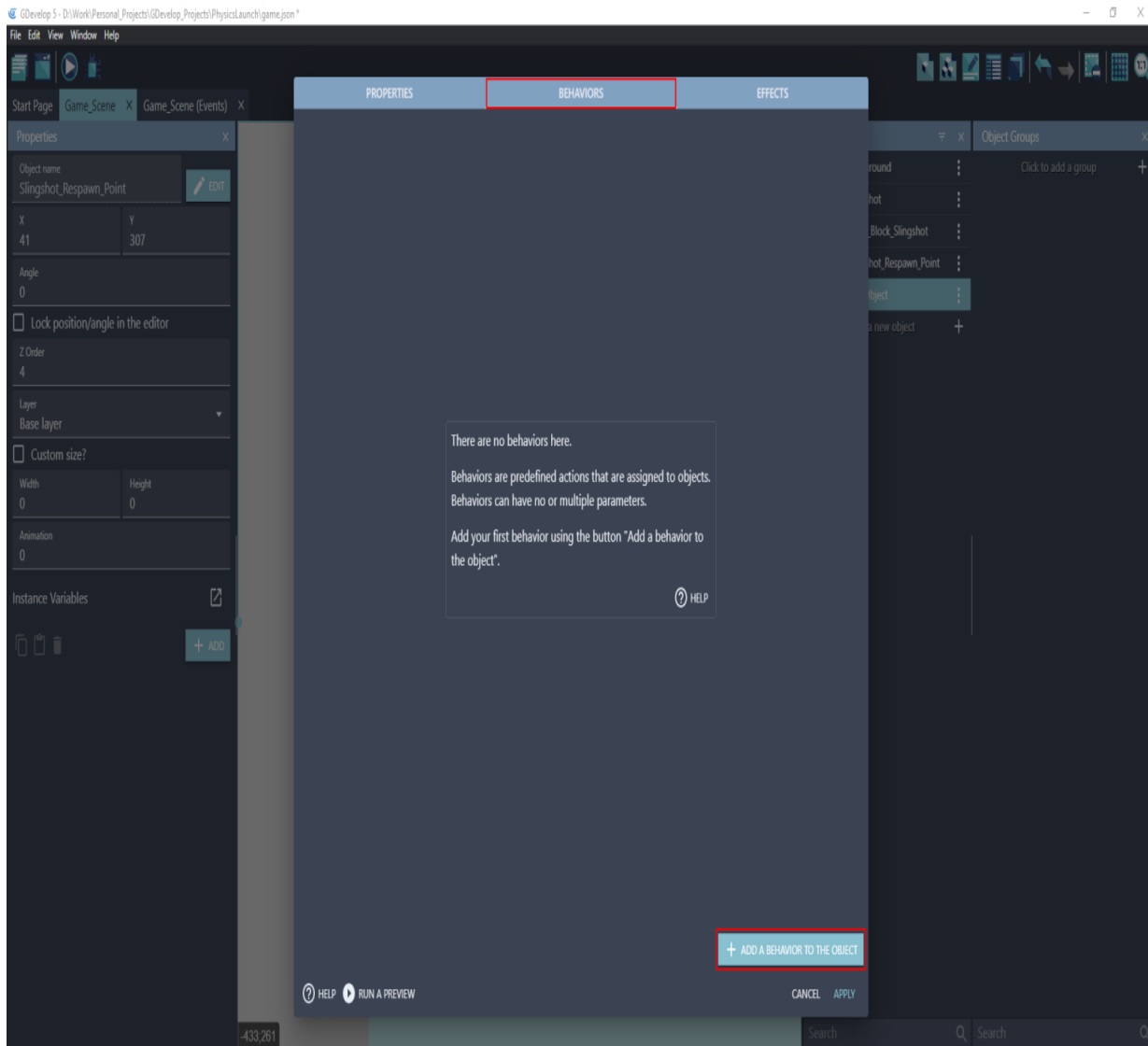
Now, before hitting apply, we have a couple more steps to take. In the top of the object window, select "BEHAVIORS". We will be adding two behaviors to this object, one to destroy the object when it's off screen and another to add physics to the object.

Tip

Behaviors in GDevelop provide a set of functionality to objects that we can use without having to program them ourselves! Behaviors are a great way to add loads of functionality to an object quickly and easily,

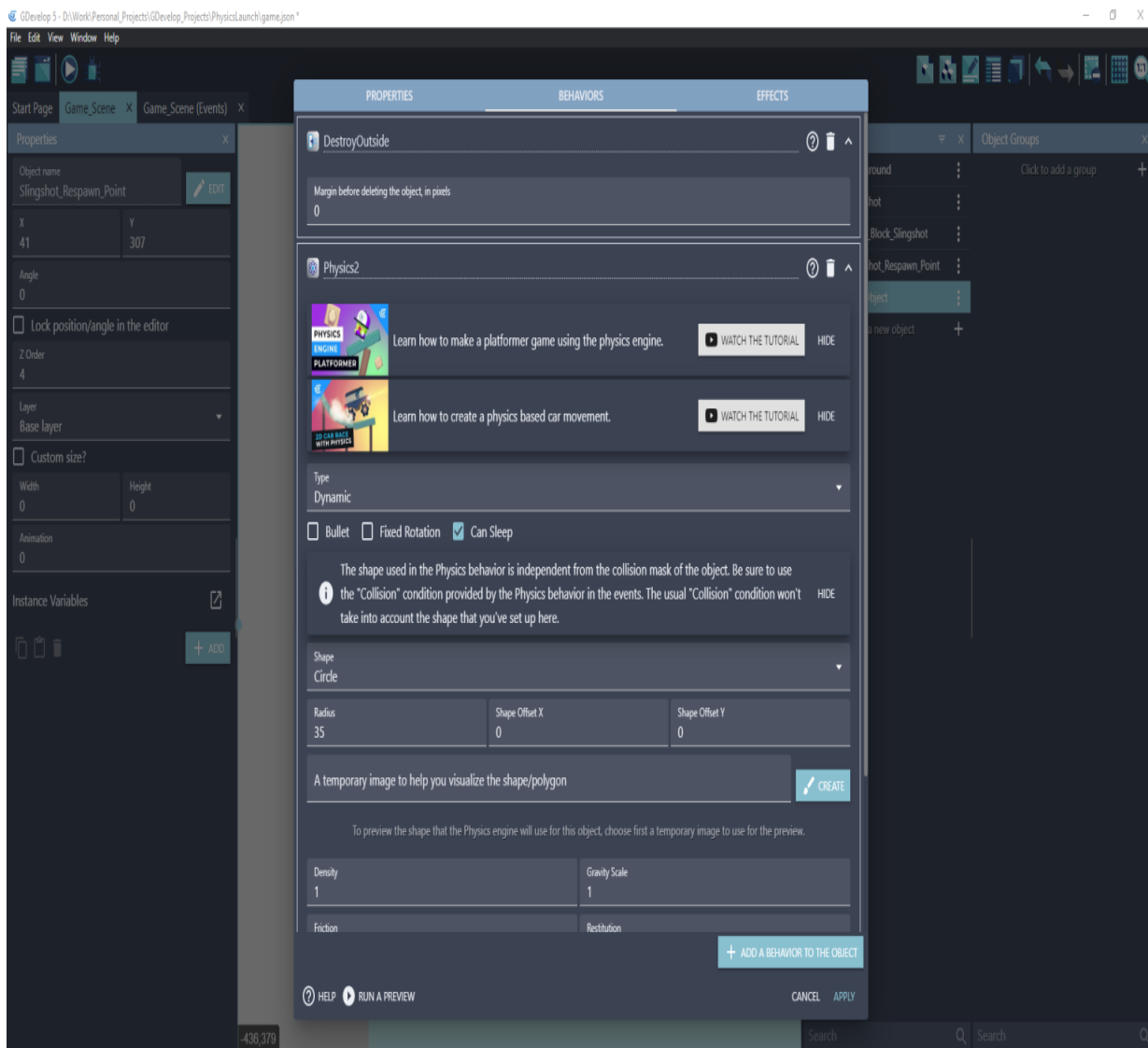
while also providing an interface we can use and manipulate through actions, which we will do later in this tutorial series.

Let's add the behavior that will destroy the object once it's outside of the screen. First, press the "+ ADD A BEHAVIOR TO THE OBJECT" button in the bottom right corner of the window.



Then in the popup that appears, type "Destroy" and select "Destroy when outside of the screen" from the options listed. This will add the behavior to the object. What this behavior will do is whenever an instance/clone of this object is off screen, it will automatically be deleted by the behavior, saving performance automatically.

Next, hit the "+ ADD A BEHAVIOR TO THE OBJECT" button again and this time select "Physics Engine 2.0". This will add a bunch of options, which can be overwhelming, but for this tutorial we will only need to set a couple. First, in the drop down titled "Type", make sure it's set to "Dynamic". Then, set the "Shape" drop down to "Circle" and set the number in the "Radius" property to 35. Here's how it all looks when setup:



There are more settings that allow us to refine and adjust the behavior of the object and how it interacts with physics, but for this tutorial we don't need to configure anything else and will leave it at the defaults.

Tip

While we won't be adjusting many of the properties in the Physics2 behavior, let's quickly go over the two we changed and what they do.

The "Type" property defines how the physics engine in the GDevelop Physics2 behavior will interact with the object. There are three types: **Dynamic**, **Static**, and **Kinematic**.

- **Static**: A non-movable physics object that can interact with other physics objects, but it will not be moved by these collisions. These objects are the simplest and most performance friendly for non-moving objects. They are typically used for objects that make up the level, barriers, or really anything that does not move but is able to be collided with.

- **Dynamic**: A movable physics object that uses velocities, linear and

angular, to move around the physics world. Dynamic physics objects will be automatically moved by the physics behavior and will respond to other collisions, velocity changes, and more. Dynamic objects can be used for almost anything that you want to move and interact with the physics world. We'll be using them to automatically move the balls we fire.

- **Kinematic:** Kinematic mode is similar to Dynamic, however it doesn't move automatically and gives the game developer a lot more control over how it interacts with the physics world. Kinematic bodies are generally used when you have moving physics objects, but you want to precisely control how they work and what they can do.

For our purposes, we want the physics engine to handle movement and how the object collides with the rest of the physics objects in the world, so we'll select "Dynamic" so the physics engine has control.

For the "Shape" property, this just tells the physics engine what shape our object has. Because our object is a circle, we want to use a circle physics shape as well. For the radius, it is set to 35 pixels because the circle alien image we're using is 70 pixels long in diameter. You can verify that 35 pixels is the right radius size by selecting the green alien image in the "A temporary image to help you visualize the shape/polygon" drop down. After selecting the image you will have a little view that shows the collision shape (red) over the image.

File Edit View Window Help

Start Page Game_Scene X Game_Scene (Events) X

Properties X

Object name
Slingshot_Respawn_Point EDIT

X
41

Y
307

Angle
0

☐ Lock position/angle in the editor

Z Order
4

Layer
Base layer

☐ Custom size?

Width
0

Height
0

Animation
0

Instance Variables

+ ADD

PROPERTIES BEHAVIORS EFFECTS

Dynamic

☐ Bullet ☐ Fixed Rotation ☒ Can Sleep

i

The shape used in the Physics behavior is independent from the collision mask of the object. Be sure to use the "Collision" condition provided by the Physics behavior in the events. The usual "Collision" condition won't take into account the shape that you've set up here.

HIDE

Shape
Circle


Radius
35

Shape Offset X
0

Shape Offset Y
0

A temporary image to help you visualize the shape/polygon
Assets\Aliens\alienGreen_round.png EDIT

Q Q



Density
1

Gravity Scale
1

Friction
0.3

Restitution
0.1

Linear Damping
0.1

Angular Damping
0.1

Layers

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

Masks

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

+ ADD A BEHAVIOR TO THE OBJECT

HELP RUN A PREVIEW

CANCEL APPLY

With all that done, we've gotten everything we need setup and can move on to creating the beginnings of the game in Part 2: launching balls from the slingshot! You can find part 2 here: [Part 2](#).

You can find the finished project for part 1 right here: [Download Link](#).