GDevelop Physics Launch Tutorial – Part 2

Posted on: October 14, 2021 Last updated on: October 14, 2021
Categorized in: GDevelop Tutorials Written by: TwistedTwigleg

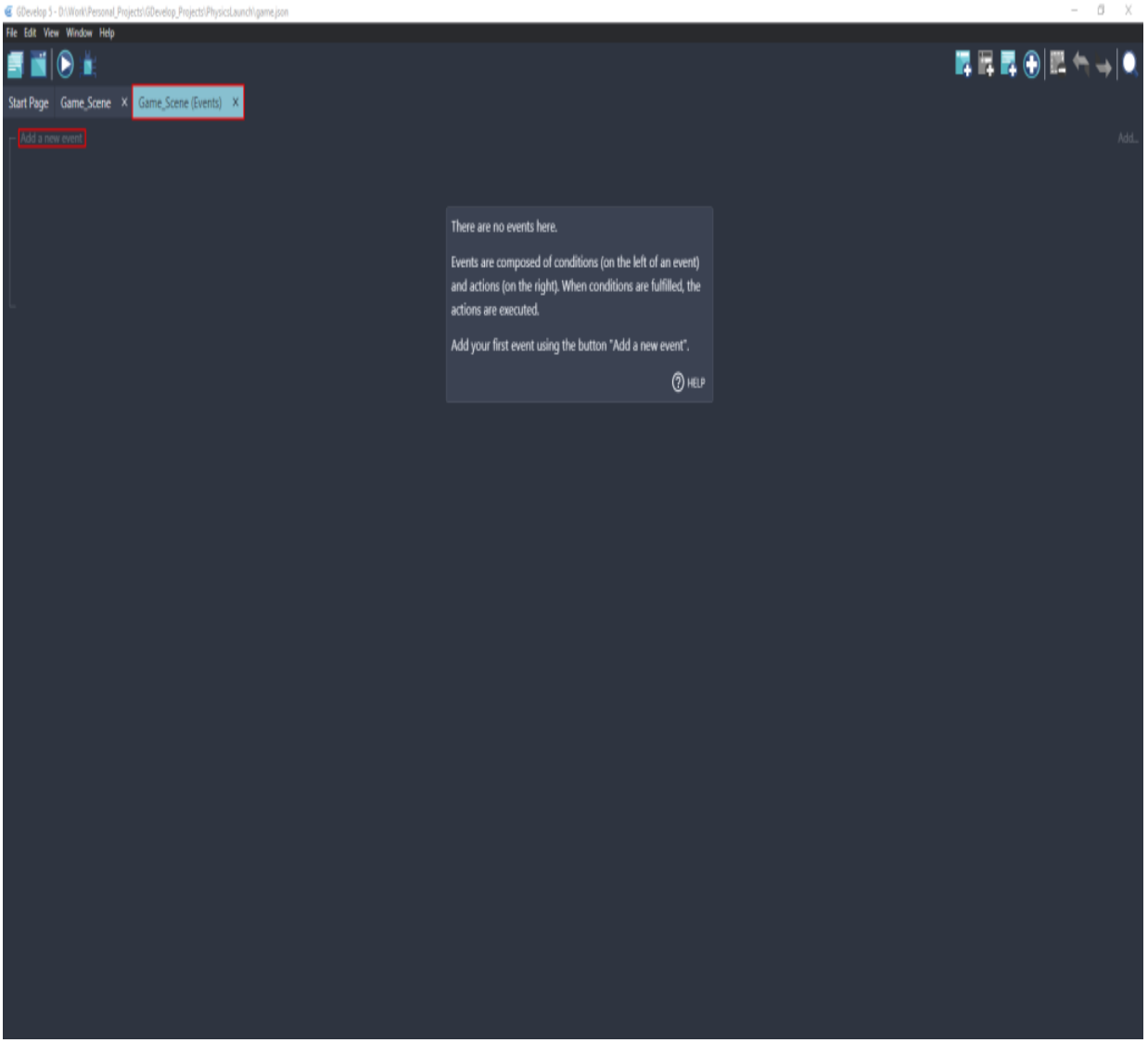
In this tutorial series, we are creating a physics-based game similar to that of Angry Birds using the GDevelop game engine. In part 1 we got everything setup, and now in part 2 we are going to start creating the main mechanic for the game: launching balls from the slingshot!

If you want to jump on to the tutorial from this part, you can find the finished result from part 1 right here: Part 1 Result.

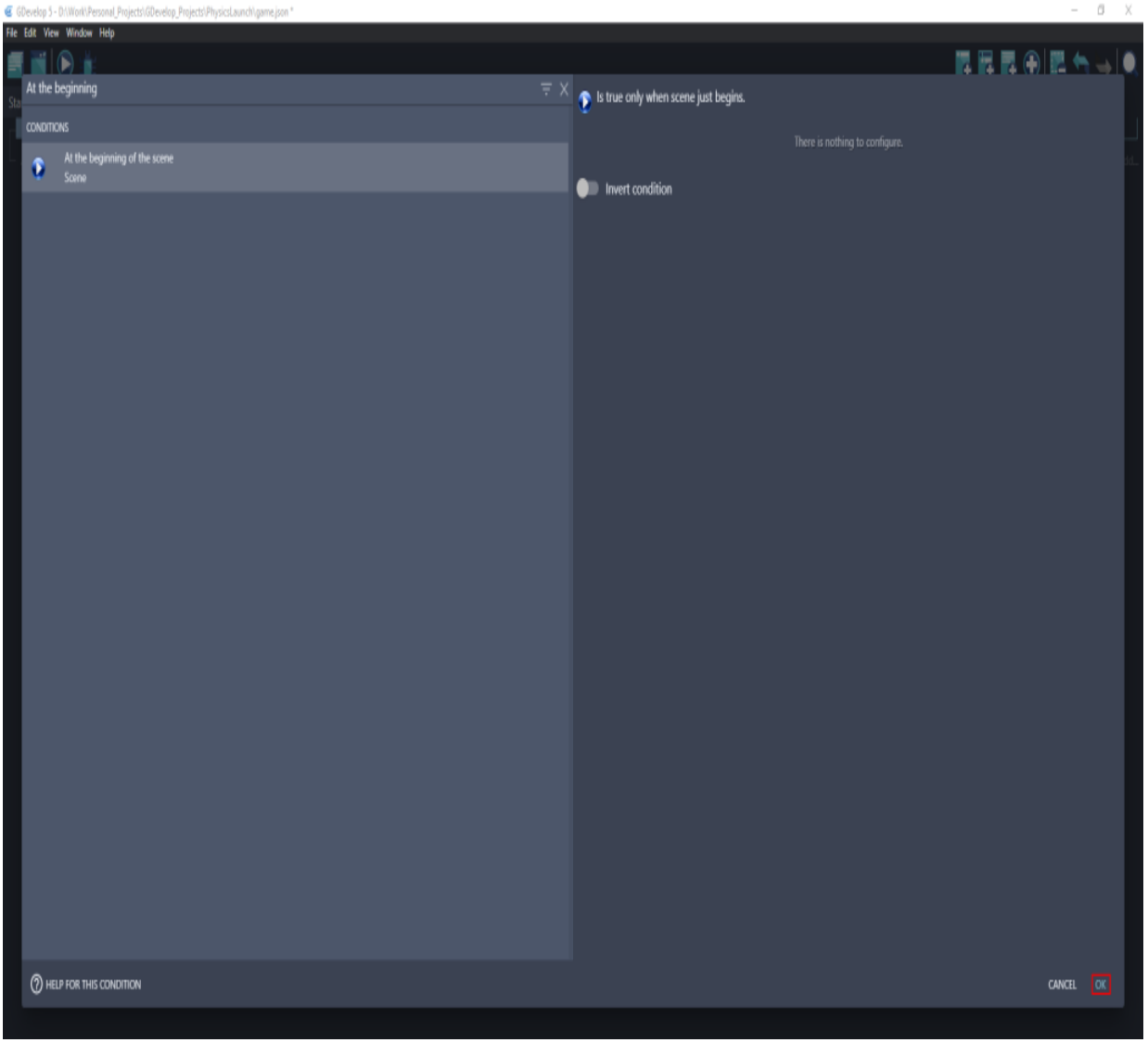
Setting up the slingshot

In the last part we got everything we needed setup and ready. We made objects for the background, slingshot, a grass block for the slingshot, a respawn point for the balls we'll be launching, and finally an alien ball to launch.

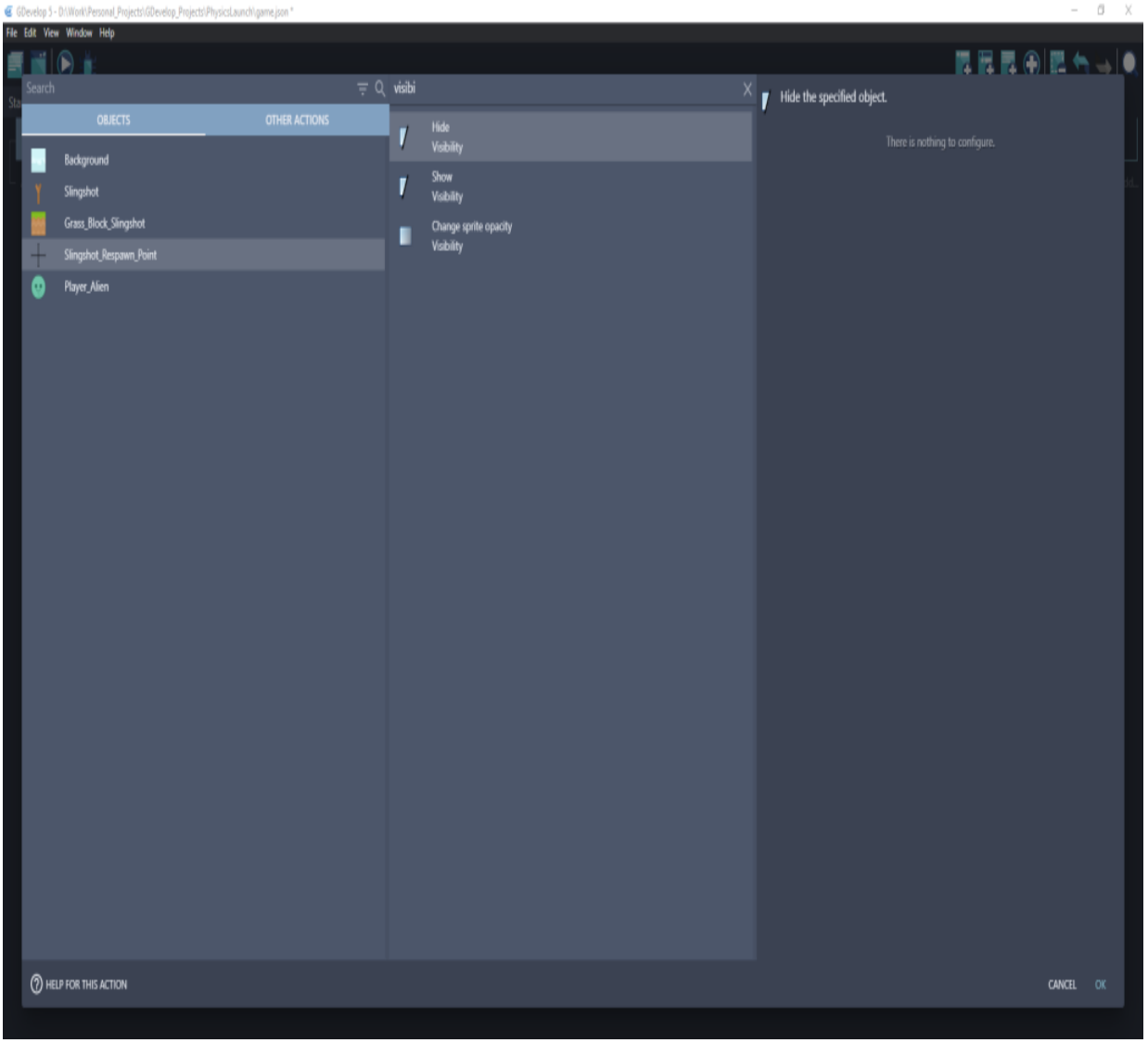
To make our game function, we need to jump into the events graph for the Game_Scene file we made in part 1. Click the "Game_Scene (Events)" tab on the top of the main window and then click "Add a new event".



This will add a new event with no conditions or actions. Conditions are the requirements needed for an event to activate, while actions are what happens when the event is activated. Go ahead and click the “Add condition” button in the newly added condition. In the window that pops up, search for “At the beginning of scene” and click it. Then on the bottom right, press the “OK” button to add it to the event.



This will make it where whatever actions we have on this event, they will only be called once the scene just started. We can use this to do some initial setup. In this case, what we want to do is change the visibility of the slingshot respawn point so it’s not visible in the game. To do this, click “Add action” and then select “Slingshot_Respawn_Point” from the list of objects. Once selected, search for visibility and select “hide”, and then press OK on the bottom right of the screen.



This will make the slingshot respawn object invisible, so the player cannot see it but we can still use it to position the objects we spawn, which is exactly what we're wanting.

Next, let's add the events to spawn balls when we click the screen. Click "Add a new event" and search for "Mouse button pressed or touch held". Select it and set the button to the left mouse button, and then hit "OK" to add it.

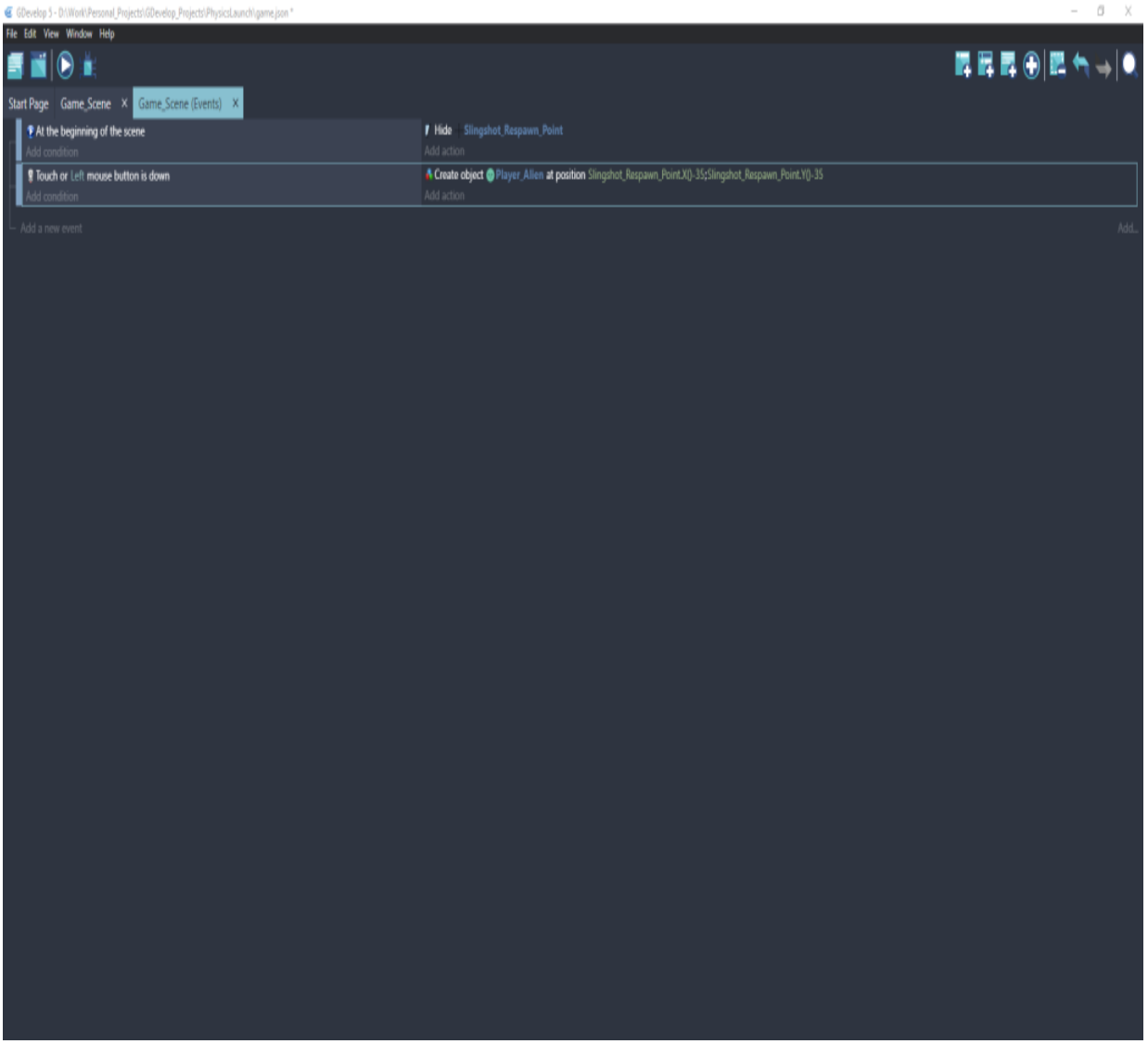
Next, click "add action" on the newly created left mouse button event and search for "Create an object". Once you find it, select it. In the properties tab, we have a few things we'll need to setup in order to get it working. First, in the "Object to Create" property, select "Player_Alien". In the "X Position" property, write "Slingshot_Respawn_Point.X()-35" and for the "Y Position" property, write "Slingshot_Respawn_Point.Y()-35". What this will do is get the position of the Slingshot_Respawn_Point object in the scene and make a new alien ball object at this position.

Tip

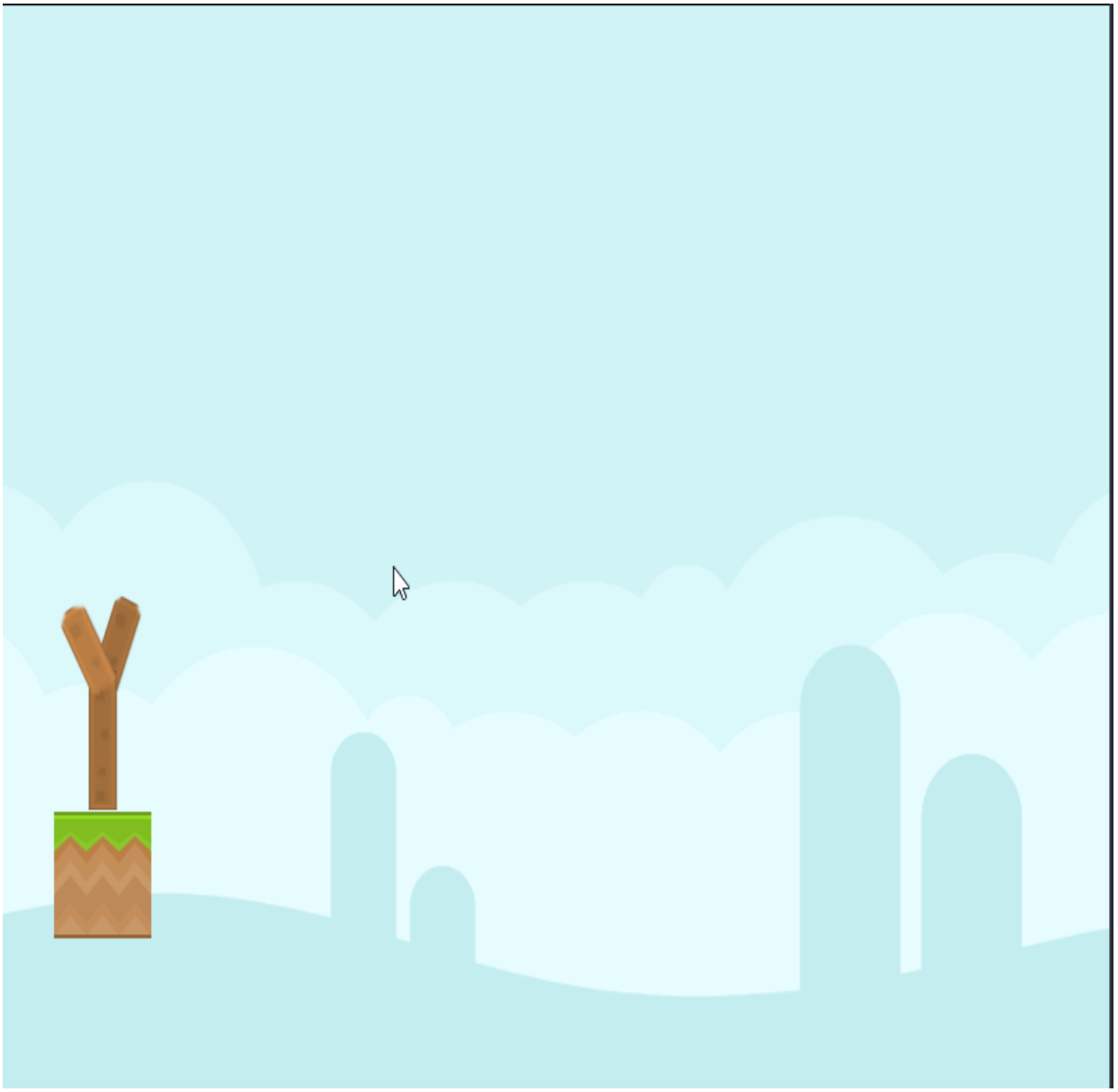
You might be wondering why we are subtracting 35 from the X and Y

position of the slingshot. This is because when creating an object, it positions the new object's top left corner to the position we pass in, rather than the center. However, we want the alien to appear in the center of the slingshot, so we add a 35 pixel offset, which is half the size of the alien object, so it's properly centered when added to the scene!

That is all we need to do for now, so go ahead and click "OK" in the bottom right. Right now, the events should look something like this:



Let's give it a test! Go ahead and click the "Play" button in the top left and try clicking around. You should get something like this:



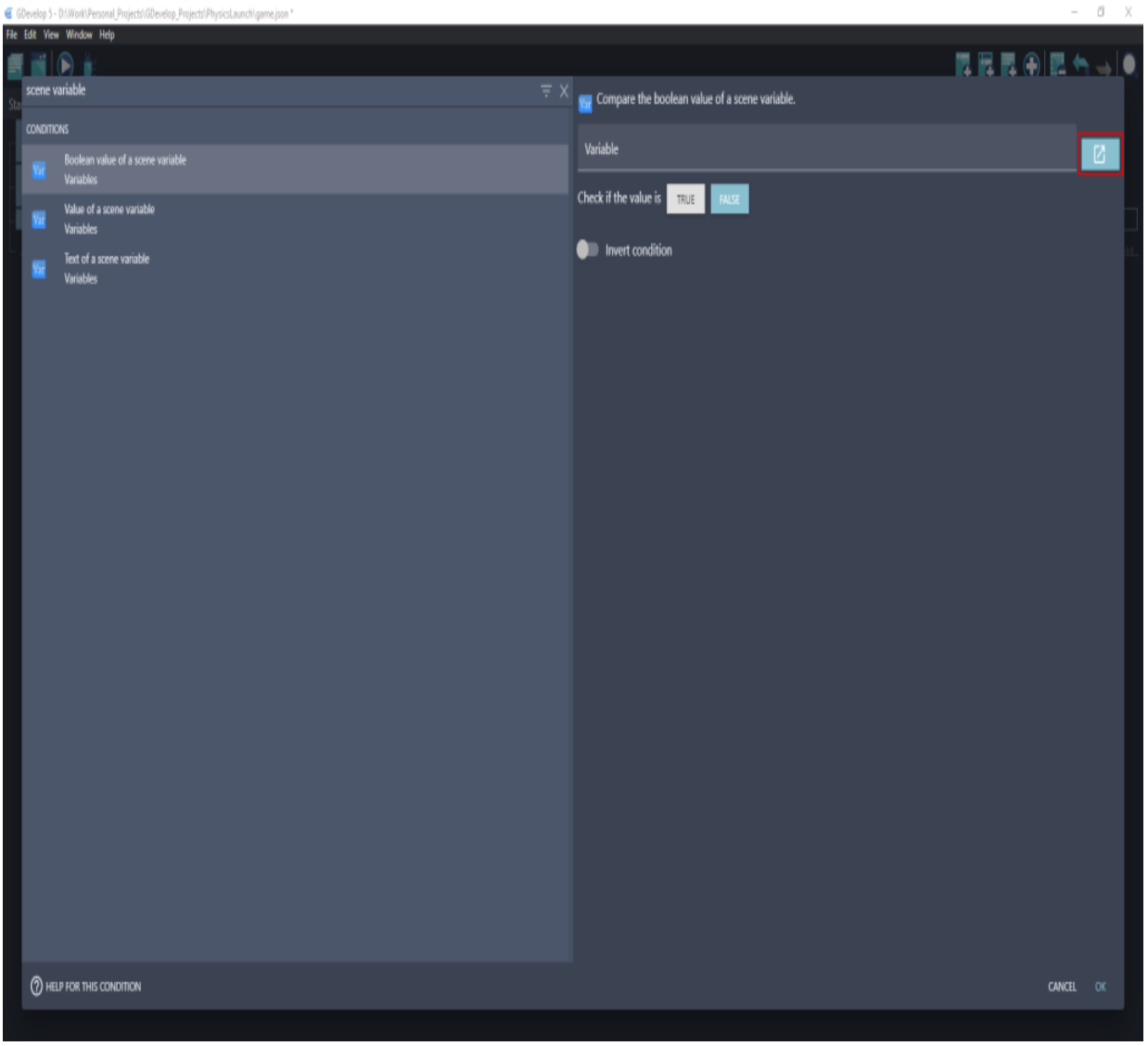
Great! We are spawning balls, but right now we don’t have any control over it, it’s spawning more than one per click, and it’s not really being launched from the slingshot. Let’s fix all of that and get something closer to what we’re expecting

Controlling the slingshot: Alien creation

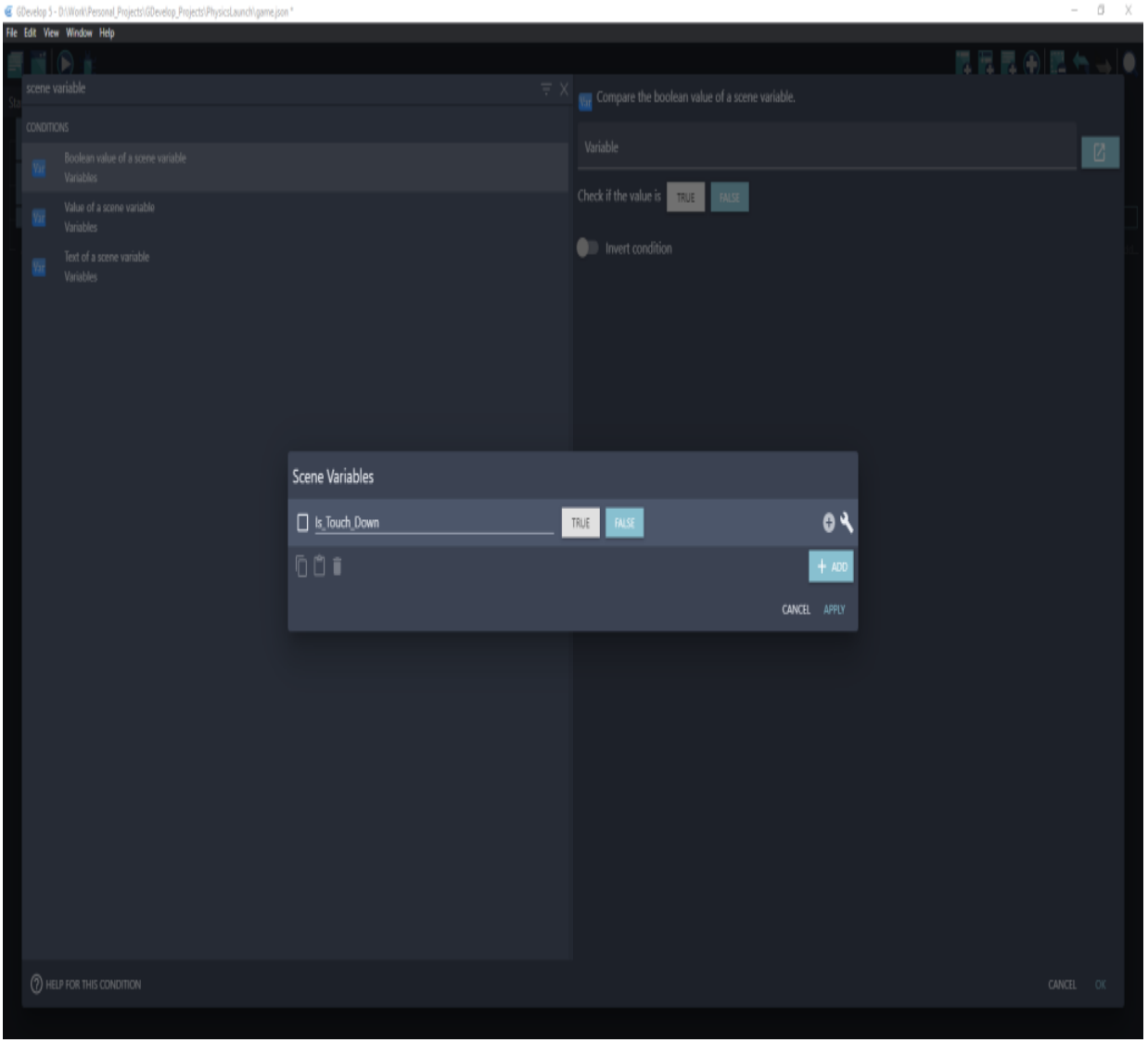
First, let’s work on getting only a single ball to spawn per click. Open up the events again and add a new event. Select “Add condition” for the new event and search for “Scene Variable”. This will bring up a bunch of options, but what we’re looking for is “Boolean value of a scene variable”, so go ahead and select that.

We’re going to use a scene variable to track whether the mouse button was just pressed and just released, so we only spawn a single ball per mouse click. We need to do this because the event for tracking a mouse press activates every time the mouse is pressed, which isn’t ideal and leads to the issue of having so many balls spawn. A scene variable is just a way to store data (a variable) in the scene file, that we can access from only within the scene but we can use anywhere in the scene.

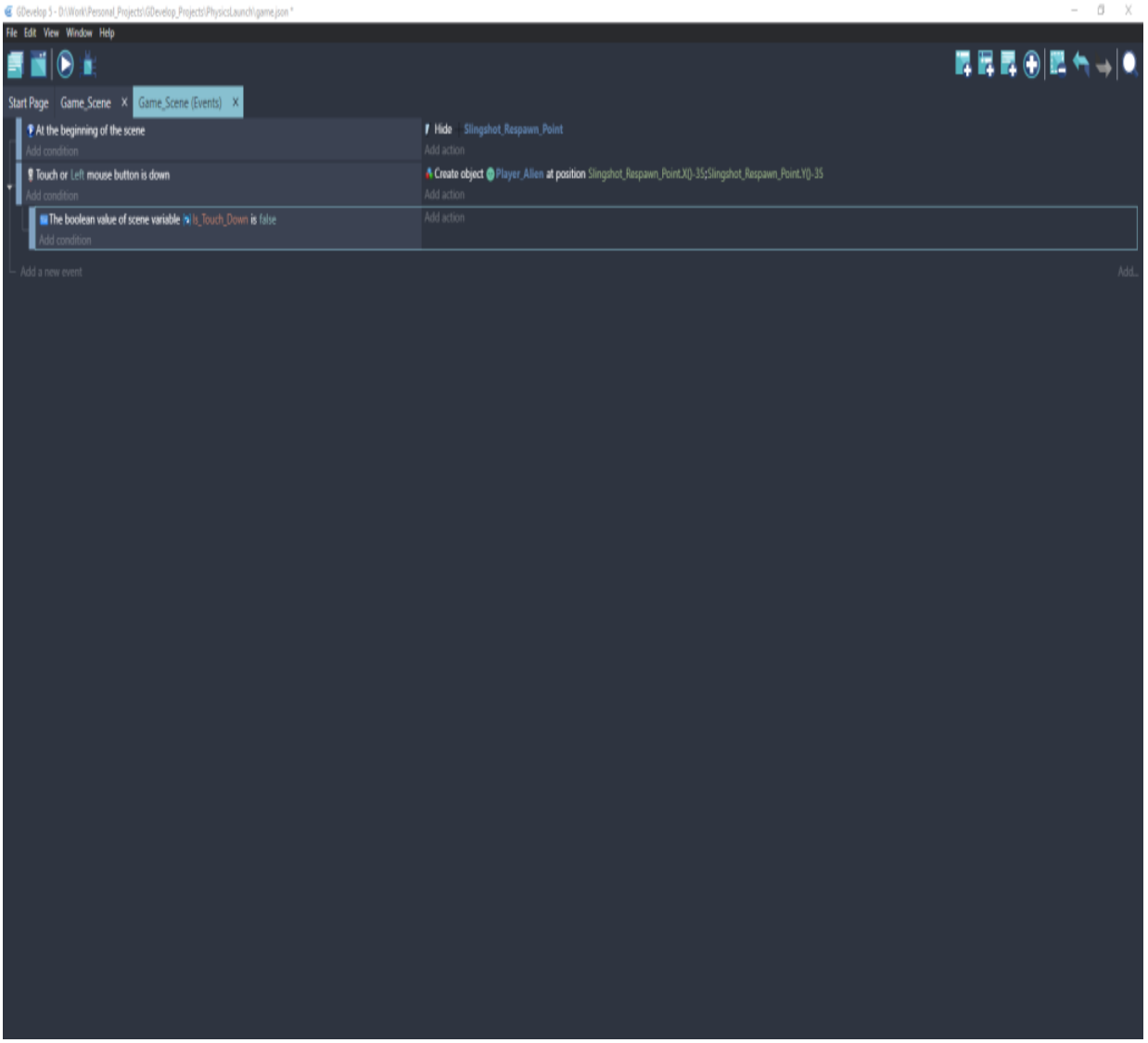
First, we need to define the scene variable, which we can do from the events page. Select “Boolean value of a scene variable” and then next to the “Variable” tab, press the little blue button:



Then in the window that pops up, press “+ ADD” to add a new scene variable. Call this variable “Is_Touch_Down”. Then press the little wrench icon, select “primitive types” and finally select “Boolean”. This will give a TRUE and FALSE value to select, which we want to make sure “FALSE” is selected. When all setup, it should look like this:



Then click “APPLY”. Now in the properties of the condition, click “Variable” and then select “Is_Touch_Down” and set “Check if the value is” to “FALSE” and then press OK. Now what we want to do is make this action a sub-action of the “Touch or left mouse button is down” action, so it only gets called if the left mouse button is down AND it’s own condition (checking if “Is_Touch_Down” is false) are both true. To do this, click the little blue bar on the left and drag it to the right a little until it is under the “Touch or left mouse button is down” action. Once it’s under, drop it and you should have something like this:



Great! Now add an action to the “Is_Touch_Down” event and search for “Scene Variable”. Select “Boolean value of a scene variable”, select “Is_Touch_Down” in the variable property, and set the new value to “TRUE”. Then hit OK.

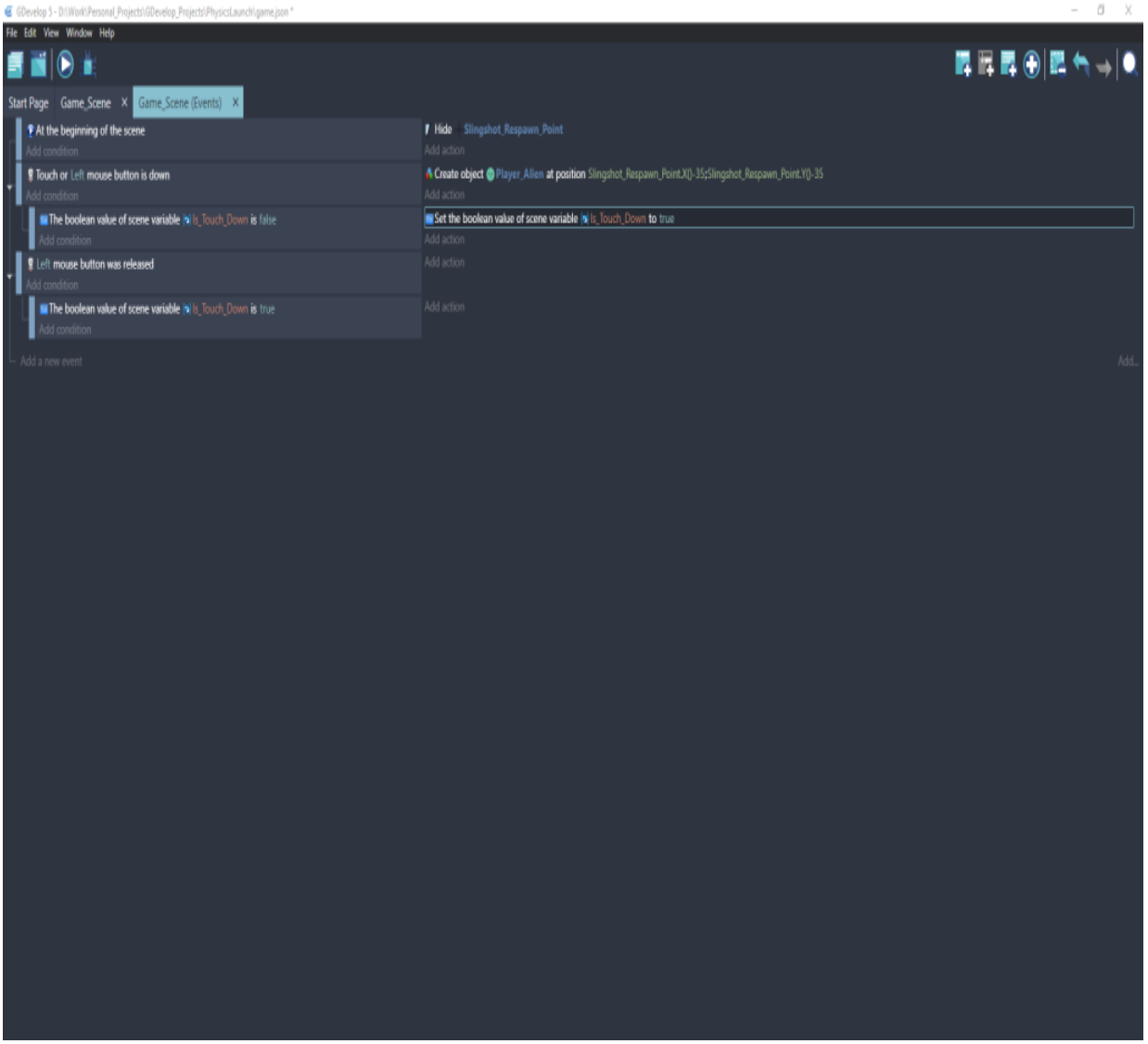
Tip

You might notice that what this will do is immediately turn the variable to true, which will prevent the condition from being called. This is exactly what we want, as then whatever other actions we place in “The boolean value of scene variable Is_Touch_Down is false” will only be called once on just the first mouse press. However, we will also need to reset this variable back when the mouse is released, otherwise this event would only ever be called on just the very first mouse press!

Next, let’s add another new event. Select “Add a new event”, select “Add a new condition” and search for “Mouse release”. Select “Mouse button released” and select “Left” and then “OK”. Then add another new event, add a condition, search for “Scene Variable” and select “Boolean value of a

scene variable”. Select “Is_Touch_Down” again, but then set “Check if the value is” to “TRUE” instead of “FALSE”. Finally, drag the event so it’s a sub-event of the “Left mouse button was released” event.

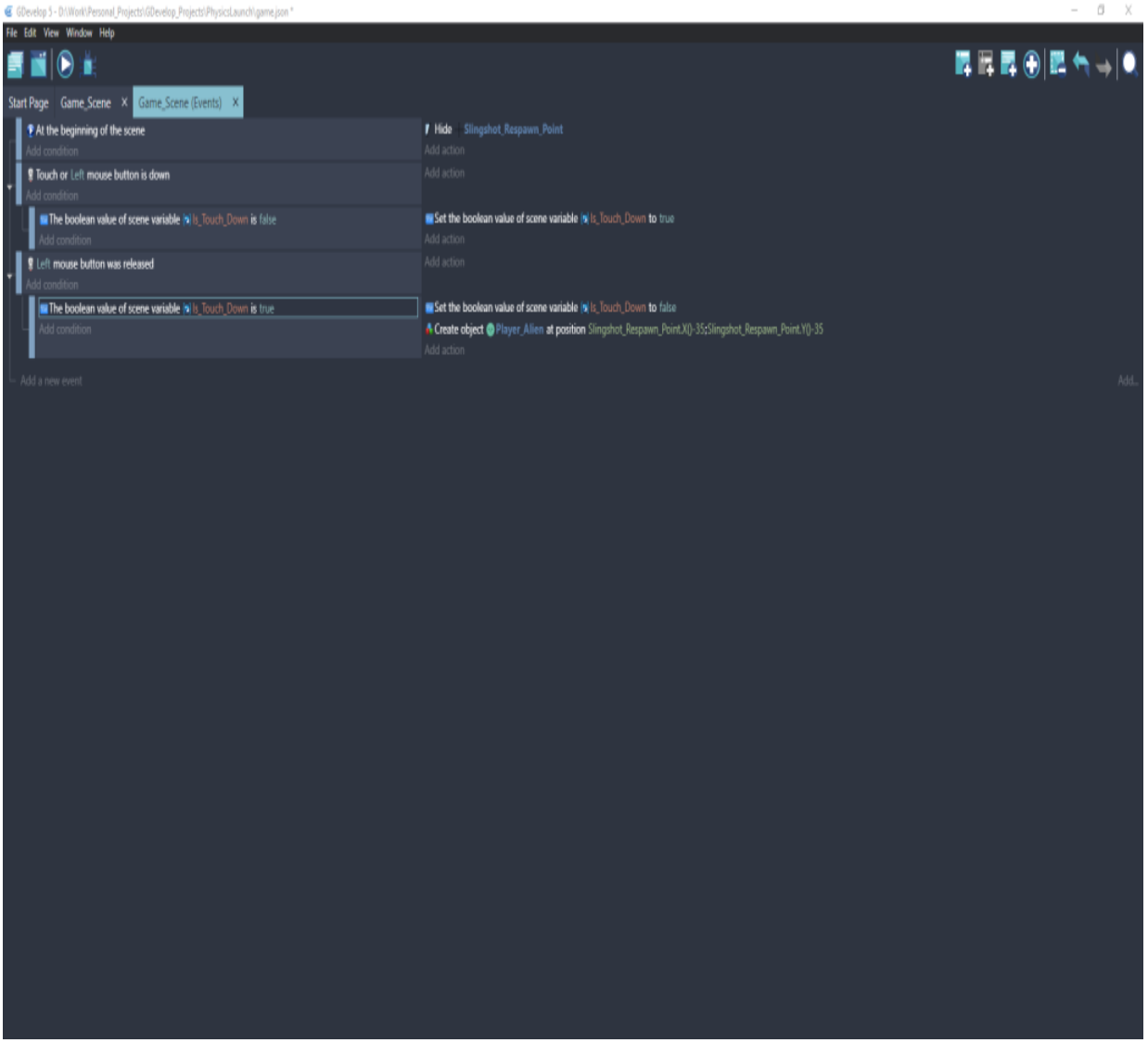
When finished, it should look something like this:



Next, we need to add an action to the “The boolean value of a scene variable Is_Touch_Down is False”. Click “Add action” and then type “Scene Variable” and select “Boolean value of a scene variable”. Set it to “Is_Touch_Down”, set the value to “False”, and then press “OK”.

What this will do is reset “Is_Touch_Down” back to “false” when the mouse button is released, which will let the events for when we detect the mouse button being pressed to be called again. This allows us to track the mouse press and release exactly when they happen, which is what we want.

Finally, let’s drag the “Create object Player Alien at position …” action from the “Touch or Left mouse button is down” event to the “The boolean value of scene variable Is_Touch_Down is false”, so we create a new alien whenever the mouse button is released. Just select the action, and then drag and drop it right under “Set the boolean value of scene variable Is_Touch_Down is false”. When finished, it should look like this:



Go ahead and give it a try. You might find that your balls are not spawning quite in the center, so adjust the slingshot respawn point if needed so they appear right in the center of the slingshot curves. You should find when playing now, that the balls only appear when you release the mouse, which is exactly what we're looking for!

Next, let's working on sending the balls based on the position we drag, similar to how sending birds from a slingshot works in Angry Birds.

Controlling the slingshot: Velocity

The first thing we are going to need to do is have a way to store the position of the mouse click, and then we can compare that position against the position of the mouse release to find out what angle the player wants to release the ball at. To do this, however, we will need something to store the location. Let's use an object for this!

Select "Game Scene" and make a new object. Select "Sprite" and call the object "Touch_Start_Pos". Add a new animation and use the image at "Assets/Custom/Respawn_Graphic-1.png". Like with the respawn point, this is just to help us visualize the results and will not be in the finished game. Once that is done, go ahead and hit "Apply". Finally, drag the

“Touch_Start_Pos” object to anywhere in the scene. It’s position doesn’t matter, as we will be overriding it in the events.

Open up the “Game_Scene (Events)” tab again. First, let’s add an action to “The boolean value of scene variable Is_Touch_Down is false”. Press “Add action” and select “Touch_Start_Pos” as the object from the object list. Then, search for “Position” and choose the option that simply says “Position”. In the properties that come up, set the modification’s sign for both the X and Y position to “= (set to)”. Then set the “X position” to “MouseX(“”, 0)-32″ and the Y position to “MouseY(“”, 0)-32″, which will get the X and Y positions of the mouse cursor, but with a 32 pixel offset.

Tip

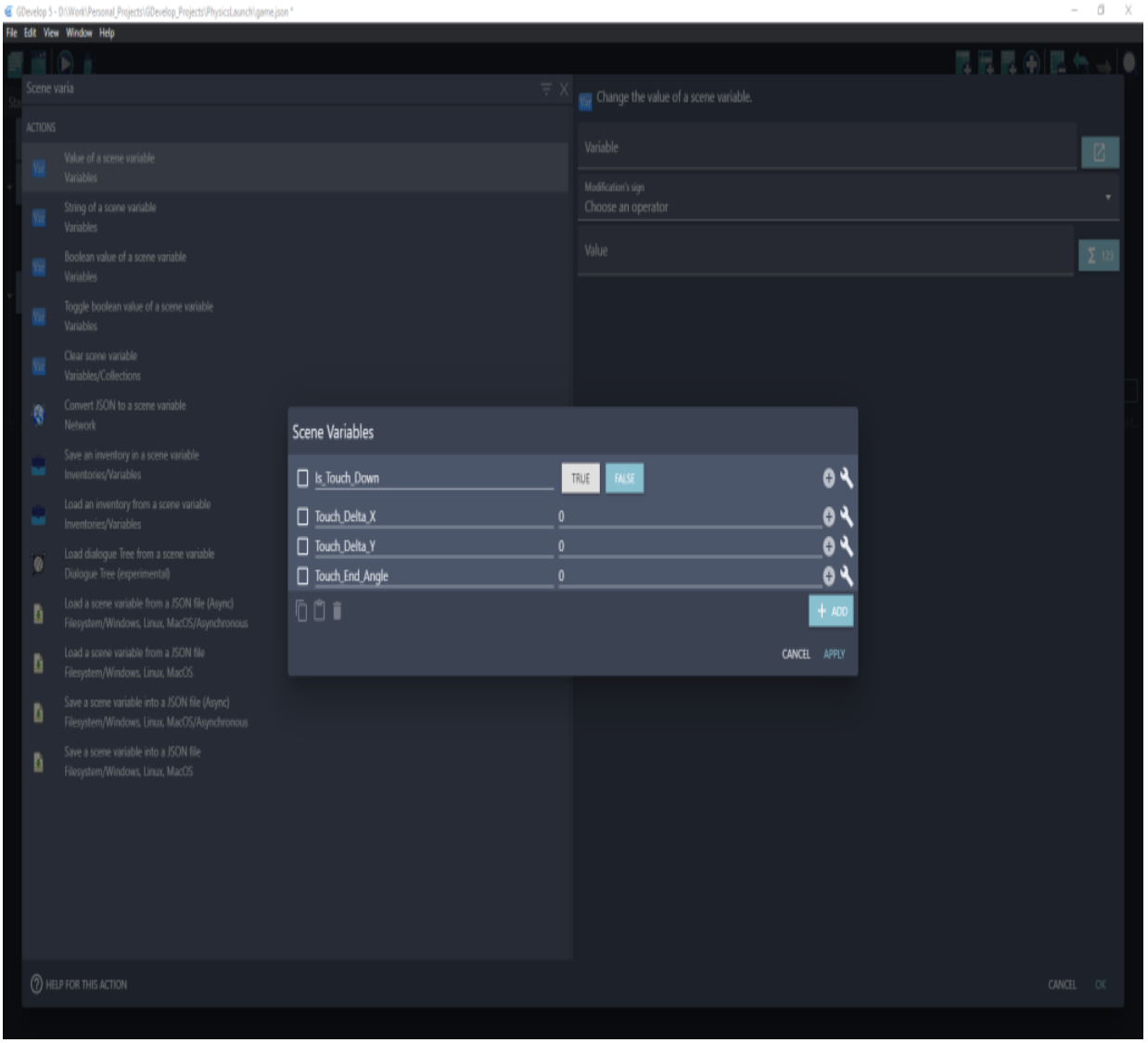
> The offset is for positioning it in the center. You might be wondering why we’re not just using “Center Position” instead then, as it would position it in the center automatically, and that is a good question! You could totally use “Center Position” instead and just skip the -32 pixel offset and it should work just fine. I didn’t for this tutorial simply because I didn’t see it when I was writing it and when I noticed I was too far along and decided to just leave it.

If you try the game now, you will find that when you click, the “Touch_Start_Pos” object goes to the exact position of the click, which is what we are looking for. Now we can use this in data to get the angle of our drag, which we can use as the direction to launch the balls from the slingshot.

First, add a new event and make it a sub event of “Left mouse button was released”. Now we’re going to need to do some math to get the direction from the mouse press start to the mouse press end, and then get the angle from this direction. Don’t worry though, it’s easy to do and GDevelop has some nice functions that makes it easy.

In the new event we just added, click “Add action” and search for “Scene Variable”. Then select “Value of a scene variable”. In the properties, select the little blue button so we can add some new scene variables. We will be adding three new scene variables called “Touch_Delta_X”, “Touch_Delta_Y”, and “Touch_End_Angle”. For all three of these new scene variables, make sure to press the little wrench icon and select “primitive types” and then “convert to number”. You can leave the default values as 0, as we will be overriding them.

Once finished, your scene variables should look like this:



Go ahead and hit “Apply”. Then in the “Change the value of a scene variable” properties, select “Touch_Delta_X”, set the modification sign to “= (set to)” and change the value to “MouseX(“”, 0) – Touch_Start_Pos.X()”. What this will do is set “Touch_Delta_X” to the distance and direction between the mouse’s current position and the position of Touch_Start_Pos on the horizontal axis, which we’ve set to the position of the mouse click. If the value is negative, it means the mouse was dragged to the left, while if it’s positive it means the mouse was dragged to the right.

Once that is done, go ahead and hit OK to add the action. Next, repeat the same process but instead of selecting “Touch_Delta_X”, select “Touch_Delta_Y”. For the value, set it to “MouseY(“”, 0) – Touch_Start_Pos.Y()” so we get the distance and direction between the mouse’s current position and the position of Touch_Start_Pos on the vertical axis. Now Touch_Delta_Y will be positive if the mouse was dragged down, and negative if the mouse was dragged up.

Okay, now all that is left is to calculate the angle of the drag. Luckily, with a little bit of geometry we can calculate the angle using the atan2 function. Atan2 takes a direction on the X and Y, and will return a angle that points in the same direction, which is exactly what we need.

Add a new action right under “Change the scene variable Touch_Delta_Y set to …”. Search for “Scene variable” and select “Change the value of a scene variable”. In the properties, select “Touch_End_Angle” as the variable and set the modification sign to “= (set to)”. For the value, set it to the following: “ToDeg(atan2(Variable(Touch_Delta_Y), Variable(Touch_Delta_X)))”.

Tip

“ToDeg(atan2(Variable(Touch_Delta_Y), Variable(Touch_Delta_X)))” might be a tad confusing, so let’s break it down into smaller steps.

First, “Variable(Touch_Delta_Y)” and “Variable(Touch_Delta_X)” gets the scene variables with the same name that we defined and calculated earlier. “atan2” is the geometry function I mentioned and what it does it it takes the Y direction and X direction (in that order) and returns an angle that points in the same direction. However, the angle it returns is in radians, which is another way to represent angles but isn’t as intuitive as degrees, which are both easier to read and what GDevelop uses for angles. To fix this, we use the “ToDeg” function, which takes an angle in radians and converts it to degrees!

Hopefully this helps explain a bit on what this code is doing. If you are interested in how the Atan2 function works, I would highly recommend doing some research into angles and geometry, as there are resources online that explain angles and the functions around them way better than I can explain them.

Great! With that done, we now have all the data we need to send the alien in the direction of the mouse drag. Now all we need to do is set the velocity of the Physics2 behavior we added to Player_Alien and then it should work!

Add a new event and make it a sub event of “Left mouse button was released, making sure it’s right after the event we added that calculates the angle of the drag. Then, drag and drop the “Create object Player Alien at position …” action from “The boolean value of scene variable Is_Touch_Down” is true” action to this new action. We want to do this so we create a new alien after we calculated the angle of the mouse drag.

Next, add a new action and select “Player_Alien”. Then search for “Velocity” and select “Linear Velocity X”. Set the modification sign to “= (set to)” and the value to “-XFromAngleAndDistance(Variable(Touch_End_Angle), 1000)” What “XFromAngleAndDistance” does is that it takes an angle (“Touch_End_Angle” in this case) and gets the direction on the horizontal axis, the X value. We then can specify how far this X value goes with the second part, which we have set to 1000.

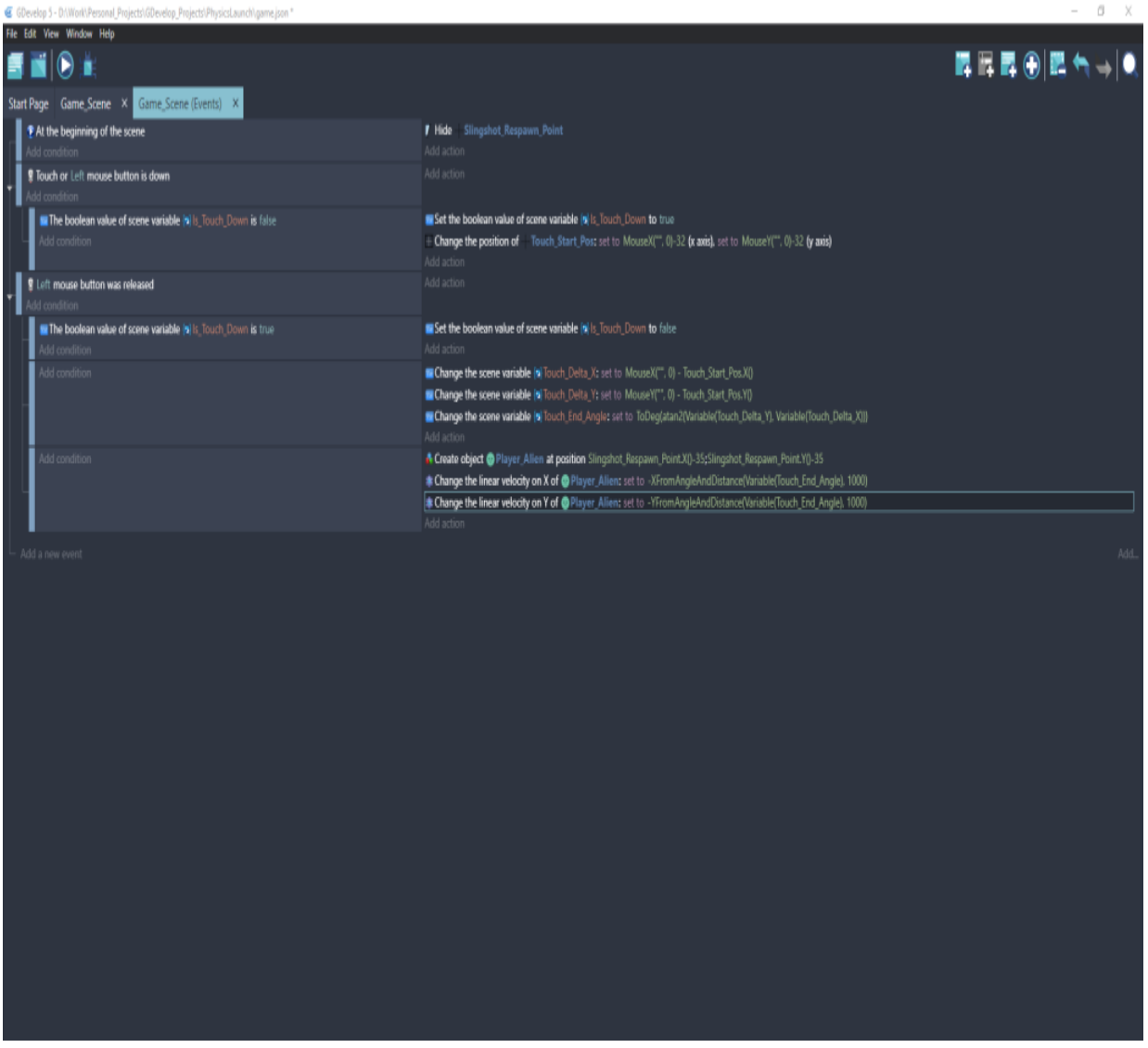
The reason we add “-” to the beginning is because we want to launch the

balls like if we are dragging them from a slingshot, which means they need to go in the opposite direction of the drag itself. This is so if we drag left, the ball will go right away from the mouse, instead of towards the mouse.

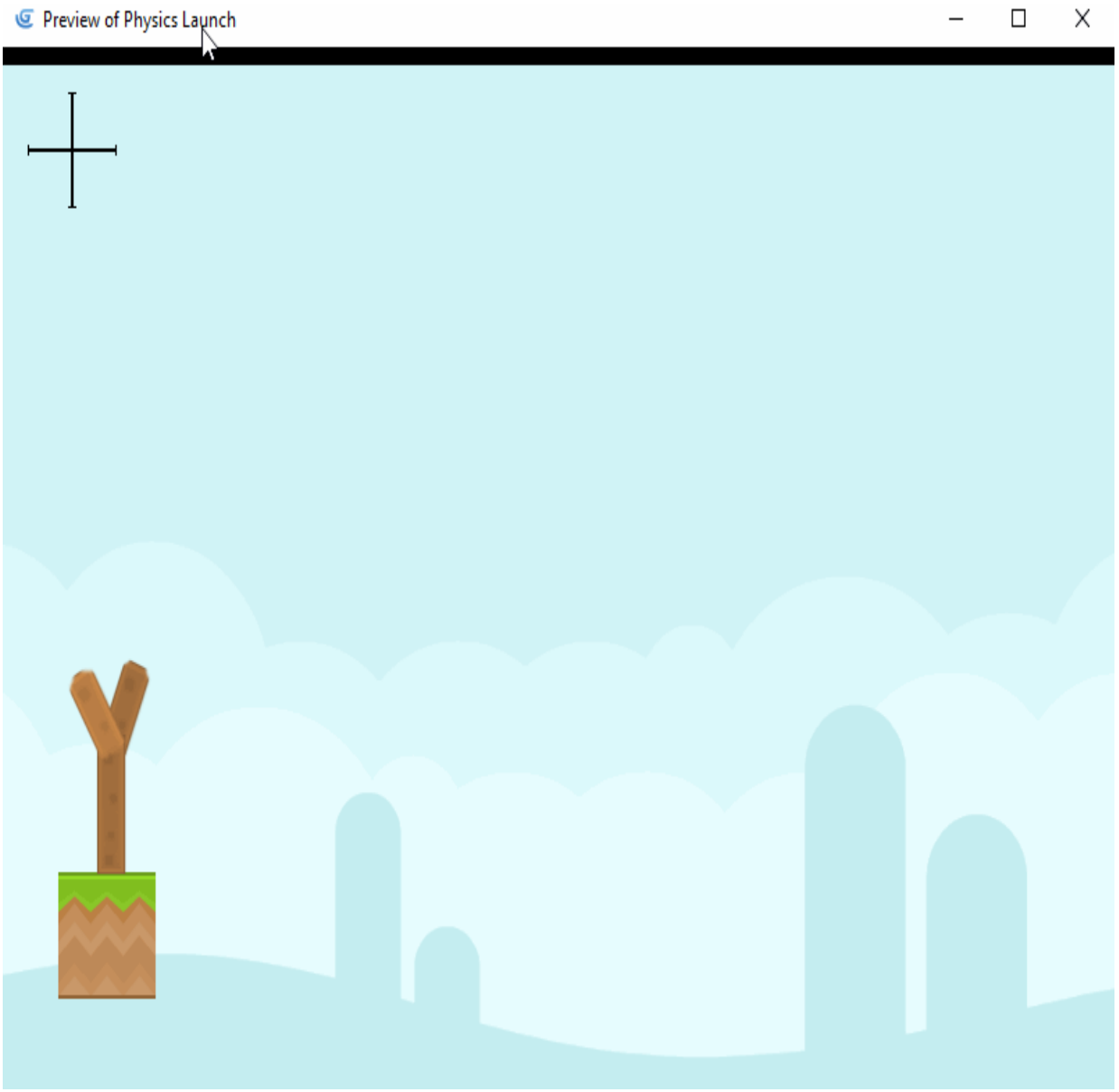
Once done, go ahead and click “Apply”.

Now we need to do the same process for the Y linear velocity. Select “Change the linear velocity on X of Player Alien …” and copy paste it right underneath. Then double click it and change the action to “Linear Velocity Y”. Set the value to “-YFromAngleAndDistance(Variable(Touch_End_Angle), 1000”. This does exactly the same thing as the previous action, but for the vertical axis, the Y value, rather than the X. Once done, go ahead and click “Apply”.

Here’s how the event graph should look currently:



With that, we’ve did it! If you play the game now, you should find that if you click and drag, the alien will be launched from the slingshot in the angle that you dragged from the starting position, which is exactly what we are looking for! It should look something like this:



Now, it can be quite hard to tell which angle the ball is going to be going towards currently which makes it difficult to aim. In part 3, we'll add a line that shows the path the alien is going to go towards, as well as refine a few things. You can find part 3 right here: Part 3.

You can find the completed project for part 2 here: Part 2 project.