# Collision Shapes

A blue object with red dots

Description automatically generatedThe CollisionShapes are what give each object in the game the ability to collide with other objects. I can create all the CharacterBody2Ds I want and give them scripts to run into each other, but without the CollisionShapes, they won’t *actually* be able to hit each other. The collision shapes give you the ability to code an event to happen when two or more objects collide with each other. It’s also important to assign a shape to the CollisionShape, otherwise it won’t know what kind of hitbox it has and will be just as useful as not having a CollisionShape at all.

The blue box around the paddle is the CollisionShape. It represents the paddle’s hitbox. Every object in the game that is “collideable” has one (i.e. the ball, the bricks, and the walls).

# Listening for Keys to be Down

Every game (mostly) takes inputs to be played. They can be a button on a controller, a bicycle wheel, a key on a keyboard, or pretty much anything you can think of. In this case, we only listen for two inputs: the left arrow key and the right arrow key. To do this, we go into Project > Project Settings > Input Map and add new input actions to the map.

A screenshot of a computer

Description automatically generated

Once the inputs are added, we can go into the script of the thing(s) that are supposed to do something when the key is pressed and make the code do all the work.

In this case, we check if the input that was given is “left”, which is associated with the left arrow key, using **is\_action\_pressed**. This function returns true every frame that the key is pressed down. A screen shot of a computer

Description automatically generatedIf the key is pressed, the paddle moves -5 pixels in the x direction. The same, but opposite, can be said for moving the paddle to the right.

# Velocity and Manipulation of Velocity

A screen shot of a computer code

Description automatically generatedVelocity is just movement in a specific direction. In the context of our game, it’s the movement of the ball. Here is the ball’s script.

On line 3, a variable, *rng*, is initialized as a new random number generator. In **\_ready**, *rng* is randomized, which allows us to get a random number… On line 8, we set the velocity of the ball using a Vector2 and by passing our random number to that Vector2 as a parameter. Now, upon starting the game, the ball is given a random velocity.

In terms of manipulating the velocity, we use **\_physics\_process** to do the ball’s physics calculations. *Collision\_info* is initialized and returns true if the object collides with something. If it does, we called **velocity.bounce()**, which literally bounces the ball back the other way, like it would in real life.

# Detecting Overlap

A screen shot of a computer code

Description automatically generated

In this project, we detect overlap when the ball enters the body area of a brick using the **\_on\_body\_entered** function, which is actually a signal that is sent by the brick to itself. It’s basically saying, if something enters my body, I’m going to do this thing. In this case, “this thing” is the brick killing itself (which is the **queue\_free** function).It also tells the ball to bounce.

# Why and How to Create Scene Resources

A screenshot of a computer

Description automatically generatedIt’s important to create scene resources because it allows you to reuse resources throughout different areas of the game. For example, I only have one brick resource, but the game is full of bricks for the ball to hit. This is because I turned the brick into a scene resource and used a script to instantiate more bricks. Basically, it’s important because it will save time by removing the need to create a new resource each time it’s needed. You can just keep reusing preexisting ones.

# Partner Discussion

The most important thing I want to bring up here is the snippet of code that was used to increase the score. Inside the Brick script is also a line that increases the score of the game, it’s within the **\_on\_body\_entered** function. The snippet is:

**get\_parent().get\_parent().update\_score()A screenshot of a phone

Description automatically generated**

Evan G. helped me figure this out, and I think I understand it. In the Scene hierarchy, I create my bricks under the node called Bricks, which is a child of the Main node. So, what I think happens is, in the Brick script, when **get\_parent()** is called, it takes it up to the Bricks node, so another **get\_parent()** is needed to get out of the Bricks node and into the Main node, which then lets me call **update\_score()**.