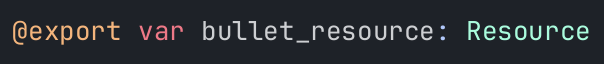
[GitHub Link](https://github.com/spencersmith24/CS315-Godot-Exercises/tree/main)

# How to Use Export Variables

A screenshot of a computer

Description automatically generatedExport variables are created with the keyword **@export**. They make it much easier to reference other scenes/resources that exist within the game. When an export variable is created, a new block in the inspector of the object is created. In this case, the player object needs to be able to shoot bullets, so we created an export variable for the bullet, and then simply dragged the bullet scene into the export variable section in the inspector. This allows us to instantiate the bullet within the player.

We also use an export variable to spawn in our enemies.

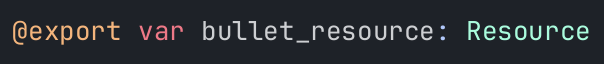
# Reasons for creating "game controller" scripts

A screen shot of a computer program

Description automatically generatedIt’s important to create a game controller script because there are some things that need to be handled, like a score, that should be separate from all the other parts of the game. It’s also used to allow one smaller part of a game, like an enemy, to talk to the game as a whole thing. For this game, we use the game controller script to handle the score with a function called **increment\_score()**, which is then called by the enemy object when it gets hit with a bullet. What happens is the enemy object notices that it gets hit, so it tells the game controller that it’s dead and that the score needs to increase.

# The process of instantiating and using resources to make new items in a game

A screen shot of a computer program

Description automatically generatedInstantiation is important because it can save a lot of time and effort. There are multiple ways to do instantiation, but in this case, we used an export variable. The export variable was given the **bullet** resource and is instantiated with **var bullet = bullet\_resource.instantiate()**. Now there’s a new bullet *somewhere* in the game, but it doesn’t know where to go. This is fixed by assigning it as a child to the player with **add\_child(bullet)**. Since this script is attatched to the player object, we don’t have to specify *who* gets the child.

We also use instantiation to spawn in our enemies. A screen shot of a computer code

Description automatically generated

# Why instantiation can be powerful

Instantiation can be powerful because it allows programmers to reuse the same resource as many times as they want, without requiring manually placing them. It’s what allows a gun in a game to shoot its bullets without crashing the game. Each bullet is loaded only when it needs to be and is removed when it’s ready. Instantiation saves a lot of time and extra work.

# Potential pitfalls of instantiation

The biggest pitfall that I can think of is poor performance. It’s important to set up a system to remove any unused instances of an object because if they are left in the game, even if the player can’t see them, they are taking up processing space. For example, if, instead of completely deleting each enemy ship in our game, we just turned them invisible, the computer would still allocate space for that object because it’s still there. In a game like ours, where the enemies are infinitely spawning, this can cause performance issues. All of the enemy instances will eventually start to clog the system memory.

# A screen shot of a computer code Description automatically generatedConversation

Aidan helped me figure out diagonal movement using a **ship\_direction** variable. Initially, my ship *could* move diagonally, but the speed would get all wonky because of the way the Vector2 values were being added together. Instead, Aidan recommended that I use a blank Vector2 that can then be added too when the movement keys are pressed. This allowed for much smoother movement, both straight and diagonally.