Intro to Programming for Games Study Sheet #9: Communicating with other Objects in Unity

1. Motivation.

```
// In Processing, we had a "global" space outside of any function to
declare variables.

// In Unity, we don't have this global space any more. When you write a
script, there's a default set of variables the script can access. These
include:

// 1. Our transform (i.e. the transform of the Game Object our script is
attached to):
transform.position = new Vector3(0, 0, 0); // Resets the position of our
game object.

// 2. Our gameObject (i.e. the Game Object our script is attached to):
gameObject.SetActive(false); // Turns off our Game Object in the scene.

// These defaults let us do a few basic things, but we need to be able to
access other objects in the scene to be able to affect them. For instance,
we might want to:

// 1. Change the color of our sprite or change WHICH sprite we're
displaying.
// 2. Move some Game Object that ISN'T the one our script is attached to.


// To do either of these things, we need to be able to access other objects
in the scene. Usually, we'll do so by storing another object in a variable
that we can then use to communicate with it.
```

2. Accessing other Components attached to your Game Object.

```
// Often, we want to be able to access other components attached to our
Game Objects so we can change their properties. Some examples:
// - The color tint of our Sprite.
// - The velocity of our physics body.
// - The position of our transform.

// The pattern we use to do this is almost always the same.
// 1. We declare a variable outside our functions to store a "reference" to
the component we'd like to talk to. (this variable will have the same type
as the component, so a Transform, SpriteRenderer, Rigidbody2D, etc.)
// 2. We fill the variable in our Start function by calling the
GetComponent function.
// 3. We use the variable in our Update function.


// Example: accessing our SpriteRenderer to change our color tint when we
press the space bar.
SpriteRenderer ourSpriteRenderer; // Declare the variable OUTSIDE Start and
Update so it's accessible to both functions.

void Start() {
      ourSpriteRenderer = GetComponent<SpriteRenderer>(); // The
GetComponent Function is how we access other components on our Game Object.
      // While you could use GetComponent directly in Update. It's a bit
slow so it tends to be better to use it once in Start to fill a variable.

      // Note: The angle brackets "<>" require a type, so make sure you
provide the Capitalized "SpriteRenderer" type.
}

void Update() {
      // -- Other Update Code
      if (Input.GetKey(KeyCode.Space)) { // If the space bar is pressed...
            ourSpriteRenderer.color = new Color(1, 0, 0); // Then change
our Sprite's color tint to red!
      }
}
```

```
// It turns out, we're effectively doing the SAME THING whenever we talk to
our Transform, the only difference being that Unity has handled steps 1 and
2 for us.

// If Unity hadn't created the "transform" variable for us, we'd write code
like this to access it:
Transform ourTransform; // We could name the variable "transform" like
Unity does, but I'll give it a different name here (more on that below).

void Start() {
      ourTransform = GetComponent<Transform>(); // The GetComponent
function works for any component type, just make sure you include the type
(capital lettters) in the <> and include the (); at the end.
}

void Update() {
      // Then in Update, we can use this new ourTransform variable the same
way we'd use the transform variable.
      if (Input.GetKey(KeyCode.Right)) {
            ourTransform.Translate(moveSpeed*Time.deltaTime, 0, 0);
      }
}
```

3. Accessing Assets.

```
// When you're making your OWN components (i.e. writing a script),
frequently you want access to Assets!

// To gain access to Assets, you can create a Public Variable of the Asset
Type (i.e. Sprite, Mesh, AudioClip, etc.)

// Then a slot will appear in the Inspector Tab. You can drag an Asset to
that slot to assign that Asset to the variable.

// Example: Changing the Sprite that's displayed based on if the mouse is
held or not.

// Public variables to store our two Sprite Assets.
// We Must Assign an Asset to these variables in the Inspector Tab.
// DON'T FORGET THAT STEP!
public Sprite mouseHeldSprite;
public Sprite mouseNotHeldSprite;

SpriteRenderer ourSpriteRenderer; // To do something with these sprite
assets, we need access to our SpriteRenderer (see the previous examples)

void Start() {
      ourSpriteRenderer = GetComponent<SpriteRenderer>(); // Using Get
Component to gain access to our SpriteRenderer.
}

void Update() {
      if (Input.GetMouseButton(0)) { // If the left mouse button (button 0)
is pressed...
            ourSpriteRenderer.sprite = mouseHeldSprite; // Then display the
Mouse Held Sprite.
      }
      else { // Otherwise...
            ourSpriteRenderer.sprite = mouseNotHeldSprite; // Display the
Mouse NOT Held Sprite.
      }

}
```

}

4. Accessing other Game Objects.

```
// So far, all of our examples have involved talking to the Game Object our script is attached to (i.e. we've been talking to OUR SpriteRenderer, OUR Transform, etc.)

// But we'd like to be able to talk to other game objects. For instance, we might want to check if we're near another object or move another object. Here are 2 common methods for gaining access to other Game Objects.

// Method #1: GameObject.Find
// The GameObject.Find function will find a GameObject in your scene that has a speciic name. Be careful though, it's a slow function, so we want to use it as little as possible.

// Often, we'll do the same thing we did with GetComponent, i.e. we'll declare a GameObject variable outside of all our functions, fill it with GameObject.Find in Start, and then use it in Update.

GameObject enemyGameObject; // Declaring a variable of type GameObject.

void Start() {
	enemyGameObject = GameObject.Find("waluigi"); // The name we provide to GameObject.Find is the name of the game object in the SCENE (i.e. the name it has in the Hierarchy tab).
	// Be VERY CAREFUL that your Game Object doesn't have a trailing space in its name. That's a good way to produce a very confusing bug (i.e. its name is "someName " but you're looking for "someName").
}

void Update() {
	// Now we can access the other Game Object's variables such as its transform or its GetComponent function.
	float distanceFromEnemy = Vector3.Distance(transform.position, enemyGameObject.transform.position); // Store the distance between us and the other game object in a variable.
	if (distanceFromEnemy < 0.1f) { // If we're close to the other game object..
		Debug.Log("OUCH!"); // Do something!
	}
}
```

```
// Method #2: A public variable.

// Just like how we gained access to Assets, we can make a public
GameObject variable. Then a slot will appear in the inspector. We can
assign Game Objects to that slot by dragging them from the hierarchy to the
slot.
public GameObject mouseDownObject; // Make sure to assign an object to this
in the inspector.

void Update() {
      // Turn the object on or off depending on if the mouse is pressed.
      if (Input.GetMouseButton(0)) { // If the left mouse button is
pressed...
            mouseDownObject.SetActive(true); // Turn on the Game Object!
      }
      else { // Otherwise...
            mouseDownObject.SetActive(false); // Turn Off the Game Object!
      }
}
```

5. Common Mistakes/Confusions

```
// Some of the most common points of confusion in Unity is the difference between "types" and "variables".
// This is a big point of confusion because Unity gave a lot of default variables the SAME name as their type.

// For instance. The transform variable has the TYPE Transform (note the capitalization)
// If we were to declare that variable ourselves, it would look like this:
Transform transform; // Read it as: create a variable of type Transform NAMED transform.

// Unity did the same thing with the variable named "gameObject"
GameObject gameObject; // Read it as: create a variable of type GameObject NAMED gameObject.

// This becomes confusing because it's similar to if we declared a variable of type int NAMED Int
int Int; // Probably not the best name for an integer variable.

// Mistake #1:
Transform.position = new Vector3(0, 0, 0);
// Here, we're talking to the Transform TYPE (because of the capital T). To access the position variable, we need to talk to a variable that HAS the type Transform (usually the variable named transform).

// Mistake #2:
enemyGameObject = gameObject.Find("waluigi");
// Here, we're making the opposite mistake. The GameObject.Find function is a "static" function, meaning it belongs to the TYPE GameObject and not to any variable that has type GameObject (we'll learn more about static later).
// So here we'd need to use a capital G for GameObject.Find.
// The difference between Game Objects, Components, and Assets is another big point of confusion for new Unity Programmers.
// (This is made worse by the fact that Game Objects can exist in your scene as Scene Objects AND exist in your Assets folder as Prefabs. More on that later.)
// Here's a quick handy guide to remember the differences.

// 1. Game Objects are objects in your Scene. Everything listed in the Hierarchy Tab is a Game Object (Specifically a Game Object in your Scene).

// In Code, you can store a reference to a Game Object in a variable of type GameObject.
GameObject someGameObject = GameObject.Find("someGameObject");

// 2. Components are ATTACHED to Game Objects. They can be pre-made by Unity (such as Transforms, Rigidbodies, Renderers) or written by you (i.e. Scripts!). The Hierarchy Tab doesn't display any Components, but if you select a Game Object in the Hierarchy Tab, the Inspector Tab will show you all Components attached to that Game Object (By default, every Game Object will have a Transform Component).

// In Code, you can store a reference to a component in a variable of that component's type (i.e. the name of one of your scripts or Transform, Rigidbody2D, SpriteRenderer etc.)
SpriteRenderer someRenderer = GetComponent<SpriteRenderer>();

// 3. Assets are files in your Assets folder that Unity recognizes as a specific type of data (i.e. Sprites, Audio Clips, TextFiles). Assets are found in the Project Tab (and also in your Assets folder). Assets are NOT Components. This is important. You don't attach a "Sprite" to a Game Object, you attach a SpriteRenderer Component that then keeps track of a Sprite Asset.

// In code, you can store a reference to an asset in a variable of that Assets's type (i.e. Sprite, Texture, Mesh, AudioClip, TextAsset, etc.)
public AudioClip winSoundEffect; // Assigned in the Inspector Tab.

// A quick way to remember it is to remember this process:
// 1. Game Objects are placed in your scene!
// 2. Components are attached to your Game Objects!
// 3. Assets are assigned to your Components!
```