Individual Notes

Part 0 - Unity

- Unity Hub is a place to manage projects and multiple installations of Unity
- Add the Microsoft Visual Studio community as an additional module and WebGL build support
- Unity is a game engine while Unreal is another well-known game engine
- Game engines provide a visual interface for creating games.

Part 1 - Intro

- Focusing on basic fundamentals of C# programming
    - Talking about variables
    - Talking about if statements and creating methods
    - Playing around with triggers and colliders
    - Looking at references within code, accessing properties in components

Part 2 - Game Design

- Game design is important to establish guidelines and make better decisions throughout the project.
- Start with a rough gameplay overview screen to visualize the game.

Part 3 - Introducing Methods

- Methods are also called functions, they execute blocks of code to make a game do things
- We can use existing methods in Unity or create our own
- Methods are created by giving them a name and defining what they should do
- Start and Update are methods available in Unity
- Using statement refers to the namespaces in Unity that can be used in the project

Part 4 - Transform.Translate

- This method allows us to move anything with a transform
- Transforms are Unity game objects that have the transform component
- Transforms are handled using x, y, and z as axis and Transform.translate can change the value

Part 5 - Introducing Variables

- Variables can be thought of as boxes that store and manipulate information
- Variables have a name (e.g. hit points), data (e.g. 20), and type (e.g. int for whole numbers)
- Common variable types: int for whole numbers, float for fractional numbers, double for high precision fractional numbers, bool for true/false values, and string for sequences of characters
- To declare a variable, the type and name is specified, followed by an equal sign and a value and a semicolon at the end of the statement
- Variables can be declared and initialized in the class, above the start method

Part 6 - How to use serialize field

- Use Serialize Field attribute to make variables available in the inspector
- You can use the inspector to visually see and change variables
- The value in the inspector becomes the overriding value even if the initial value is different in the code

Part 7 - Using Input.GetAxis()

- The input system is used to convert the player's physical actions (e.g., button press or key press) into information for the game to understand
- Unity has two input systems, an old system and a new system. The old system is being used in this project because it's easier to understand and use
- Input Manager in the Project Settings shows the axes (horizontal, vertical) that can be used to control the game. The keyboard (arrows and WASD) and joystick/gamepad can be used as input devices
- The name of the axes must be exact or the input won't work.
- The horizontal and vertical axes go from -1 to 1

Part 8 - Using Time.deltaTime()

- Code that is inside the update method takes up a lot of resources because it executes every frame
- To make code frame rate independent, Unity uses the concept of time, delta time.
- Delta time tells us how long each frame took to execute.

- By multiplying the values by delta time, we can make sure that the game behaves the same on all devices, irrespective of the frame rate.
- In code, the steer amount and move amount are multiplied by delta time to make them frame rate independent.

Part 9 - Colliders & Rigidbodies

- The objects need colliders to have bumping behavior
- The green line around the object represents the collision area
- Triggers are different from colliders, allowing objects to pass through but sending a message when entered, exited or still inside

Part 10 - Using OnCollisionEnter2D()

- This method is called whenever two objects collide
- Each object with a collider gets their own OnCollisionEnter2D() method
- The "Collision2D" information is passed to the method, which provides information about the colliding object