

# TileVania Section



Rick Davidson





# Section Intro - TileVania



Rick Davidson 



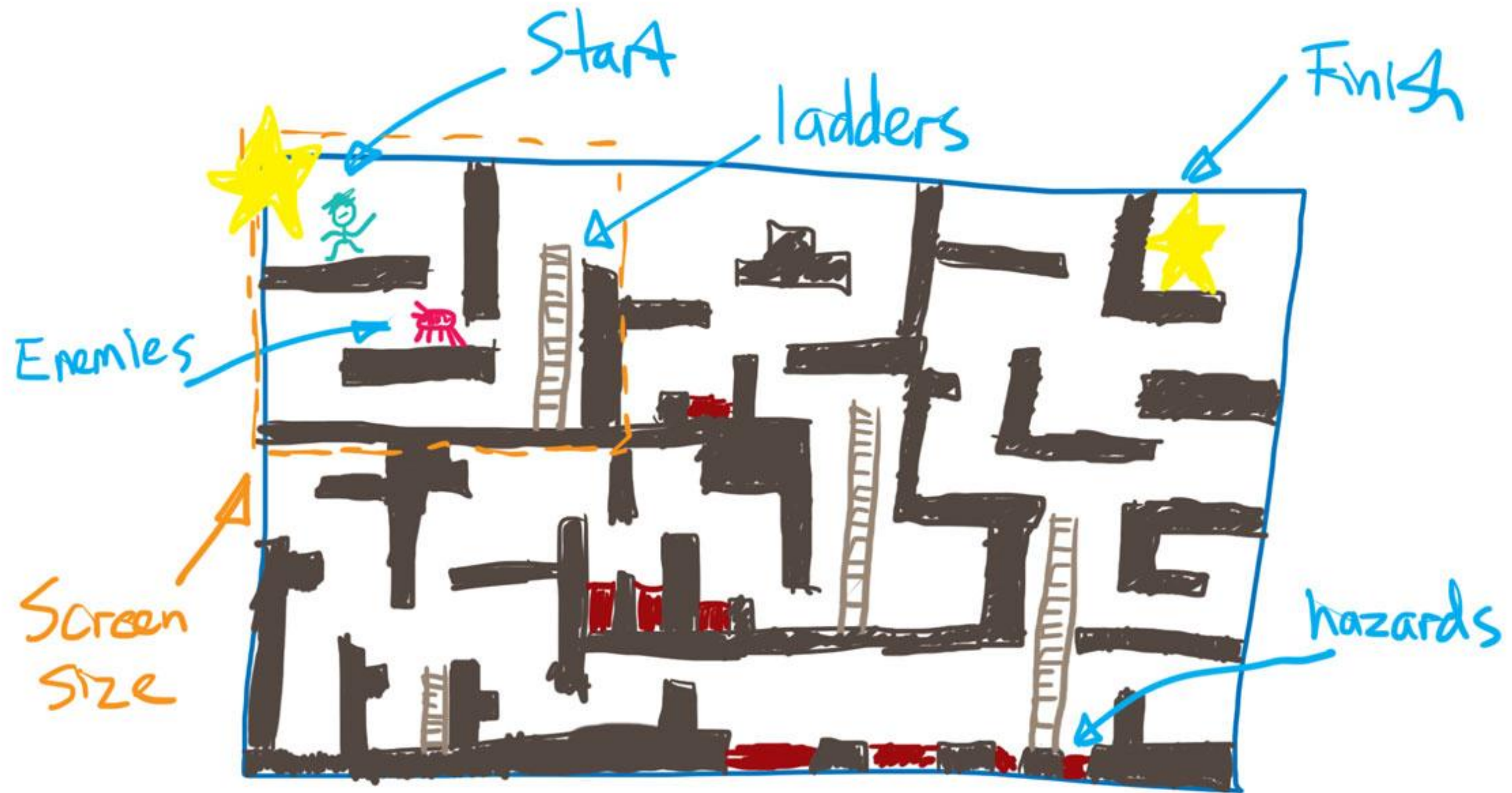
# Game Design - TileVania



Rick Davidson

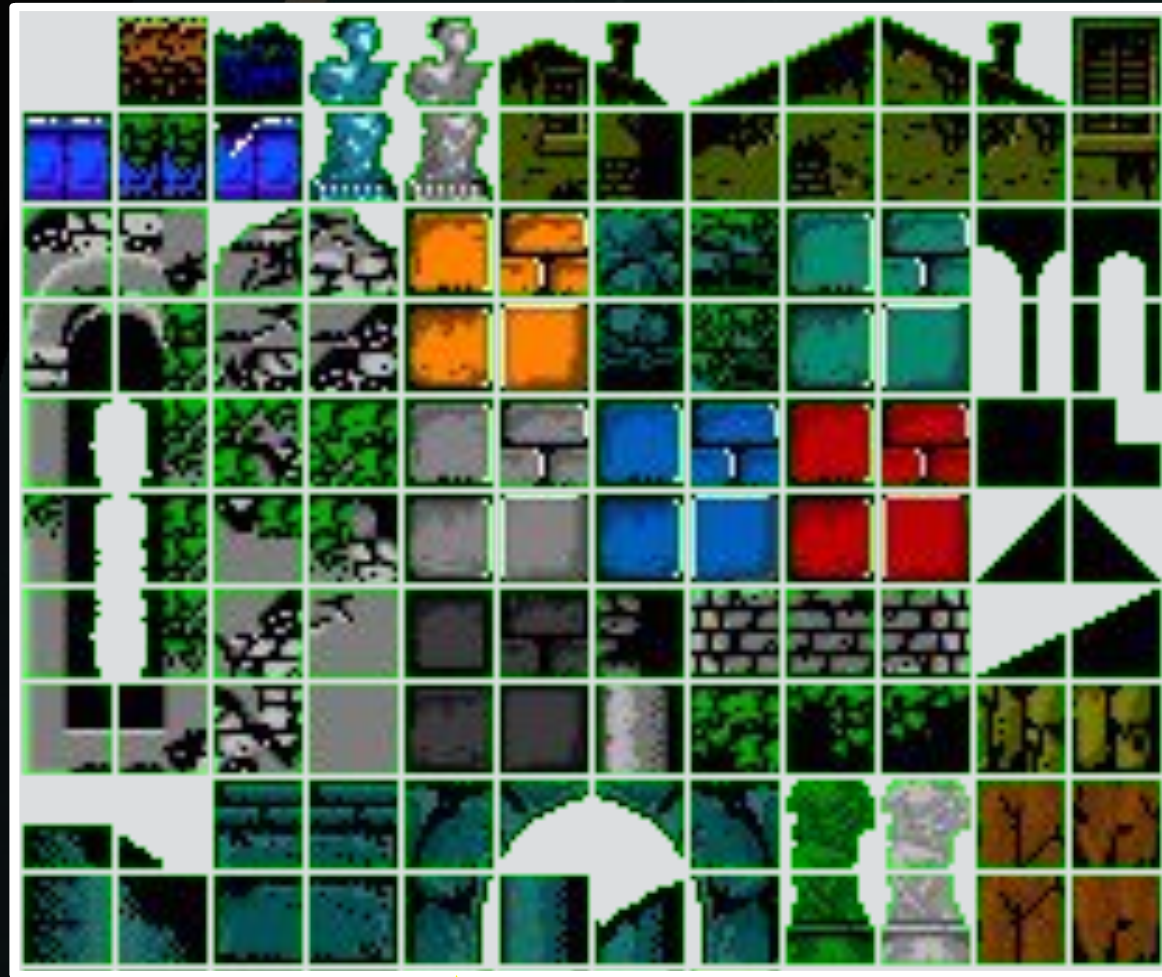


# Core Gameplay Overview





# Castlevania



Using Sprite Sheets  
to make Tile-based  
levels



Mega Man



Spelunky



# TileVania Game Design

## Player Experience:

Under pressure

## Core Mechanic:

Run and jump

## Core game loop:

Get from A to B by  
navigating platforms and  
avoiding traps and enemies





# Game Theme

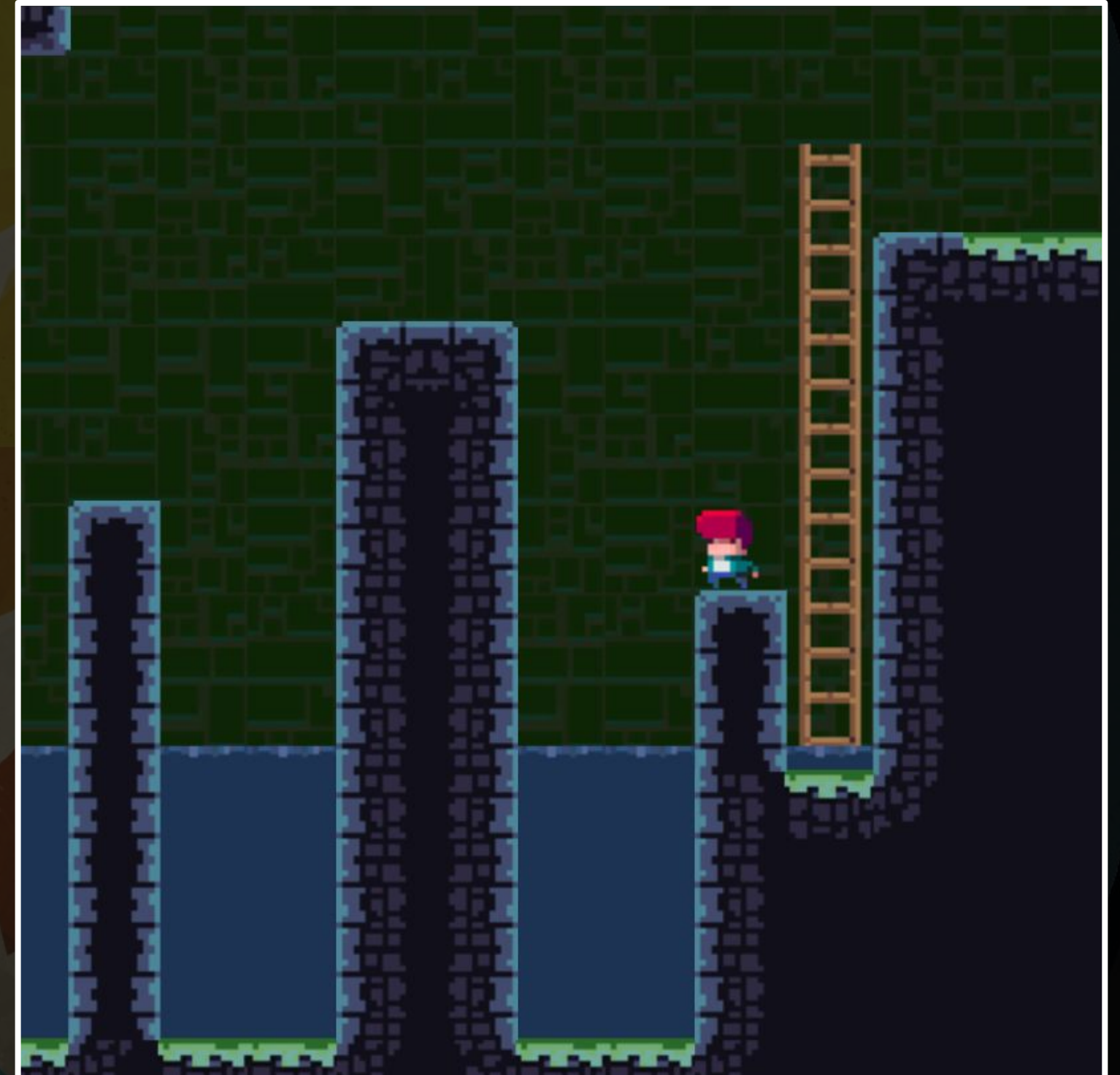
- Escape. You're a prisoner. Clock is ticking.





# MVP Gameplay Features

- **Character movement:** Player can run and jump
- **Traps / obstacles:** Instantly kill the player
- **Level loading:** A way to finish the level and start next level
- **Countdown timer:** Some system to create time urgency





# Find Your Art Assets

- We will provide basic sprite sheet assets for world and character.
- Here is the time for you to flex your artistic muscles, or find an asset pack you like.
- You will need sprite sheets for:
  - World tiles
  - Character
  - Enemy





# How To Slice Sprite Sheets





# Slice Those Sheets

- Slice up your sprite sheets so that we have clean, individual assets for:
  - Background tiles
  - Platforming tiles
  - Character animation
  - Enemy animation
  - Anything else you might have cooked up





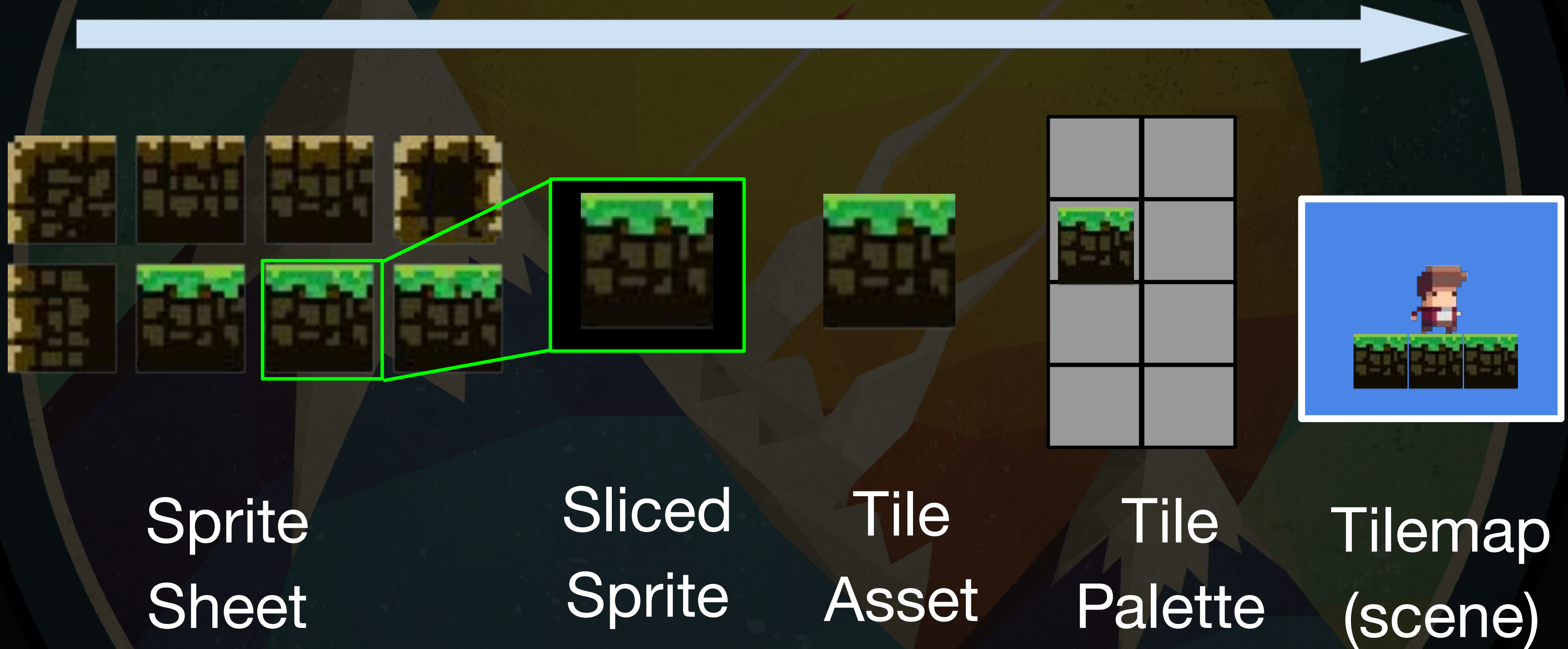
# Introduction To Unity Tilemap



Rick Davidson 



# Tilemap Pipeline





# Create A Simple Platform

- Create Platforms Tilemap
- Create Tile Palette
- Format Tile Palette (rearrange the tiles if need be)
- Paint a simple platformer layout





# Unity Tilemap With Layers



Rick Davidson 



# Do You Remember?

- Create a new tilemap, call it Background Tilemap
- Add our background tiles into the Palette
- Paint some background tiles
  - Don't worry if the order sorting is wrong, we'll be going through that together





# How To Use Unity Rule Tiles



Rick Davidson



# Finish Your Rule Tiles

- Finish setting up your rule tile so that all shapes and positions work and look good
- Using your rule tile, create a couple of simple platformer layouts





# Create Player Idle Animation



Rick Davidson





# Terminology

- **Animator Component** - Assigns animations to GameObjects through an Animator Controller
- **Animator Controller** - Arrangement of animations and transitions (state machine).
- **Animation** - Specific pieces of motion
- **Sprite Renderer** - displays the 2D sprite on screen



# Set Up Your Character's Idle

- Import spritesheet and slice
- Add sprite renderer to Player
- Create idle animation clip
- Create Character Animator Controller
- Add idle animation to Animator Controller
- Add Animator to Player
- Assign Character Animator Controller to Player





# Animation States & Transitions



Rick Davidson



# Implement Climbing

- Implement climbing animation state
- Create transition from idling
- Create a bool for climbing state and test





# Using Prefabs In Unity



Rick Davidson 



# What Are Prefabs

- Prefabs are game objects which we have turned into reusable templates
- The original template is called the **Prefab** and the copies we add into our scene are called **Instances**
- Turning a game object into a prefab means we can easily load it into any scene in our game



# Experiment With Prefabs

- We won't be saving our work in the lecture so be sure to have fun experimenting with prefabs as I'm explaining them





# Composite Collider For Tilemap



Rick Davidson



# Make Your Player Fall And Land

- Figure out what components we need to add to the player so when we push play he falls and lands on the ground.
- Bonus points: figure out how to ensure the player doesn't fall over





# Set Up Input System



Rick Davidson 



# Set Up New Input System

- Add Player Input Component
- Create settings asset
- Add a Jump action and bind it to space on keyboard and Button South on gamepad
- Create PlayerMovement.cs script and attach to player





# Horizontal Player Movement



Rick Davidson



# One Simple, One Tricky

- Simple:
  - Add something to our script to give us control over how fast the player runs
- Tricky:
  - Try to figure out how to make the character behave normally on the y axis (ie. respond to gravity, not fly up and down in the air, not get stuck)





# Flip Player Sprite



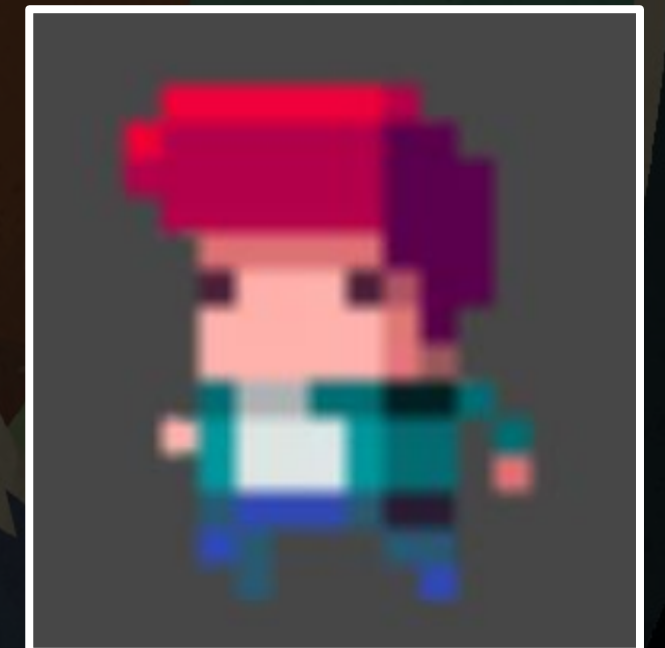
Rick Davidson





# Here's Our Flippin' Logic

- If the player is pushing right, the velocity will be positive. If left, then negative.
- If moving right, we should face right. If left, face left.
- We can change the facing direction (right or left) by changing the localscale using +ve or -ve value.
- Only change facing direction if moving, so weird things don't happen when velocity is zero.





# Animation State In Code



Rick Davidson 



# Finish The Run State

- Make the `isRunning` bool true if running and false if not.
- HINT: Use the code we created for flipping our sprite





# Jumpy Jump



Rick Davidson





# Tune Your Jump

- Using gravity and jump speed, tune your jump so it feels good.
- Decide on how many tiles you want your player to be able to jump up.
- Decide on how far you are okay with your player floating horizontally when jumping.





# Jump If `IsTouchingLayers`



# Layers

- Layers are useful if we have the same functionality across multiple GameObjects.
  - Eg. Ignored by camera, not clickable, collision check
- To stop jumping anytime we use `Collider2D.IsTouchingLayers()`



# Stop Your Player Multi-Jumping

- Finish the logic to stop our player multi-jumping.
- You will need to use:
  - `Collider2D.IsTouchingLayers`
  - `LayerMask.GetMask("layer")`





# Climb That Ladder





# Mega Challenge

- Implement ladder climbing
- Remember:
  - Set up your Climbing tilemap and Climbing layer
  - Create **ClimbLadder()** method
  - Check for touching ladder
  - Apply climb velocity to y axis
- In next lecture we'll set animation and fix sliding





# Stop Sliding On Ladder



# Stop The Player Sliding

- Set starting gravity on player Rigidbody to the Rigidbody's current gravity.
- Set gravity while on ladder to zero.





# Ladder Climb Animation



# Climbing Animation

- Set up the climbing animation so it plays when the player climbs the ladder. If the player is stationary on the ladder, the climbing animation should not play.
- Good luck!





# Cinemachine Follow Camera



# Cinemachine Confiner



# Confine Your Camera

- Add enough sandbox game level for your camera
- Add collision to your background layer
- Update your physics layers





# State-Driven Cameras





# Add A Ladder Camera

- Add an additional state camera for climbing ladders.
- Make sure the blending to and from the camera is how you like it to be.





# Using Physics Material 2D





# Playing With Friction

- Stop the player from gripping to the walls by applying a new Physics Material 2D to the player.





# Prevent Wall Jump





# Stop The Player Wall Jumping

- Prevent the player from multi-jumping when touching the wall.
- HINT: Add a second collider to represent feet.





# Set Up Our Enemy





# Move & Flip Enemy





# Finish The Flippin' Code

- Finish our code to make the enemy flip directions.
- HINT: Look at the player movement script for clues.





# Player Mortality





# Enemies = Bad, Mmmkay

- When the player collides with an enemy, disable the player controls.
- My approach:
  - Using an **isAlive** bool
  - Creating a **Die()** method





# Player Death State





# Do Something Nifty For Death State

- When the player collides with an enemy do something interesting such as:
  - Fling the player across the map
  - Change player colour
  - Add some screen shake
  - Or some other wacky thing
- Share in the discussions what you've come up with!





# How To Create Hazards





# Place Some Hazards In Your Level

- Experiment with different objects and place some hazards in your level.





# Instantiate Bullet From Gun





# Spawn Our Bullet

- Use `Instantiate()` to spawn our bullet at our player's gun.





# Implement Bullet Behaviour





# Create Some Levels





# We Need 3 Levels

- Make sure everything you need is prefabbed
- Duplicate your existing level so that you have 3 levels
  - Level 1: A single room with a jump over spikes
  - Level 2: A Triple room with ladders and enemies
  - Level 3: As big as you like, introducing bouncing and gaps to jump as well as ladders and enemies





# Level Exit Portal





# Coroutines Explained

- Coroutines are another way for us to create a delay in our game.
- The core concept to understand is that we start a process (ie. Start Coroutine) and then go off and do other things (ie. Yield) until our condition (eg. we've waited 2 seconds) is met.





# Coroutines Code

We call:

```
StartCoroutine(NameOfMethod());
```

Our method:

```
IEnumerator NameOfMethod()  
{  
    yield return new WaitForSecondsRealtime(time);  
    // Anything you want to do after waiting  
}
```





# Load The Next Level

- When the player touches the exit, load the next level
- For bonus points, wait for a second before loading the next level





# Game Session Controller





# Our Goal

- Player has X lives
- Restart scene when player dies
- When all lives are lost, game over
- Restarting the game resets lives and score.



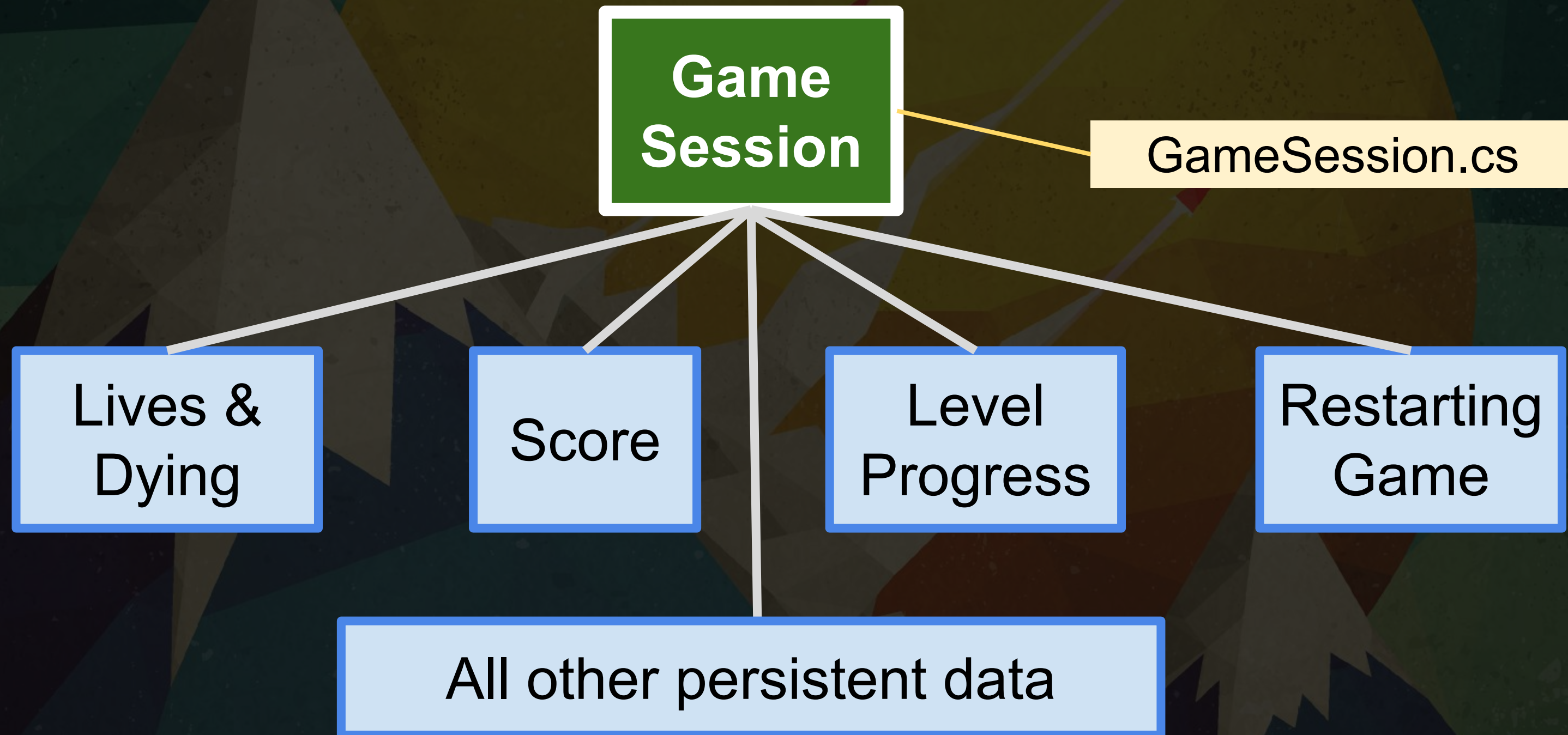


# The Problem

- After death, reloading the scene re-instantiates the Player, and all other objects.
- Any progress with the player or scene is lost.



# Game Session Object Is Responsible





# Process Player Death

- Reduce number of lives
- Load current scene





# Make A Coin Pickup





# “Pick Up” The Coin

- When the player touches the coin, destroy it so it disappears.





# Coin Sound Effect



# Play Our Sound Effect

- Use `PlayClipAtPoint()` to play a sound effect when you pick up the coin.





# Persistent Score & Lives



# Increase And Persist Our Score

- When we pick up a coin we want to increase our score by X amount (eg. 100 points)
- We want the ability to set different coins to different points value
- We want that to display on the screen
- We want it to persist when the player loses a life
- We want it to return to 0 when we start a new game





# Scene Persist



# Create Scene Persistence

- Create a public method that we can call wherever we need to reset our scene persist
- Playtest to make sure there are no edge cases





# Prefab Variants



# Create Some Prefab Variants

- Experiment with prefab variants
- Take one or more aspect of your game and create Prefab variants for them

