TL6 Presentation

Version Control

Elizabeth, Ben, Ryan, Patrick, Moe, Julia, Devon, Sohan

Contents of this Presentation

1) Oral Exam	6) Copyright
2) Test Plan	7) Building in Unity
3) Prefabs	8) Building in Godot
4) Dynamic Binding	9) Remaining Items
5) Patterns	10) Check List for Next Thursday

01

Oral Exam

Everything you need to know

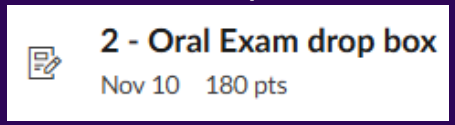
Patrick - Penguine

Oral Exam Requirements

Where to find:

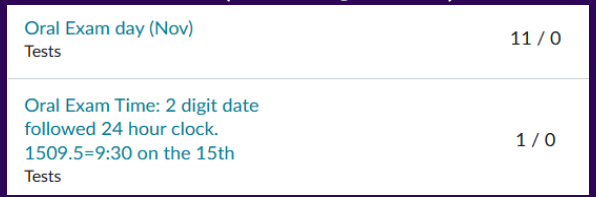
Canvas

- Oral Exam drop box



(links to marking key inside assignment)

- Date / Time (under grades)



What to do:

JEB 324 – Arrive 15 minutes early

Fill out marking key prior to exam

Patrick - Penguine

Oral Exam Marking Key

Name _____ Team _____ TL ☐ Date _____ Time _____

Fill in the underlined areas (and the boxes above), now but don't write on the remainder of this form.

Contribution: Briefly describe what your feature(s) is/are: _____ _____ Walk me through your Gantt chart. How long did this take? How long did you estimate it would take? What did you learn about your skill as an estimator? Run your game and point out places where your code is called and run. (I will cycle through asking you this question and the next one until you either run out of interesting things to talk about or it is clear that you have made an above average contribution.) Show the C++/C# code that was run. Walk me through the methods called from the time it enters your section of code.	/10
Technical: Walk me through your test plan. Give an example where a test case later found a bug in your code by things a teammate added later. (Or explain why you chose a test case specifically because you wanted to ensure that a teammate would know if they broke your code.) _____ Pick a Prefab you have created that is documented well in a separate readme file. (I will point to several places in your code documentation and ask) What question where you trying to answer here? Who do you anticipate would be asking that question? What other questions might this person need the answers to? Prefab Name: _____ Show me a class in your code where there could be either static or dynamic binding. Write some mock code on this paper showing how you would set the static type and dynamic type of a variable. Super Class: _____ Sub Class: _____ Virtual Function: _____ Choose a dynamically bound method. What method gets called now? Change the dynamic type. What method gets called now?	/4 /3 /3
Pick a statically bound method. Which one would be called in each of the two previous cases? Show me an example of reuse in your code where you violate copyright law. How does it violate copyright?_____ What did you have to do to integrate it with the code you wrote? What are the legal implications if you market your code with the re-used portion? Use fair use argue that you can use this anyway. 4. One big or two small, well-chosen patterns. Small Patterns = {Singleton, Private Class Data} Which patterns did you choose? 1._____ 2._____ Why did you choose each pattern? (Justify your use of it). Draw the class diagram for your pattern(s). Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?	/4 /4

Location:
Oral Exam Drop Box
(Canvas)

Instructions say fill in the underlined, but we are filling in everything

Patrick - Penguine

Contribution Section:

Contribution: Briefly describe what your feature(s) is/are:

/10

_____

_____

Walk me through your Gantt chart. How long did this take? How long did you estimate it would take? What did you learn about your skill as an estimator?

Copy and paste image

Run your game and point out places where your code is called and run. (I will cycle through asking you this question and the next one until you either run out of interesting things to talk about or it is clear that you have made an above average contribution.)

Video of running the game and point out your code (canvas)

Show the C++/C# code that was run. Walk me through the methods called from the time it enters your section of code.

Submit a zip (canvas)

These are important and not explicitly said on the Oral Exam Prework

Patrick - Penguine

Technical Portion of Prework

Test Plan

Test plan walkthrough, provide example of a test that led to the discovery of a bug.

Dynamic Binding

Provide example of dynamic binding from your code, answer the questions on Prework.

Copyright

Provide an example of something you used that violates copyright.

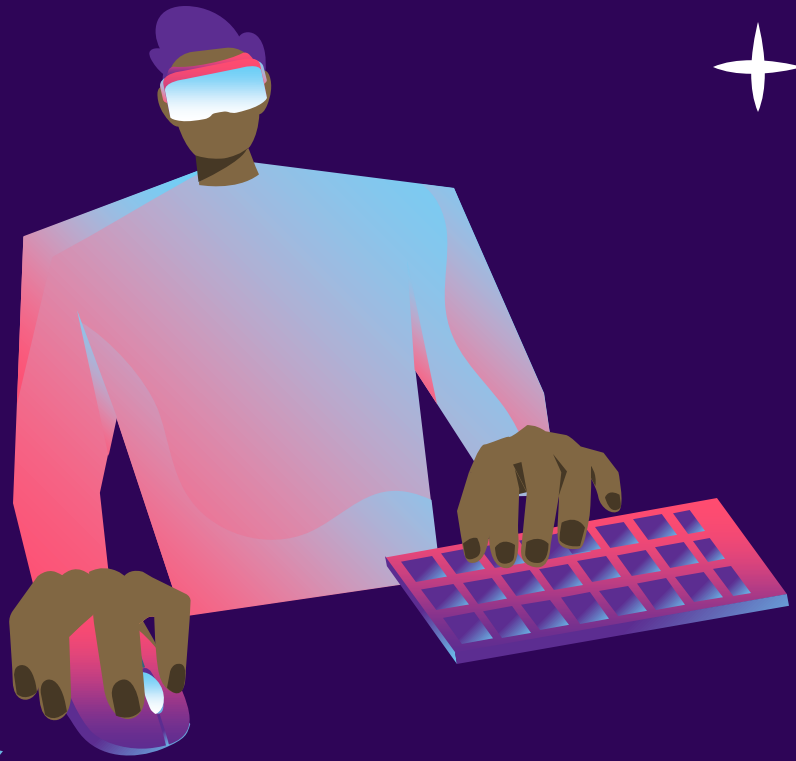
Prefab

Pick a well-documented prefab and answer the questions on Prework.

Patterns

Justify your use of patterns in your code, provide examples and class diagrams (Ensure they match up with the structure for your pattern).

Patrick - Penguine



02

Test Plan

Test runner

Elizabeth- Make it Awesome

30-35

Your test plan should consist of at least 30 tests.

Testing Pre Work:

Walk through your test plan. Give an example where a test case later found a bug in your code by a thing a teammate added later. (Or explain why you chose a test case specifically because you wanted to ensure that at teammate would know if they broke your code.

Technical:

Walk me through your test plan. Give an example where a test case later found a bug in your code by things a teammate added later. (Or explain why you chose a test case specifically because you wanted to ensure that a teammate would know if they broke your code.) /4

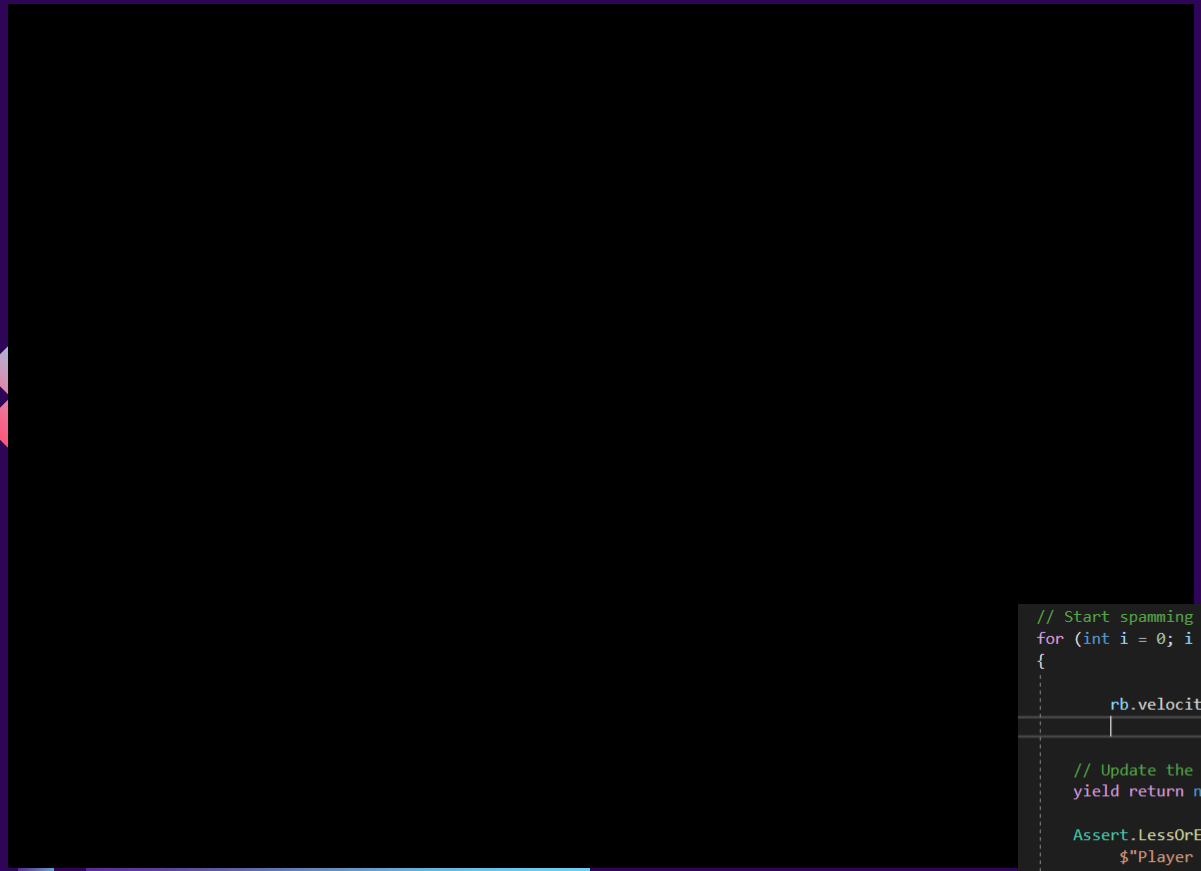
Ceiling/Jump Force Test – Does the player go past the ceiling. Wanted to test this because I wanted to both ensure the player can't go out of bounds through the ceiling. And The Jump force part was for when the powerup are added to the game, they could alter the jump force and knowing the maximum jump force before it breaks is essential to implementing the powerups correctly. /3

Healthbar – Does the math clamp work even when health is -.5 or 1.5. Wanted to test this to make sure the fill bar does not go outside of the border image and the fill works correctly. Also, to make sure the calculation of the current health clamps correctly. IF multiple enemy collisions happen we want the health bar to act appropriately, in the test when the math clamps to zero and an enemy collides (Enemy wasn't there when the test was written) the health shouldn't go out of bounds. Or when healing items are implemented we don't want to go over the maximum. /3

High Speed Collision –What speed does the player go past the enemy without a colliision. Wanted to test this to see if there is a reasonable speed where the player can zoom past the enemy. If our powerups add speed to the Player we want to make sure that collisions still work as intended. It also showed that the collisions needed to be reworked, or the enemy. As the damage is inconsistent if you stay against the enemy and never leave its collider.

Player Jump - Rapidly simulate movement and jumping by the player. Wanted to ensure the inputs still work under high stress. And that random movement does not allow the player to reach unintended locations. If other colliders are added, can the player use those to get to places they shouldn't. For example, the enemy was added after the test, and the Player is able to jump on top of the enemy and stay on top without taking damage, then they can essentially fly on top of the enemy while the enemy still tries to path find. /4

Test 1 - Ceiling Test



Why Test This?

- Remove the grounded check and let the player fly.
- Ensure the ceiling cannot be passed just by spamming the jump button.

```
// Start spamming jumps for 500 frames
for (int i = 0; i < 500; i++)
{

        rb.velocity = new Vector2(rb.velocity.x, playerMovement.jumpForce);


    // Update the frame
    yield return new WaitForFixedUpdate();

    Assert.LessOrEqual(playerObject.transform.position.y, ceilingY,
         $"Player exceeded ceiling boundary at Y = {playerObject.transform.position.y}");

}
```

Test 2 – Jump Force Test

Why Test This?

- Ensure the Player cannot exit the boundaries.
  - When power-ups are added, jump force might increase.
  - Want to know the maximum jump force that can be added.

```
// Start spamming jumps until going out of bounds
while (true)
{
    // Check if the player is grounded before jumping
    if (playerMovement.grounded)
    {
        Debug.Log($"Jump Foce: {jumpForce}");
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
        playerMovement.grounded = false;
        jumpForce += 5; // Increase jump force
    }

    // Update the frame
    yield return new WaitForFixedUpdate();

    // Check if the player has gone out of bounds
    if (playerObject.transform.position.y > -33)
    {
        break; // Exit the loop if the player has gone out of bounds
    }
}

// Assert that the player went out of bounds
Assert.IsFalse(playerObject.transform.position.y > -33, "Player went out of bounds.");
```

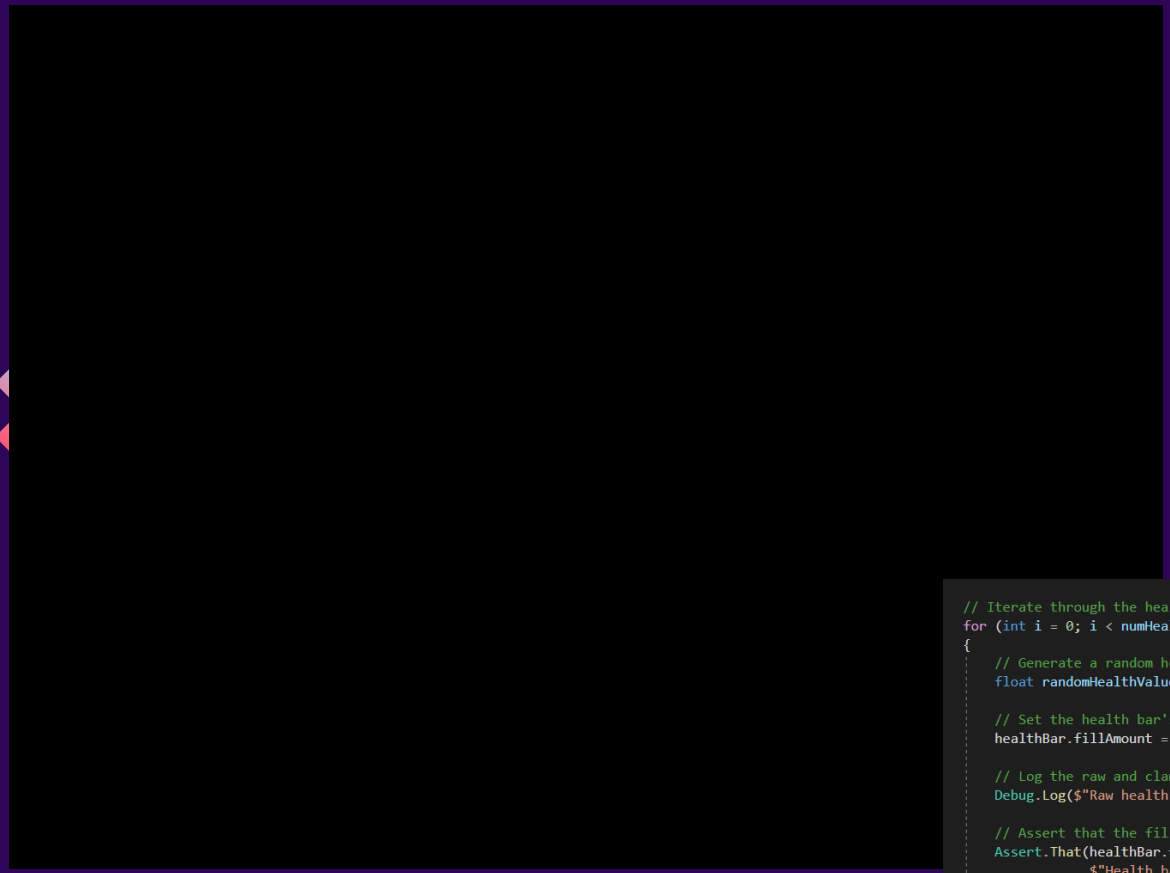
Test 3 – Enemy Collision

Why Test This?

- Stress test the collider
- Test how the player's collider interacts with the enemy
- Does the health update correctly?

```
while (true) // Loop indefinitely until collision or position condition is met
{
    // Reset player to the initial position before each new speed test
    player.transform.position = new Vector3(-116, -43, 0);
    collisionDetected = false; // Reset the collision flag

    // Apply velocity to the player to move right into the enemy
    for (int frame = 0; frame < 500; frame++) // Simulate movement for 500 frames
    {
        player.transform.position += Vector3.right * initialSpeed * Time.deltaTime;
        yield return new WaitForSeconds(waitTime); // Wait for a specified time to allow for collision

        // Check for collision
        if (Physics2D.IsTouching(player.GetComponent<Collider2D>(), enemy.GetComponent<Collider2D>()))
        {
            Debug.Log($"Collision detected at speed: {initialSpeed}");
            collisionDetected = true; // Collision occurred
            break; // Exit the loop if a collision is detected
        }

        // Check if player has moved past -108 on the x-axis
        if (player.transform.position.x > -108)
        {
            Assert.Fail("Player moved past -108, breaking collision");
            yield break; // Exit the test if the player moves past -108
        }
    }

    // If a collision was detected, increase the speed for the next attempt
    if (collisionDetected)
    {
        initialSpeed += speedIncrement; // Increase speed for next iteration
    }
    else
    {
        Debug.Log($"No collision detected at speed: {initialSpeed}");
        break;
    }
}

// If collision was never detected, the test fails. Collider is broken
Assert.IsFalse(collisionDetected, $"Collision was detected at speed: {initialSpeed}. Collider Broken.");
yield return null; // Ensure the test completes
```

Test 4 – Health Bar

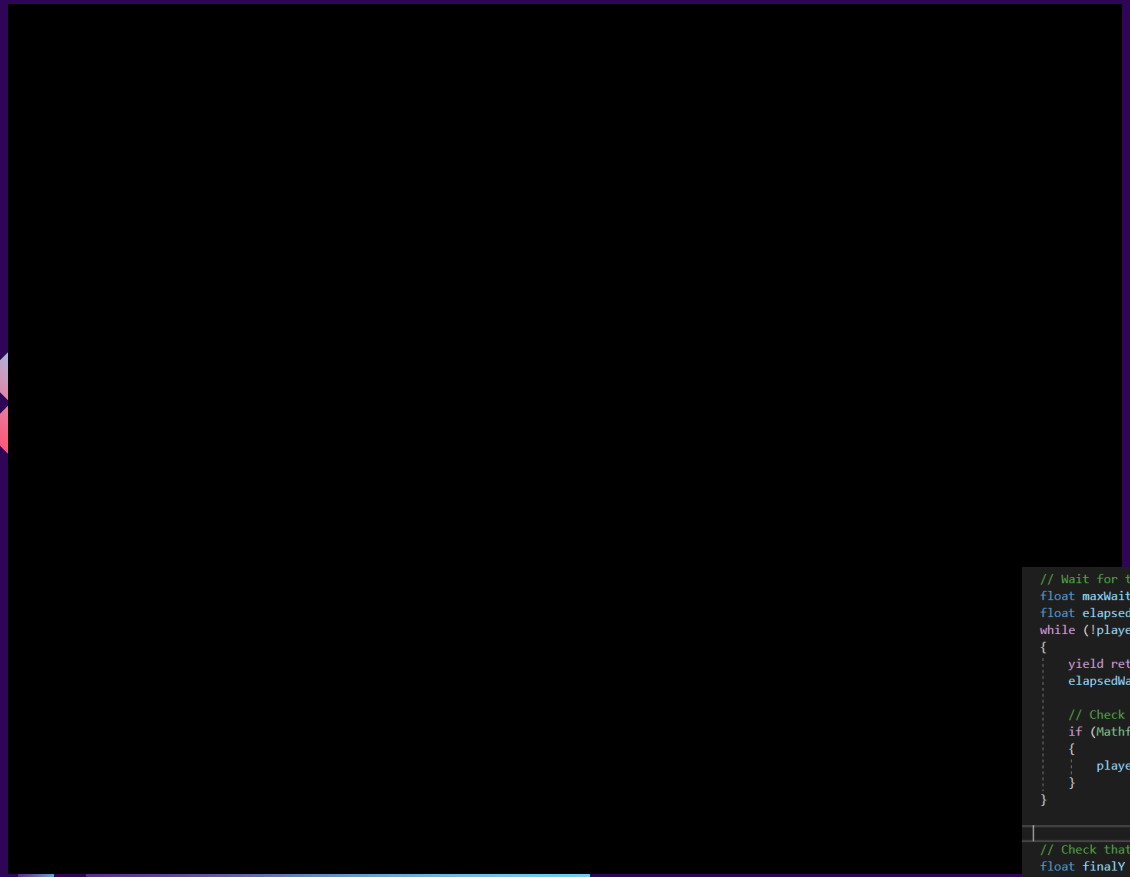


Why Test This?

- Ensure the Fill image does not exceed the border image.
- When multiple enemies and collisions are detected, the bar should still stay within the bounds
- When heal items are implemented, we don't want the health to go above maximum.

```
// Iterate through the health values, setting random values for the health bar
for (int i = 0; i < numHealthValues; i++)
{
    // Generate a random health value between -0.5 and 1.5 (allowing out-of-bounds values)
    float randomHealthValue = Random.Range(-0.5f, 1.5f);

    // Set the health bar's fill amount (clamped between 0 and 1)
    healthBar.fillAmount = Mathf.Clamp(randomHealthValue, 0.0f, 1.0f);

    // Log the raw and clamped fill amounts to check boundary
    Debug.Log($"Raw health value: {randomHealthValue}, Clamped fill amount: {healthBar.fillAmount}");

    // Assert that the fill amount is correctly clamped within bounds
    Assert.That(healthBar.fillAmount, Is.GreaterThanOrEqualTo(0.0f).And.LessThanOrEqualTo(1.0f),
            $"Health bar fill amount {randomHealthValue} (clamped to {healthBar.fillAmount}) is out of bounds.");

    // Wait for 1.5 second to observe the change visually
    yield return new WaitForSeconds(1.5f);
}
```

Test 5 - Jump Test



Why Test This?

- Ensure the game can still handle inputs at high rates.
- Test the grounded check to ensure the player can't get to unintended areas.
- This test, the Player is able to jump on top of the enemy and control where it goes to fly on top of the enemy.

```
// Wait for the player to land after the test
float maxWaitTime = 3f; // Max time to wait for the player to land --> Makes sure it does not end before the player has f
float elapsedWaitTime = 0f;
while (!playerMovement.grounded && elapsedWaitTime < maxWaitTime)
{
    yield return new WaitForFixedUpdate();
    elapsedWaitTime += Time.fixedDeltaTime;

    // Check if the player has stopped falling (vertical velocity close to zero)
    if (Mathf.Abs(rb.velocity.y) < 0.01f)
    {
        playerMovement.grounded = true; // Manually set grounded if velocity is nearly zero
    }
}
```

```
// Check that the player returned to their initial Y position (or close enough within a small margin)
float finalY = playerObject.transform.position.y;
int roundedInitialY = Mathf.RoundToInt(initialY);
int roundedFinalY = Mathf.RoundToInt(finalY);
Assert.AreEqual(roundedInitialY, roundedFinalY, $"Player's final Y position ({roundedFinalY}) does not match initial Y po
Debug.Log("Test passed: Player returned to initial Y position.");
```

What you need to do.

Prior to Exam

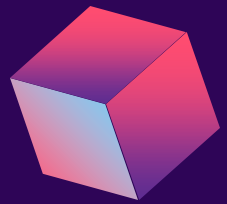
- Have your tests written by the exam
- Already marked on Stress tests. Just make sure the tests are good.
- Make sure the tests are written up in your prework and a brief explanation of what it tests and why?

During the Exam

- Highlight tests you like and explain why you chose to test that thing.
- Explain a test that broke after a teammate changed something in the game.
- **Be able to convince her you knew a test would break, and that's why you chose to test it.**

03 Prefabs

Creating and Documenting Prefabs



Ryan - Ducks

Prefabs for the Oral Exam Prework

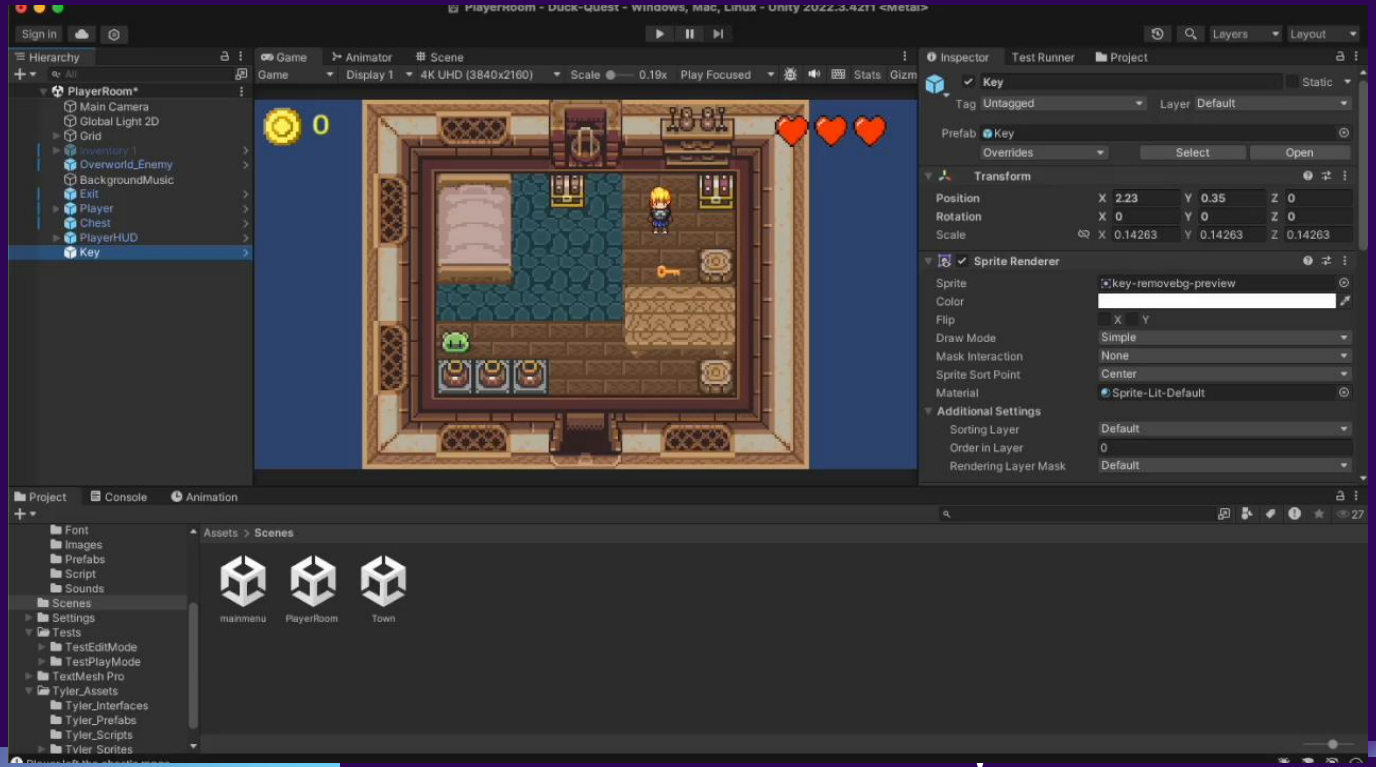
Pick a Prefab you have created that is documented well in a separate readme file. (I will point to several places in your code documentation and ask) What question where you trying to answer here? Who do you anticipate would be asking that question? What other questions might this person need the answers to? Prefab Name: Inventory / Player	/3

Ryan - Ducks

What is a Prefab?

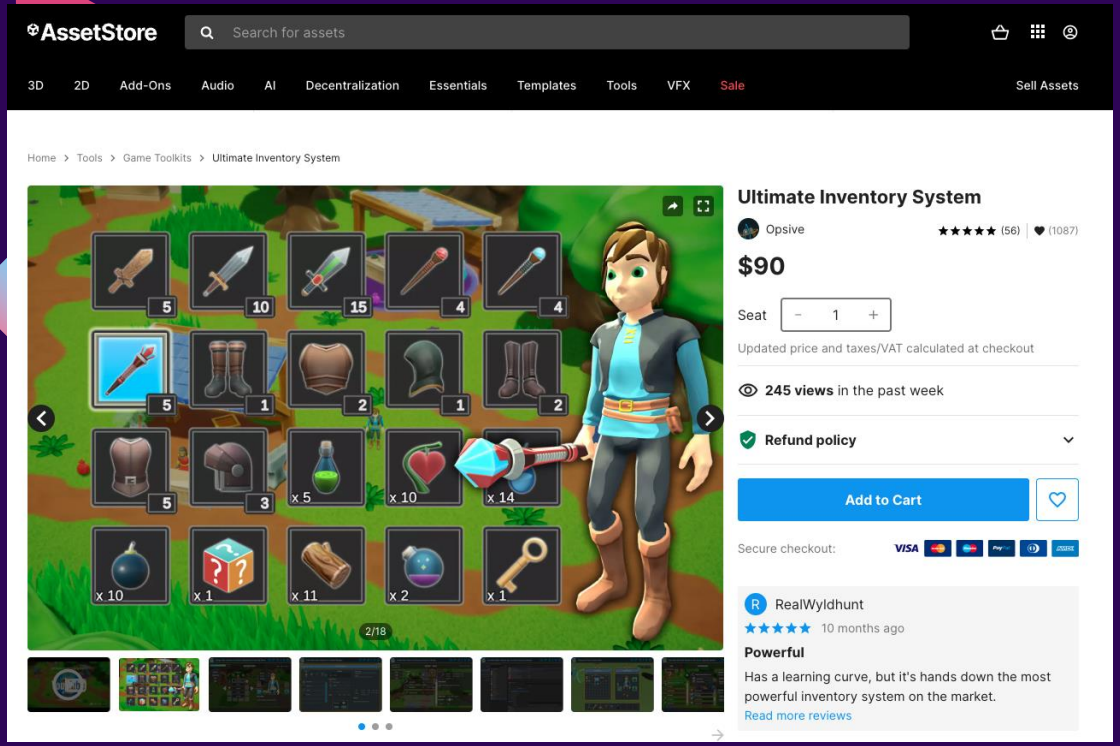
- A prefab is a game object you can edit and save to use across all of your scenes
- The main reasons to use a prefab:
  - When using the same objects throughout multiple scenes
  - Consistency in your game
  - Easier to make changes across whole game
  - Scalability
- Example Prefabs
  - Player Prefab
  - Items (Coins, Power-ups, Weapons)
  - HUD

How to create a Prefab



Ryan - Ducks

Example Prefab



The Ultimate Inventory System is a **robust and fully featured inventory system for Unity**. The system was designed from the ground up to offer flexibility for any genre.

The Ultimate Inventory System contains an intuitive editor which allows you to quickly create your inventory objects. The UI display can be customized to show the inventory using the style of your game. Advanced inventory features such as crafting, upgrades, and a shop system are also included.

Overview | Demo | Documentation | Videos | Forum | Discord

✓Features

- Item Management
- Inventory Display
- Crafting
- Upgrades
- Shop
- Currencies
- Save & Load
- Localization
- Hotbars
- Item Variants
- Item Actions
- Item Shapes
- Import/Export
- Split Screen
- Mobile Friendly
- Controller Support
- Supports 2D and 3D

Ryan - Ducks

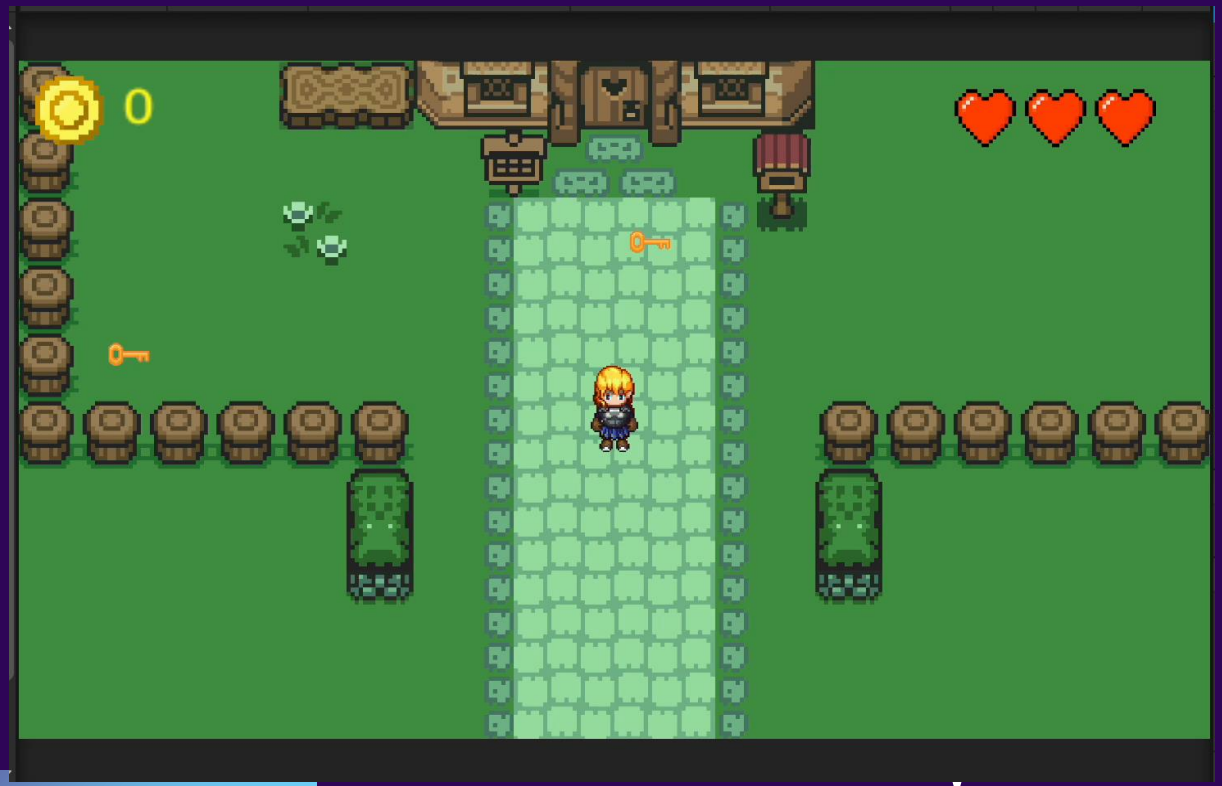
Inventory Prefab Overview

1. Overview

- **Description**
  This prefab is designed to act as a UI for the users to manage their inventory.
- **Key Features**
  - Inventory Management
  - Use, discard, stack items
  - View descriptions for each item
  - Quit Game or Change Volume
- **Includes**
  - Inventory Menu (Canvas)
  - Inventory Slots
  - Inventory Description
  - Inventory Pause

Ryan - Ducks

Inventory Prefab



Ryan - Ducks

Inventory Prefab Set-up

2. Setup Instructions

- **Importing the Prefab**
  - If the prefab is shared as a `.unitypackage` file, import it by going to **Assets > Import Package > Custom Package...**, then select the file.
- **Basic Usage**
  - To use the prefab, simply drag it from the **Project** window into your **Scene**.
- **Dependencies**
  - Ensure the following assets or components are available for proper functioning:
    - Scripts:

      `InventoryItem.cs`

      `InventoryManager.cs`

      `ItemSlot.cs`

    - Assets:

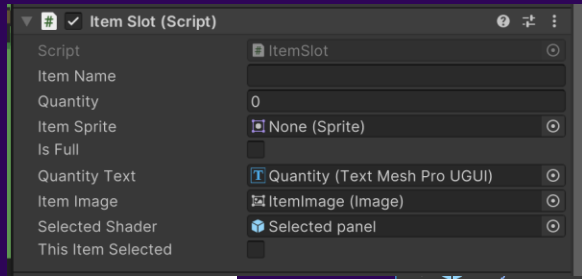
      `Item Description Texture`

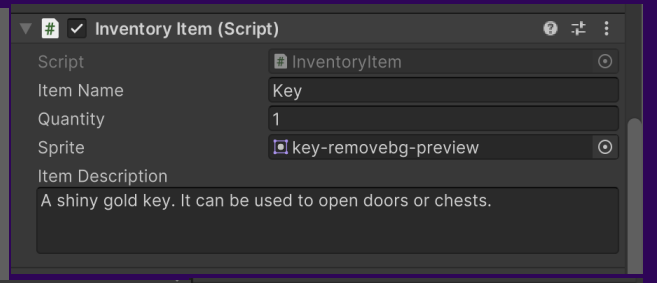
      `Inventory Slot Texture`

      `Inventory BG Texture`

Ryan - Ducks

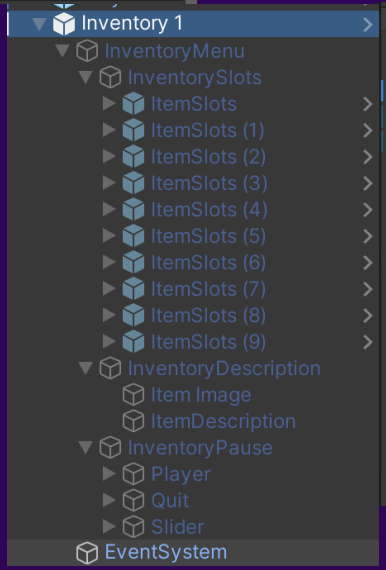
Prefab Components





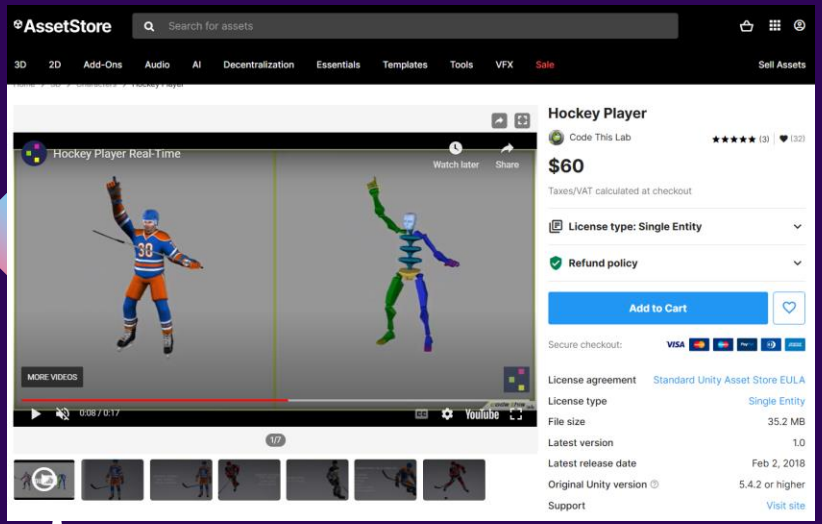
3. Components and Configuration

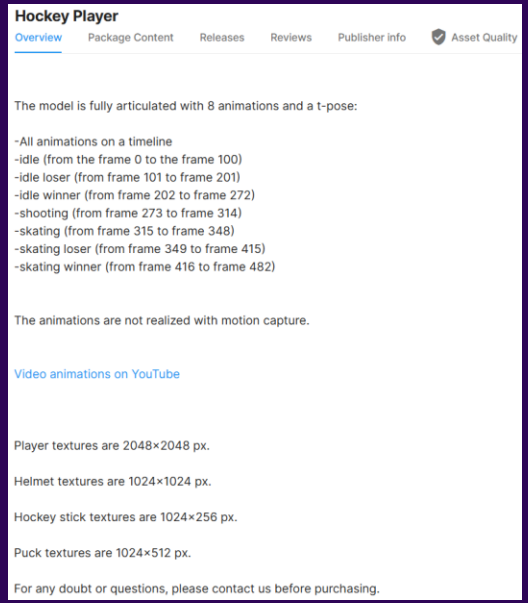
- **Main Components**
  - **Inventory Slots**
  - **Inventory Description**
  - **Inventory Pause**
  - **Item Slot Prefab**
- **Adjustable Parameters**
  - **Slots:** Adjust the total number of slots needed to store inventory items
- **Scripts**
  - **InventoryItem.cs**
    - **Description**: Communicates with InventoryManager item information and destroys items on pick up
  - **InventoryManager.cs**
    - **Description**: Manages inventory functions
  - **ItemSlot.cs**
    - **Description**: Manages information displayed in each item slot, communicates with inventory manager

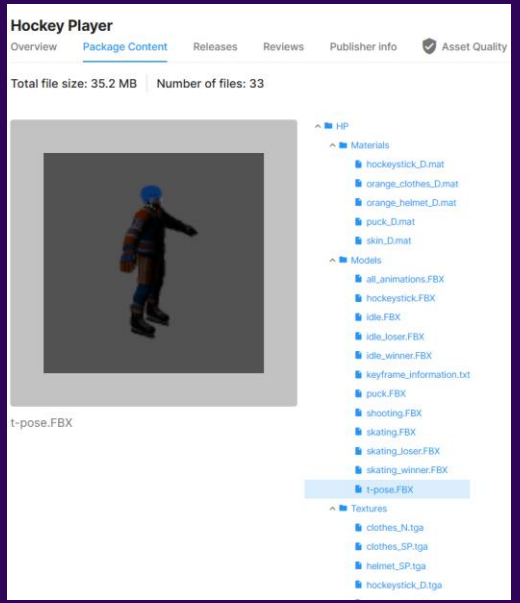


Ryan - Ducks

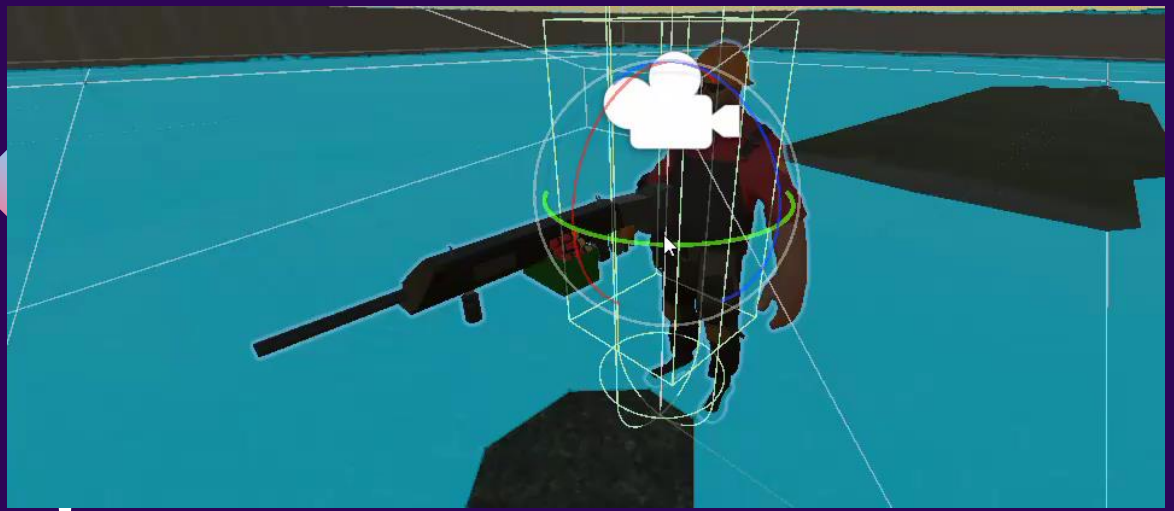
Another Prefab Example







Another Prefab Example



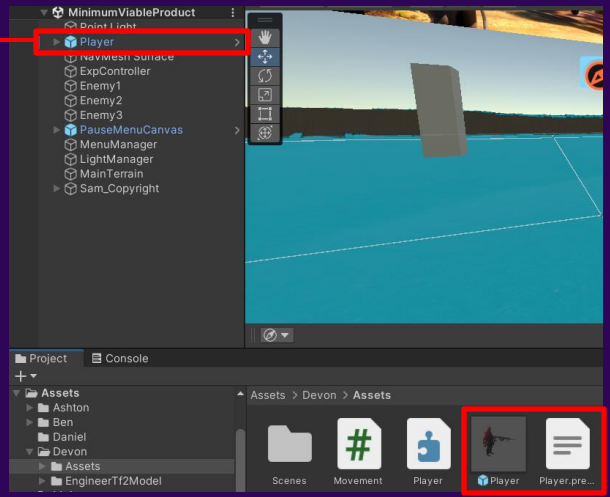
Another Prefab Example

Player.prefab.README.txt

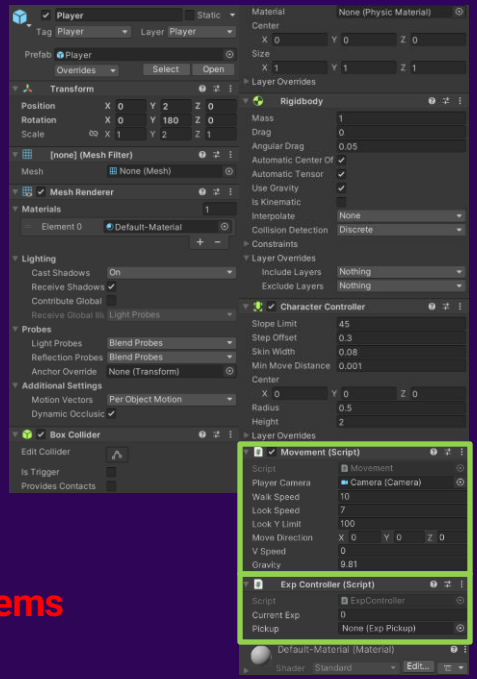
File Edit View

```
Player.prefab README
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Contents:
   - Player
   - Player Model
   - Camera
   - Belt-Fed Shotgun Model
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Player:
   - Contains player information like Box Collider, RigidBody, Character Controller, Movement (Script), and Exp Controller (Script)
         - Movement (Script) handles player gravity, handles camera movement, and uses Character Controller to handle 4-directional player movement
         - Exp Controller (Script) handles how the player collects and stores experience points
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Player Model:
   - Contains all 3D models, textures, materials, and bones for the player model
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Camera:
   - Contains the player camera/field of vision, Camera settings, and an Audio Listener
         - The Audio Listener makes sure that sounds are received as close to the camera as possible
--------------------------------------------------------------------------------------------------------------------------------------------------------------
Belt-Fed Shotgun Model:
   - Contains all 3D models, textures, materials, and bones for the weapon model
   - Contains Gun System (Script)
         - Gun System (Script) handles the amount of ammo in the weapon, firing the weapon, how many times the weapon can fire before reloading, the amount of time it takes to reload,
checking if the weapon can fire, creating muzzle flashes, and damaging enemy entities if in range
--------------------------------------------------------------------------------------------------------------------------------------------------------------
What questions are being answered here? Who would be asking these questions? What other questions would they be asking?
   - What is contained in this prefab? What role does each element serve? How is it organized?
   - Boss/Supervisors/Project Directors
   - Customer/End User
   - Stakeholders?
   - Could this prefab be improved? Does it contain everything it needs? Is it missing anything? Could it be structured more efficiently? How will it interact with other prefabs?
```

Another Prefab Example



Prefab Items



Scripts

04

Dynamic Binding

Everything you need to know

Moe - MILCCN

Dynamic Binding

Show me a class in your code where there could be either static or dynamic binding. Write some mock code on this paper showing how you would set the static type and dynamic type of a variable. Super Class: _____ Sub Class: _____ Virtual Function: _____ Choose a dynamically bound method. What method gets called now? Change the dynamic type. What method gets called now?	/3

Pick a statically bound method. Which one would be called in each of the two previous cases?	

Moe - MILCCN

Dynamic Binding

Show me a class in your code where there could be either static or dynamic binding.

Super Class: UIManager

```
6   public class UIManager : MonoBehaviour
7   {
8       public virtual void ActivateUI()
9       {
10          // Virtual method - This will demonstrate dynamic binding
11          UnityEngine.Debug.Log("UI Activated");
12      }
13      public virtual void DeactivateUI()
14      {
15          UnityEngine.Debug.Log("UI Deactivated");
16      }
17
18      // Non-virtual method - This will demonstrate static binding
19
20      public void ShowUI()
21      {
22          UnityEngine.Debug.Log("UI Displayed");
23      }
24  }
```

Moe - MILCCN

Dynamic Binding

Write some mock code on this paper showing how you would set the static type and dynamic type of a variable.

```
UIManager uiManager; // Static type UIManager
uiManager = new MainMenu(); // Dynamic type MainMenu
```

Dynamic Binding

Super Class: `public class UIManager : MonoBehaviour` Sub Class: `public class MainMenu : UIManager` Virtual Function: `public virtual void ActivateUI()`	/3

Dynamic Binding

Choose a dynamically bound method. What method gets called now?

- Method: ActivateUI()
- When ActivateUI() is called on an instance of MainMenu but referenced by a UIManager type, it will dynamically bind to and call MainMenu.ActivateUI().

```
UIManager uiManager = new MainMenu();
uiManager.ActivateUI(); // Output: "Main Menu Activated"
```

Dynamic Binding

Change the dynamic type. What method gets called now?

```
uiManager = new UIManager();
uiManager.ActivateUI(); // Output: "UI Activated"
```

If we change the dynamic type back to UIManager,
the method from the superclass is called.

Dynamic Binding

Pick a statically bound method. Which one would be called in each of the two previous cases?

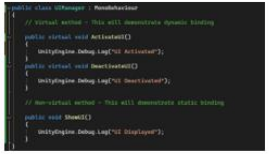
```
public class UIManager : MonoBehaviour
{
    public virtual void ActivateUI()
    {
        // Virtual method - This will demonstrate dynamic binding
        UnityEngine.Debug.Log("UI Activated");
    }
    public void DeactivateUI()
    {
        UnityEngine.Debug.Log("UI Deactivated");
    }

    // Non-virtual method - This will demonstrate static binding

    public void ShowUI()
    {
        UnityEngine.Debug.Log("UI Displayed");
    }
}
```

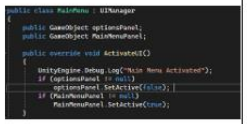
```
// Creating an instance of UIManager
UIManager uiManager = new UIManager();
uiManager.ShowUI();  // Output: "UI Shown" - static binding to UIManager's ShowUI

// Creating an instance of MainMenu, but referenced as UIManager
uiManager = new MainMenu();
uiManager.ShowUI();  // Output: "UI Shown" - static binding to UIManager's ShowUI
```

Moe - MILCCN

How it would look like

Show me a class in your code where there could be either static or dynamic binding.





/4

Write some mock code on this paper showing how you would set the static type and dynamic type of a variable.

```
UIManager uiManager; // Static type UIManager
uiManager = new MainMenu(); // Dynamic type MainMenu
```

/4

Super Class: `public class UIManager : MonoBehaviour`
Sub Class: `public class MainMenu : UIManager`
Virtual Function: `public virtual void ActivateUI()`

Choose a dynamically bound method. What method gets called now?

- Method: `ActivateUI()`
- When `ActivateUI()` is called on an instance of `MainMenu` but referenced by a `UIManager` type, it will dynamically bind to and call `MainMenu.ActivateUI()`.

```
UIManager uiManager = new MainMenu();
uiManager.ActivateUI(); // Output: "Main Menu Activated"
```

Change the dynamic type. What method gets called now?

```
uiManager = new UIManager();
uiManager.ActivateUI(); // Output: "UI Activated"
```

- If we change the dynamic type back to `UIManager`, the method from the superclass is called.

Pick a statically bound method. Which one would be called in each of the two previous cases?

```
// Creating an instance of UIManager
UIManager uiManager = new UIManager();
uiManager.ShowUI();  // Output: "UI Shown" - static binding to UIManager's ShowUI

// Creating an instance of MainMenu, but referenced as UIManager
uiManager = new MainMenu();
uiManager.ShowUI();  // Output: "UI Shown" - static binding to UIManager's ShowUI
```

```
// Non-virtual method - This will demonstrate static binding

public void ShowUI()
{
    UnityEngine.Debug.Log("UI Displayed");
}
```

In both cases, `ShowUI()` from `UIManager` is called because the method is statically bound:

- It doesn't matter whether the dynamic type of `uiManager` is `UIManager` or `MainMenu`, the method call to `ShowUI()` will always execute the `UIManager's ShowUI()` because it is not overridden and not virtual.

Moe - MILCCN

Patrick's example for UI elements:

Super Class - UIElement

```
5 references
public virtual void onClick()
// public void onClick()
{
    Debug.Log("This is the superclass method.");
}
```

Sub Class – UISubmitButton : UIElement

```
5 references
public override void onClick()
// public void onClick()
{
    // Get the player's name from the input field
    string playerName = inputField.GetInputText();

    // Debug.Log($"Player Name Entered: '{playerName}'");  // Log the input value

    Debug.Log($"BCMode: '{MainPlayer.IsBCMode()}");


    if (!string.IsNullOrEmpty(playerName))
    {
        MainPlayer.SetPlayerName(playerName);
        SceneManager.LoadScene("Overworld");
    }
    else
    {
        Debug.Log("Please enter a valid name.");
    }

    //playerName = MainPlayer.getPlayerName();
    //Debug.Log($"Player stored name: '{playerName}'");
}
```

Moe - MILCCN

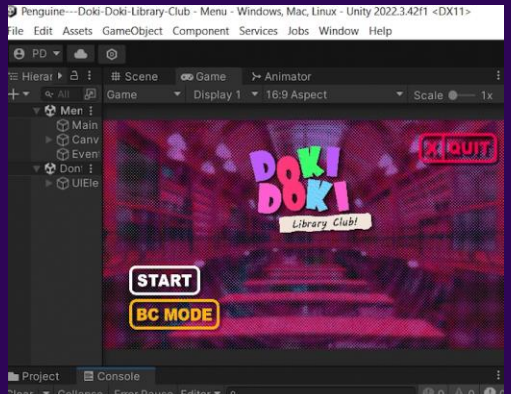
Example continued:

This is how I set up the elements in Unity dynamically

```
0 references
public class Menu : MonoBehaviour
{
    1 reference
    public GameObject submitButton;  // Assign via Inspector
    1 reference
    public GameObject inputField;    // Assign via Inspector

    2 references
    private UIElement submitButtonElement;  //submitButtonElement is of type superclass
    2 references
    private InputName inputFieldElement;

    0 references
    private void Start()
    {
        // Initialize the InputName and UISubmitButton instances
        inputFieldElement = new InputName(inputField);
        submitButtonElement = new UISubmitButton(submitButton, inputFieldElement);  //dynamic binding
    }
```

Video example of Patrick's code:

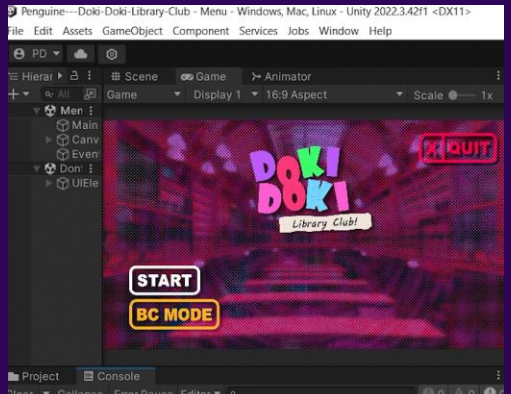
Override method:




```
5 references
public virtual void onClick()
// public void onClick()
{
```

```
5 references
public override void onClick()
// public void onClick()
{
```

Commenting out Override/Virtual:




```
// public virtual void onClick()
1 reference
public void onClick()
{
```

```
// public override void onClick()
3 references
public void onClick()
{
```

Loss of functionality

Moe - MILCCN

05

Patterns

Patterns

4. One big or two small, well-chosen patterns. Small Patterns = {Singleton, Private Class Data} Which patterns did you choose? 1._____ 2._____ Why did you choose each pattern? (Justify your use of it). Draw the class diagram for your pattern(s). Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?	/4

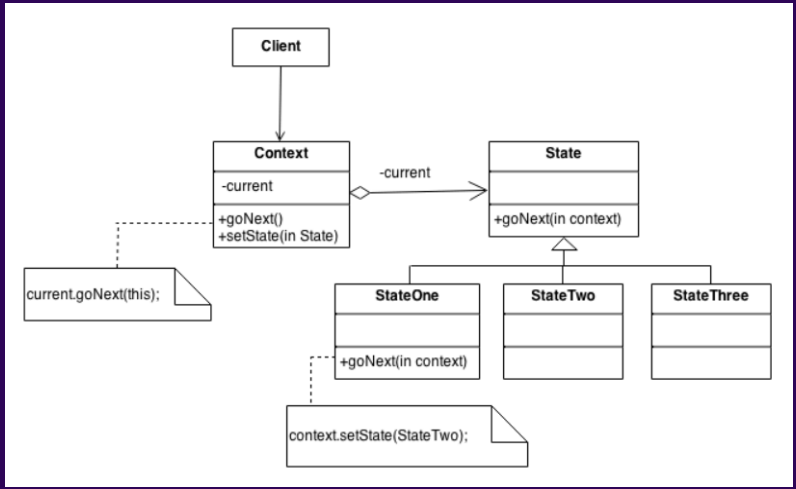
Ben - Nautilus

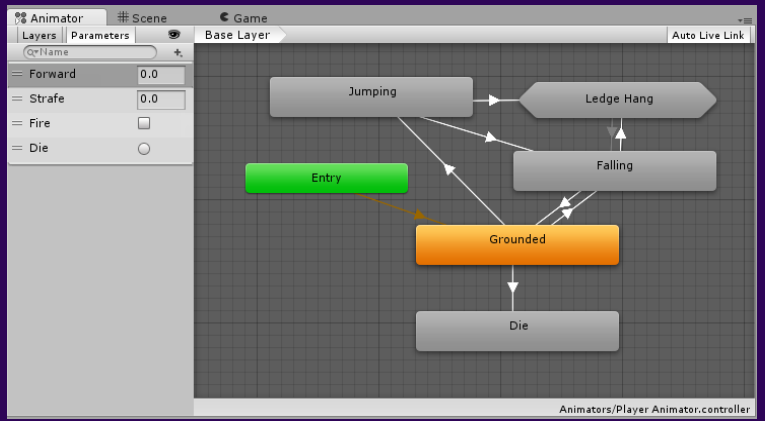
State Pattern

Ben - Nautilus

What is the State Pattern?

- With a "state machine", it utilizes dynamic binding to change behavior of code depending on specific circumstances
- Uses dedicated transitions between states to ensure predictable behavior

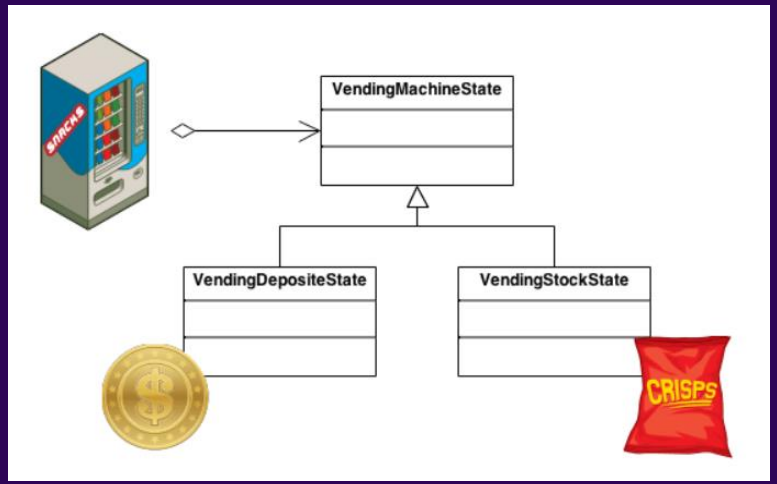




Ben - Nautilus

When to use the State Pattern?

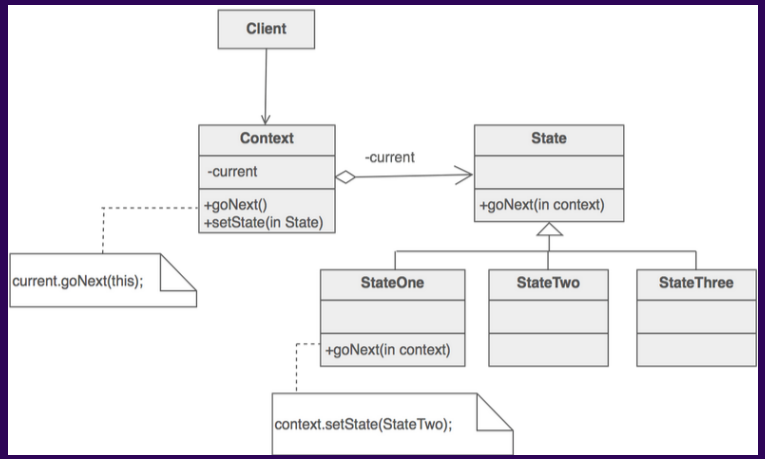
- Vending Machine
  - Depositing
  - Stocking
- UI Manager
  - Paused
  - Running
  - Main Menu
  - Inventory
- Player
  - Idle
  - Walking
  - Swimming
  - Jumping

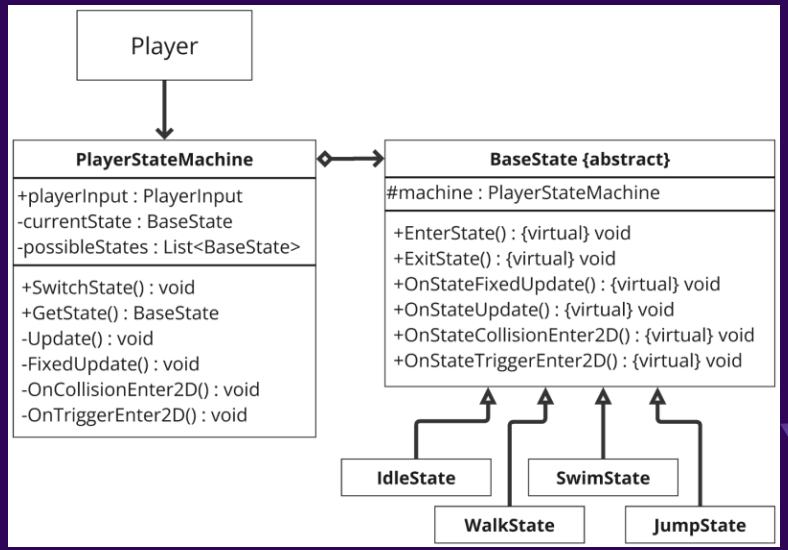


Ben - Nautilus

My Implementation

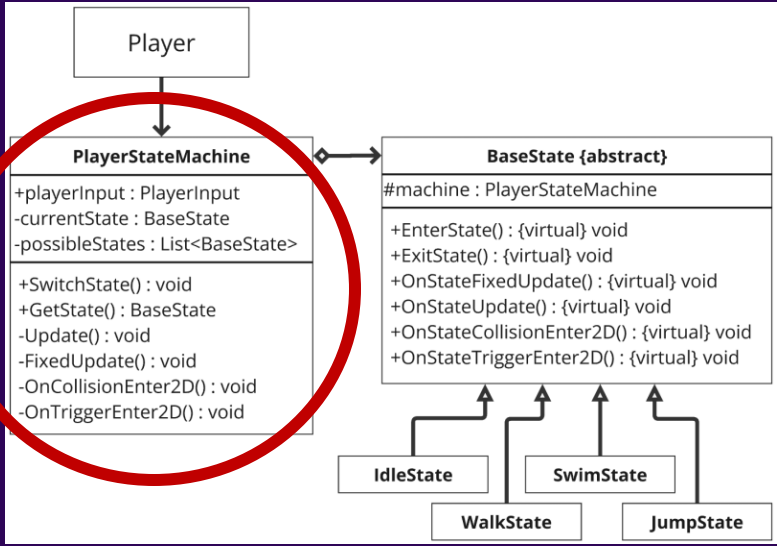
Class Diagrams





Ben - Nautilus

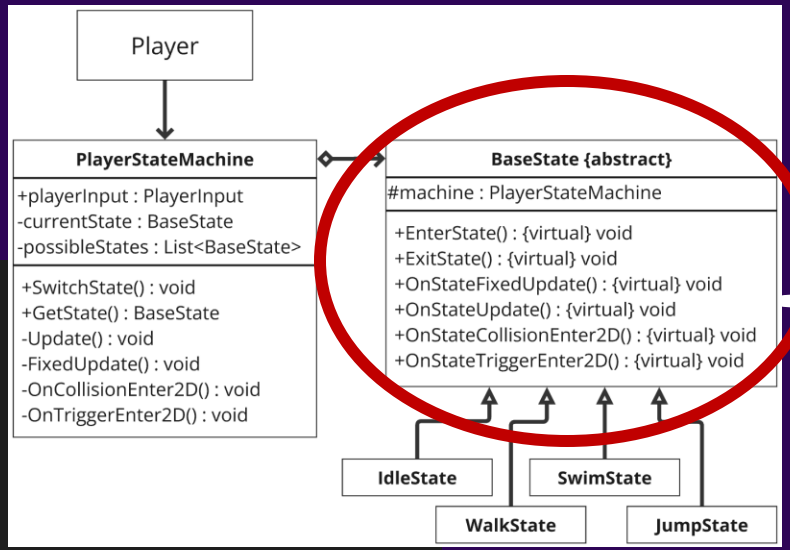
My Implementation




```
public class PlayerStateMachine : MonoBehaviour
{
    public PS_Base CurrentState;
    public PS_Base PreviousState;
    private Dictionary<PlayerStates, PS_Base> _states = new Dictionary<PlayerStates, PS_Base>();

    Unity Message | 0 references
    private void Awake() {
        // Initialize the dictionary with each player state
        _states.Add(PlayerStates.Idle, new PS_Idle(this));
        _states.Add(PlayerStates.Walk, new PS_Walk(this));
        _states.Add(PlayerStates.Swim, new PS_Swim(this));
        _states.Add(PlayerStates.Tread, new PS_Tread(this));
        _states.Add(PlayerStates.Jump, new PS_Jump(this));
        _states.Add(PlayerStates.Fall, new PS_Fall(this));
        _states.Add(PlayerStates.Attack, new PS_Attack(this));
    }

    Unity Message | 0 references
    private void Start() {
        // Sets the default / initial state
        SwitchState(PlayerStates.Idle);
    }

    16 references
    public void SwitchState(PlayerStates newState) {
        CurrentState?.ExitState();
        if(CurrentState != null) PreviousState = CurrentState;

        CurrentState = _states[newState];
        CurrentState.EnterState();
    }

    Unity Message | 0 references
    private void Update() {
        CurrentState.UpdateState();
    }

    Unity Message | 0 references
    private void FixedUpdate() {
        CurrentState.FixedUpdateState();
    }

    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision) {
        CurrentState.OnCollisionEnter2DState(collision);
    }

    Unity Message | 0 references
    private void OnTriggerEnter2D(Collider2D collision) {
        CurrentState.OnTriggerEnter2DState(collision);
    }
}
```

Ben - Nautilus

My Implementation

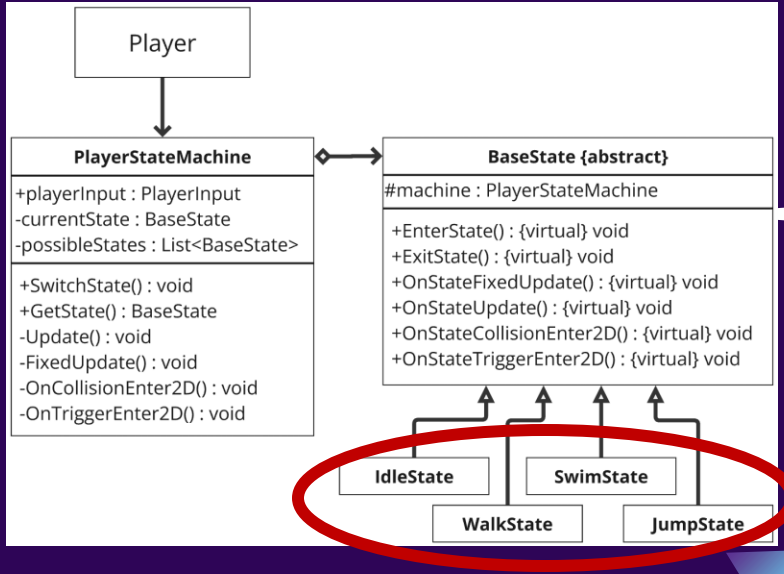



```
public abstract class PS_Base
{
    protected PlayerStateMachine _machine;

    7 references
    public PS_Base(PlayerStateMachine machine) { _machine = machine; }

    15 references
    public virtual void EnterState() { }
    15 references
    public virtual void ExitState() { }

    13 references
    public virtual void UpdateState() {
        if(PlayerInput.Instance.IsJumpPressed && PlayerLogic.Instance.JumpCount < PlayerLogic.Instance.MaxJumpCount) {
            _machine.SwitchState(PlayerStates.Jump);
            return;
        }
    }

    1 reference
    public virtual void FixedUpdateState() { }
    6 references
    public virtual void OnCollisionEnter2DState(Collision2D collision) { }
    1 reference
    public virtual void OnTriggerEnter2DState(Collider2D collision) { }
}
```

Ben - Nautilus

My Implementation

```
public class PS_Walk : PS_Base {
    1 reference
    public PS_Walk(PlayerStateMachine machine) : base(machine) { }

    9 references
    public override void EnterState() {
        base.EnterState();

        PlayerLogic.Instance.Animator.SetTrigger("Walk");
    }

    9 references
    public override void ExitState() { base.ExitState(); }

    7 references
    public override void UpdateState() {
        base.UpdateState();

        PlayerLogic.Instance.MovePlayer();

        if (PlayerLogic.Instance.Rigidbody.velocity.x == 0) {
            _machine.SwitchState(PlayerStates.Idle);
            return;
        }
    }
}
```



Ben - Nautilus

Marking Key Prework

4. One big or two small, well-chosen patterns.
Small Patterns = {Singleton, Private Class Data}
Which patterns did you choose?
1. State

Why did you choose each pattern? (Justify your use of it).

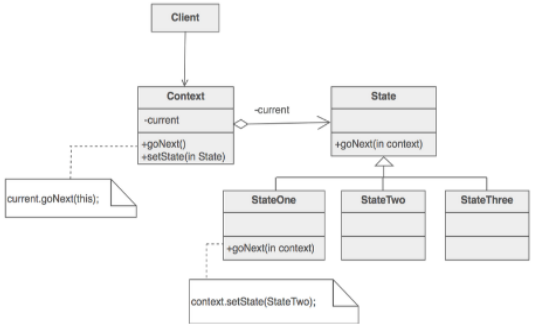
State pattern:
The state pattern is well suited for the use case of a player as a player in a video game will have several different behaviors depending on the received input. Different behaviors, such as walk, jump, and swim, are all going to have unique functionality and trying to achieve it without a state machine will very quickly become messy.

Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?

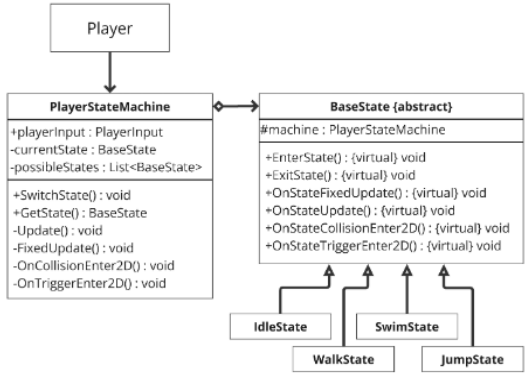
State pattern:
I believe that the state pattern worked very well for the use case of the player, and not much else would have made more sense to accomplish the same thing.

A bad time to use this would be if you have an object with very few states and no chance of scaling beyond, like a light-switch with an on and off state. Another example could be when you have an object with many states, but the logic between each state is very similar and only has minor changes. In such a case, a different pattern may be a better fit.

Draw the class diagram for your pattern(s).
State pattern:
(Pattern example)



(My implementation)



PlayerStateMachine:
(screenshot of code)

PS_Base:
(screenshot of code)

PS_Walk:
(screenshot of code)

Ben - Nautilus

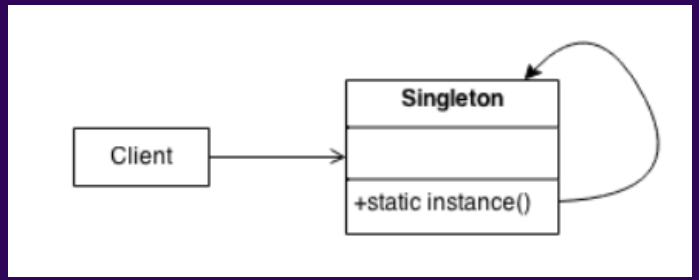
Singleton & Private Class Data Pattern

Patrick - Penguine

The Singleton Pattern

- Pattern ensures a class has only one instance, providing a global point of access to it.
- Important to implement encapsulated "just-in-time initialization" or "initialization on first use."

*Ensure your class diagram follows the structure of the pattern you are implementing!



Patrick - Penguine

My Implementation

```
5 referen
public class UIElementHandler : MonoBehaviour
{
    3 references
    private static readonly Lazy<UIElementHandler> _instance =
        new Lazy<UIElementHandler>(() => FindObjectOfType<UIElementHandler>()); //lazy initialization, thread safety

    0 references
    public static UIElementHandler UIGod => _instance.Value;

    2 references
    public Button quitButton; //Reference to the Quit button
    2 references
    public GameObject endGamePanel; //Reference to the end game panel (UI)


    0 references
    private void Awake()
    {
        //Implement Singleton pattern
        if (_instance.IsValueCreated && _instance.Value != this)
        {
            Destroy(gameObject);
        }
        else
        {
            DontDestroyOnLoad(gameObject);  // Make this instance persistent across scen
        }
    }
```

- Lazy initialization
- Private instance
- Ensures game object persists across scenes
- Global accessor function.

```
//handle end of game scenario (global access point)
0 references
public void EndGame(bool isWin)
{
    endGamePanel.SetActive(true); //Show the end game panel

    // // Update panel text based on win/loss
    // Text panelText = endGamePanel.GetComponentInChildren<Text>();
    // if (panelText != null)
    // {
    //     panelText.text = isWin ? "You Win!" : "You Lose!";
    // }

    Debug.Log(isWin ? "Game Over: You Win!" : "Game Over: You Lose!");
}
```

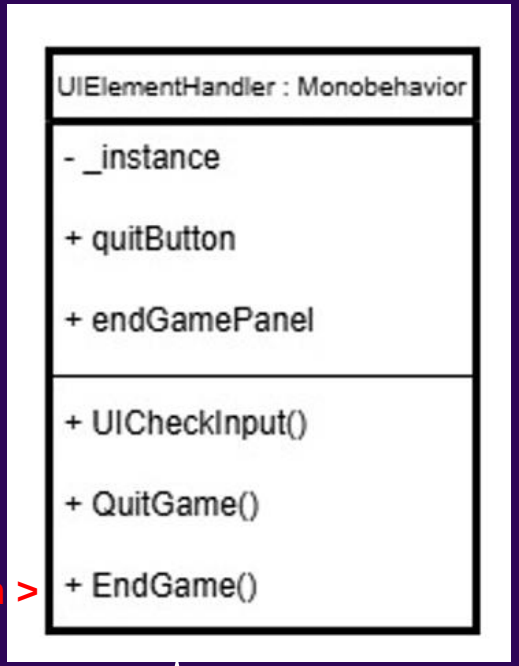
Patrick - Penguine

Diagrams:

Setting up class diagram to match the structure on sourcemaking.com

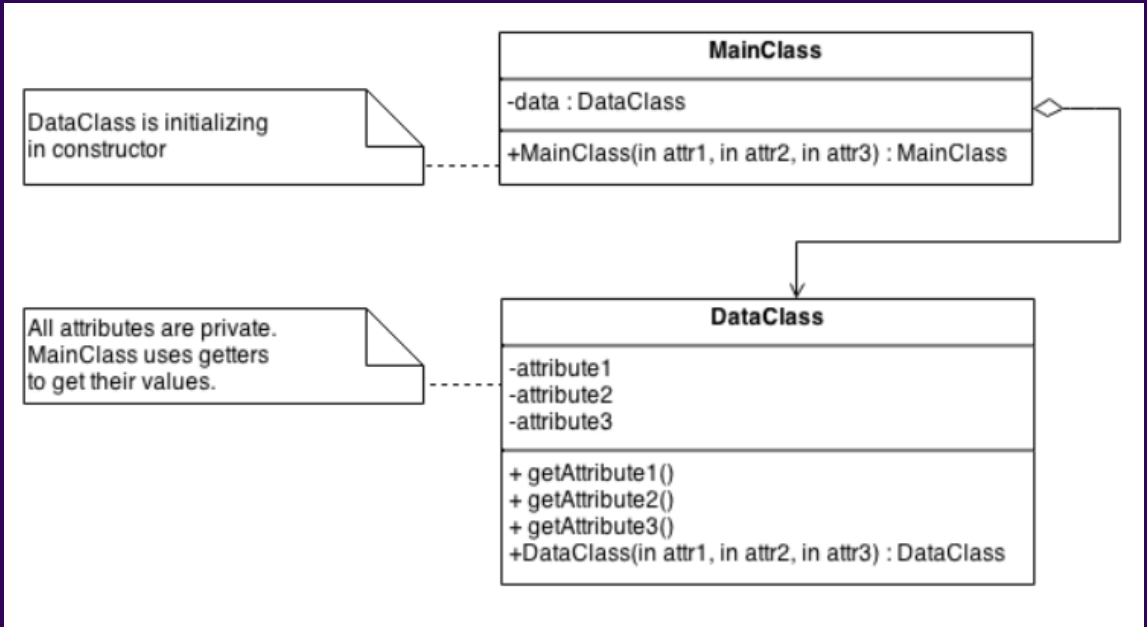
Code should line up with class diagram, and class diagram should line up with the diagram on the website for patterns

Accessor Function >



Patrick - Penguine

The Private Data Class Pattern



- Pattern encapsulates class data initialization
- Separate data from methods that use it
- Control write access to class attributes

*Again, ensure that your class diagram matches the diagram for your pattern on sourcemaking.com

Patrick - Penguine

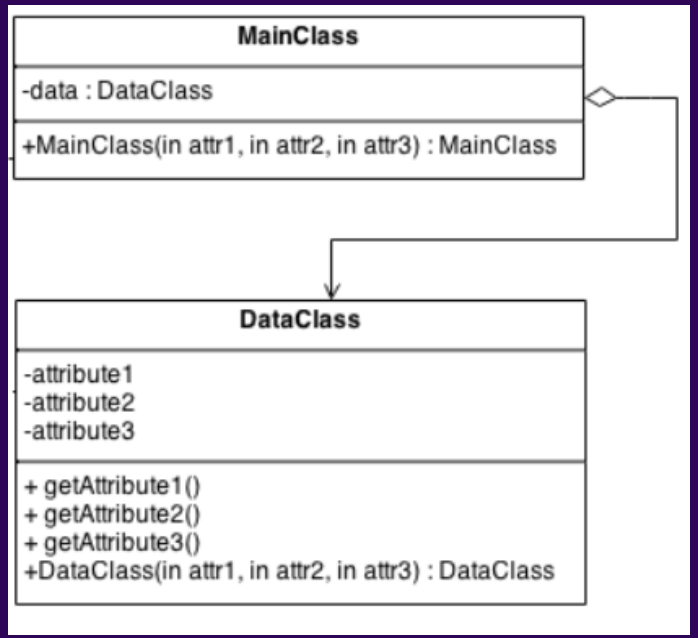
My Implementation

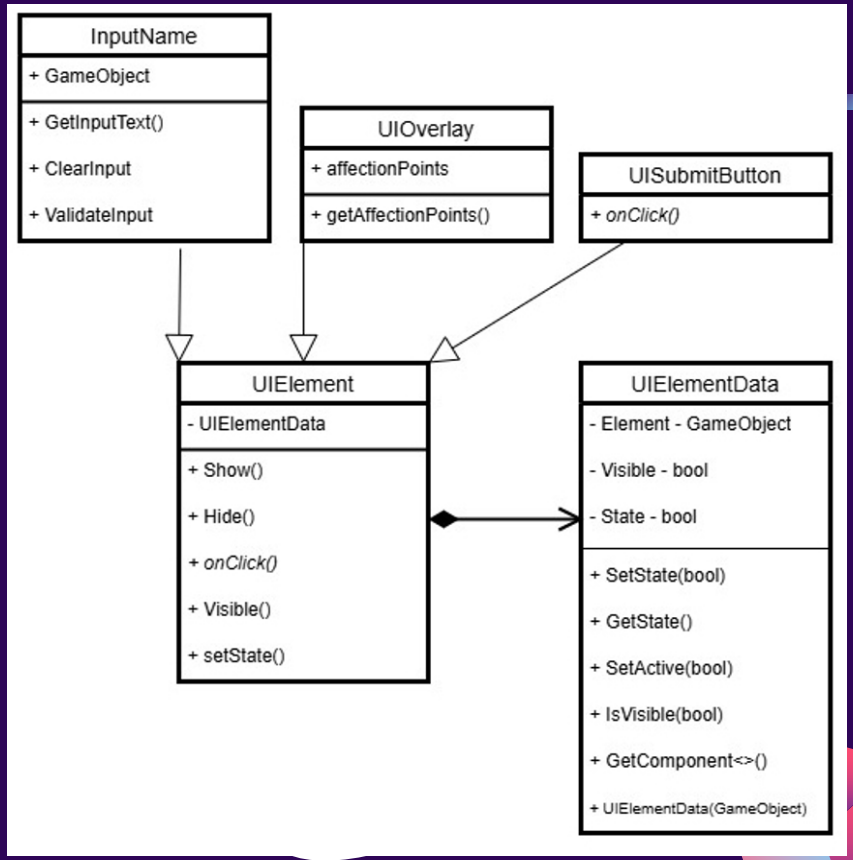
```
8 references
public class UIElement
{
    //private data class pattern:
    3 references
    private class UIElementData{
        4 references
        private GameObject Element;
        3 references
        private bool Visible;   //mainly implemented for the overlay fe
        3 references
        private bool State;     //will use for enabled/disabled button

        1 reference
        public UIElementData(GameObject element){
            Element = element;
            Visible = true;
            State = true;
        }

        2 references
        public void SetActive(bool isActive){
            Element.SetActive(isActive);
            Visible = isActive;
        }

        1 reference
        public void SetState(bool state){
            State = state;
        }

        public T GetComponent<T>() where T : Component
        {
            return Element != null ? Element.GetComponent<T>() : null;
        }  *End of private data class*
    }

    //composition: UIElement contains instance of UIElementData
    6 references
    private UIElementData elementData;

    //constructor initializes data object
    3 references
    public UIElement(GameObject element)
    {
        elementData = new UIElementData(element);
    }
```

Patrick - Penguine

Diagrams:

From sourcemaking.com:





Patrick - Penguine

Filling Out the Prework for Patterns:

4. One big or two small, well-chosen patterns.
Small Patterns = {Singleton, Private Class Data}
Which patterns did you choose?
1._Singleton_____

/4

2._Private Class Data _____

Why did you choose each pattern? (Justify your use of it).

Singleton – I used the singleton pattern for my UIElementHandler because it needs to manage certain button elements, the overlay, get input, and exist across all scenes. This singleton object allows me to manage these features easily because there is only one instance that persists across the game.

Private Class Data – I used the private class data pattern for each of my UIElements in the game. Encapsulating certain values ensures safer implementation and reduces coupling, ensuring that other classes cannot directly manipulate raw data through the use of accessor functions within the private data class. This pattern also improves the flexibility of the code with expansion and evolution in the future.

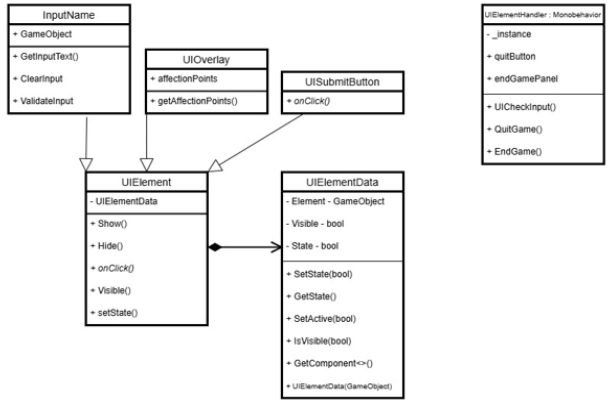
Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?

The singleton pattern is useful here since I needed to implement a management system of some sort. This lets the system persist across all scenes without getting deleted or having data changed.

The private data class may be a little overkill for my current implementation, but it still works well with my class setup since it encapsulates the game `object element itself as well as some other data relevant to each element.

Draw the class diagram for your pattern(s).

Private data class is on the left and the singleton is on the right.



*Paste your code in here too!

Patrick - Penguine

06

Copyright

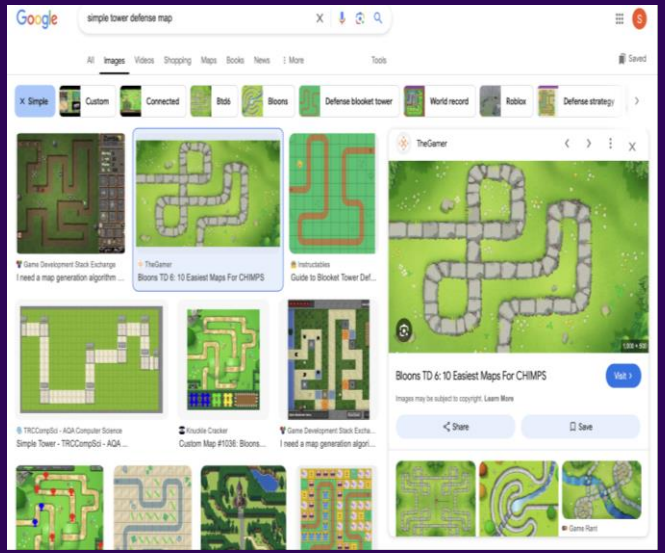
Sohan Lama

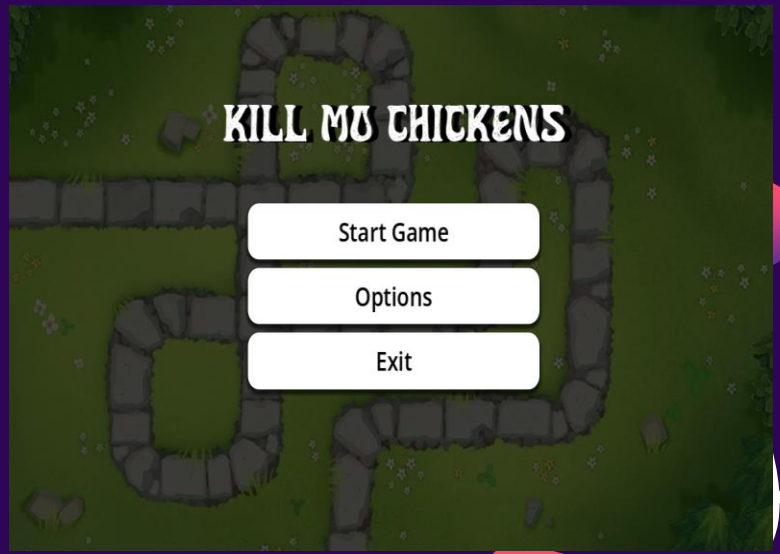
Copyright and Copyright Law

- Copyright is the exclusive legal right, given to an originator or an assignee to print, publish, perform, film, or record literary, artistic, or musical material, and to authorize others to do the same.

- Copyright law is the set of rules that protects creators' original work (like books, music, art, or software) by giving them the exclusive right to control how their work is used. It allows them to copy, sell, distribute, or modify their work and stop others from using it without permission.

Example of Copyright Violation





Sohan – Kill Mo' Chicken

How Does It Violate Copyright?

- **Unauthorized Use**:
  - The image belongs to the original creator (thegamer.com) and downloading it without permission violates their exclusive rights to reproduction and distribution.
  - Even though I darkened the image, it's still a derivative work, which requires permission from the original creator.
- **Derivative Work Violation**:
  - Copyright law protects derivative works (modifications), meaning my darkened version still infringes copyright unless I have the creator’s permission or a license.

What I Did to Integrate It?

- Downloaded the image from Google.
- Modified it by adjusting its brightness to fit the theme of the game.
- Loaded it into my game’s main menu to use as a background.

What Are the Legal Implications if I Market the Game?

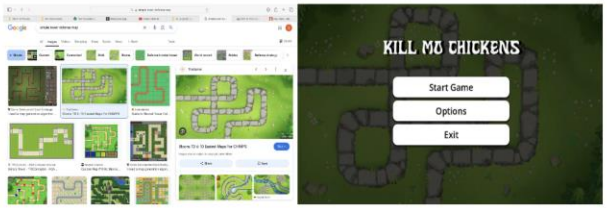
- **Copyright Infringement**:
  - The creator can issue a takedown request (DMCA notice) on platforms where my game is available.
- **Legal Action**:
  - The original creator can sue for damages, including profits from my game.
- **Monetary Compensation**:
  - I could be liable for statutory damages or settlements, depending on the case.

Can I Use Fair Use to Justify My Use?

- **Purpose and Character of Use**:
  - As of now, the game is for educational purposes, which strengthens the fair use argument. Courts tend to favor nonprofit and educational uses under fair use.
  - Argue that the use is transformative: I made the image darker, altering its appearance to fit the mood and style of the game.
    - **How to Defend in the Oral Exam**:
      - The law emphasizes that if a new work is transformative—adding new meaning or purpose—it might still qualify under fair use, even in commercial cases.

Filling Out the Prework for the Copyright.

Show me an example of reuse in your code where you violate copyright law.



How does it violate copyright?_____I used an image without permission, and modifying it still counts as copyright infringement._____

What did you have to do to integrate it with the code you wrote?

I downloaded the image, darkened it to match the game's theme and used it in the main menu background.

What are the legal implications if you market your code with the re-used portion?

- Takedown: The creator can issue a DMCA notice.
- Lawsuit: They could sue for damages.
- Fines: I could owe statutory damages.

Use fair use argue that you can use this anyway.

- Educational Use: Fair use applies since the game is for learning.
- Transformative: I changed the image's appearance to fit the game.

Sohan – Kill Mo' Chicken

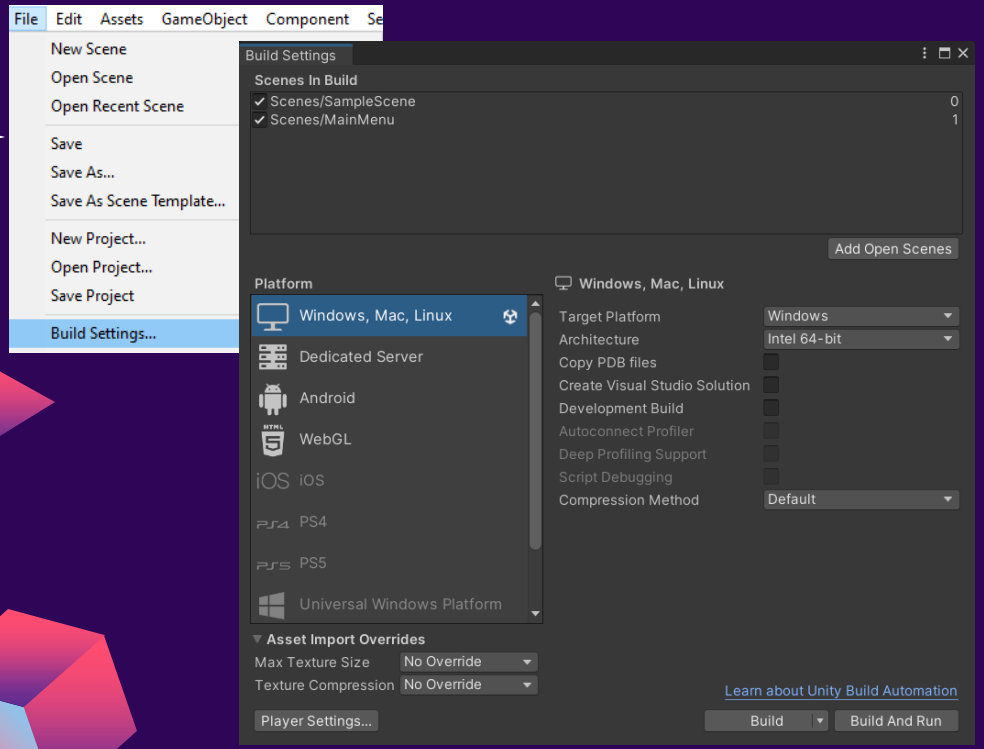
07

Building in Unity

Building and Deploying to other platforms

Ben - Nautilus

Building to Unity

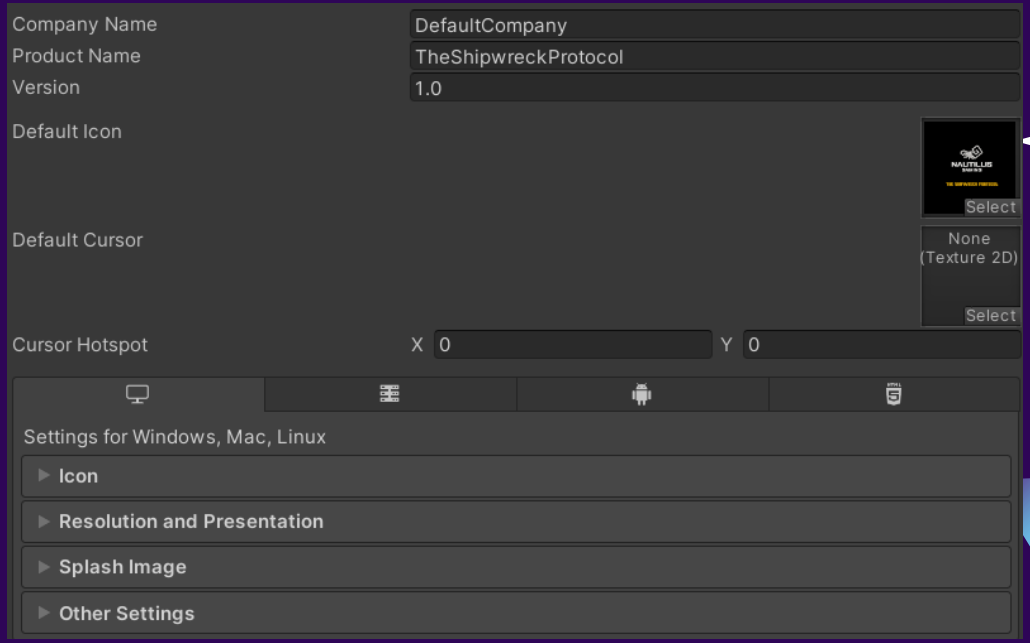


- Pick scenes to build
- Pick a platform to build for (installed separately)
- Single-click building

Ben - Nautilus

Player Settings

- Change build information about your game
  - Resolution
  - Splash screen
  - Default Icons
- Mostly hands-off



Ben - Nautilus

Unity Build Platforms

- Supports many platforms, including but not limited to:
  - Windows, Mac, Linux
  - WebGL (in-browser)
  - Android
  - Apple
  - Game Consoles (PS4/5, Xbox One/Series, Nintendo Switch)

08

Building in Godot

Building and Deploying to other platforms

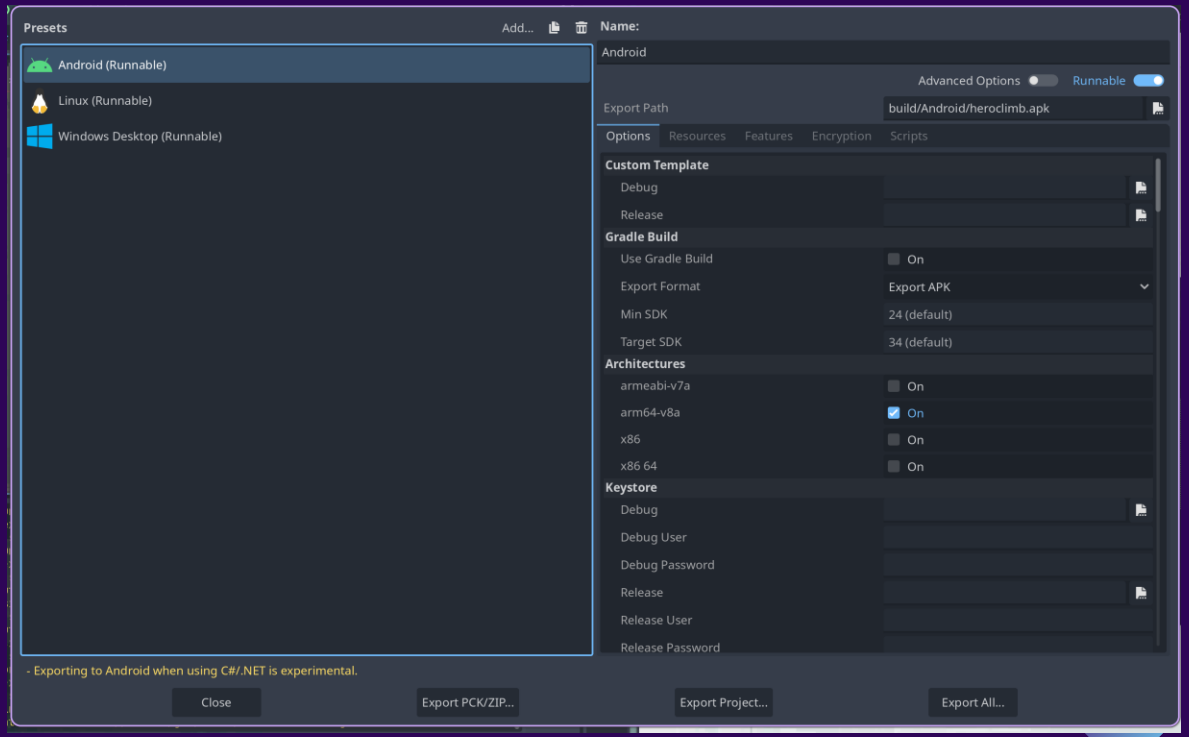
Julia Abdel-Monem - TTK

Building on Godot

- Android: You will need OpenJDK/Java 17 and the Android SDK
- Windows: To sign the executable you will need the Windows SDK or osslsigncode.
- Linux: Should just work
- If your project uses C#, The end user will need .NET runtime 8.0 or newer on their system to run the game.

- To create a compressed downloadable (.zip, etc...) I created a makefile to automate the process

Building On Godot



Julia Abdel-Monem - TTK

Building On Godot

hero-climb > build > makefile

```
pkgname = heroclimb
pkgver = 0.1.0
arch = x86_64

windows_pkg = $(pkgname)-windows-$(pkgver)-$(arch)
linux_pkg = $(pkgname)-linux-$(pkgver)-$(arch)
android_pkg = $(pkgname)-android-$(pkgver)-any.apk

build:
    echo "run export in godot"

clean:
    rm -rf Linux/*
    rm -rf Windows/*
    rm -rf Android/*
    rm -rf Pkg/*

package_windows:
    mkdir Pkg/$(windows_pkg)
    cp ../../README.md Pkg/$(windows_pkg)
    cp ../../Manual.md Pkg/$(windows_pkg)
    cp -r ../../Licenses Pkg/$(windows_pkg)
    cp -r Windows/* Pkg/$(windows_pkg)
    cd Pkg; zip -r $(windows_pkg).zip $(windows_pkg)

package_linux:
    mkdir Pkg/$(linux_pkg)
    cp ../../README.md Pkg/$(linux_pkg)
    cp ../../Manual.md Pkg/$(linux_pkg)
    cp -r ../../Licenses Pkg/$(linux_pkg)
    cp -r Linux/* Pkg/$(linux_pkg)
    cd Pkg; zip -r $(linux_pkg).zip $(linux_pkg)

package_android:
    cp Android/heroclimb.apk Pkg/$(android_pkg)

package: package_windows package_linux package_android

```

heroclimb-android-0.1.0-any.apk

heroclimb-linux-0.1.0-x86_64.zip

heroclimb-windows-0.1.0-x86_64.zip

Julia Abdel-Monem - TTK

Other Build Platforms

—Mobile

—PC

—VR

09

Remaining Items

What to expect for the rest of the semester

Coming up...

11/19

Ethics

Midterm Part 2

11/11-11/15

Oral Exam

JEB 324
Going over your contribution

11/21

Post Mortem

Timed Presentation on GRASP

12/5

Pair Programming

Program in Pairs on other Features

12/5

Final Demo

Showoff your game

Ethics

What to study for:
BC Website --> Part4: Ethics Study Guide
Canvas-->Ethics-->EthicsPDF
Worth 50-70 points (5-7% of your grade)

- The 8 areas of responsibility according to the ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices Ethical principles.
- The 4 areas of risk in a project.
- What is covered under copyright
- 4 areas of consideration in fair use
- Static/Dynamic Binding + AI

Oral Exam

JEB 324: Arrive 15 minutes early
BC Website--> Oral Exam -->Marking Key

- Fill out the Marking Key completely and submit it to the drop box
- Bring a computer with access to the code and your repository
- Walk through each section

You cannot bring any study aids except the code.
Use Comments to document your code

Post Mortem

Timed Presentation:
BC Website --> Part6: Post Mortem Presentation and Marking Key

- Exactly 12 Minutes. On GRASP, Cohesion and Coupling and what your team might've done differently after.
- Group mark, but you can choose as a group or one member to do the presentation.

Important: If the presentation goes over or under 12 Minutes, you will be marked down.

Pair Programming

- Find a pair partner – Picked based on activity, or if there is someone you really want to pair with.
- Pick an activity- During the Oral exam, there will be a list of pair programming activities. Your name will go next to an activity.
- Make a meaningful contribution (2-3 hours max)
- Choose someone from your pair to write up a short report. 300 words.

Important: Before you submit your additions, get the scrum master(TL6)'s permission.

Final Demo

Date 12/5: Showcase your final game

- Bring laptops/mobile devices that can play the game. The more devices the better.
- Focus on making sure the "public" enjoys the game.

10

TL6 Checklist

Patrick - Penguine

Check List

- TL6s will be going through an oral exam prework check list prior to NEXT THURSDAY
  - TL6 status report, the week before oral exams.
- Make sure you have everything filled out correctly for your own benefit by then.

- If you have questions about the prework, ask your Team Lead 6!

Patrick - Penguine

Questions?