|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Davin Lewis | Team | Nautilus | TL | 3 | Date | 11/11/24 | Time | 13:30 |

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:  My features are all of UI. I made the main menu, the pause menu, the settings menu, the health UI, and the ammo UI. Each of these uses Unity’s built in UI features along with my own code. Some of the features use art made by Alice.  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?    This took 34 hours of work; my estimation was around 20. I didn’t expect to be making as many features as I did so my estimations were pretty far off. Some people in my group didn’t really work on the game so I ended up scrapping a couple of my features, otherwise I’d be even farther off.  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.  [Gameplay Video](OralExam.mp4) | /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.)  I unintentionally broke one of my own tests by adding my own feature. When I added BCMode, my drowning test can pass or fail. While testing BCMode’s state is saved to playerprefs so if you turn on BCMode anytime before running the test it no longer works, but if you have it off then it does work. This can be fixed by ensuring that the BCMode is set to off in the code. I wanted this test to make sure that the drowning feature was working so players couldn’t go deep in the water without the Oxygen Tank Item (which ended up never getting added).    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: Settings Menu  <README.html>  [SettingsMenuPrefabDetails.docx](SettingsMenuREADME.docx)  These should hyperlink to the other docs in this folder.  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.  **Slider.cs**  using System.Collections;  using System.Collections.Generic;  using UnityEngine;  // Base class for slider implementations  public class SliderBase  {  // Protected field to store the slider's current value  protected float value;  // Property for accessing and modifying the slider value  public float Value  {  get => value; // Getter for the value  set  {  this.value = value; // Set the value  OnValueChanged(value); // Notify derived classes of the change  }  }  // Virtual method to handle actions when the value changes  protected virtual void OnValueChanged(float value)  {  Debug.Log("Value changed to: " + value);  }  // Virtual methods for loading and saving slider settings  public virtual void LoadSetting()  {  value = 0f;  Debug.Log("LoadSetting called in base class");  }  public virtual void SaveSetting()  {  Debug.Log("SaveSetting called in base class");  }  }  **MusicVolumeSlider.cs**  using UnityEngine;  using TMPro;  public class MusicVolumeSlider : SliderBase  {  private void Start()  {  // Initialize the music source volume and update the volume display on start  AudioManager.Instance.musicSource.volume = value;  UpdateVolumeText(value);  }  // Text component to display the volume value  private TextMeshProUGUI volumeText;  // Set the volume text display  public MusicVolumeSlider(TextMeshProUGUI volumeText)  {  this.volumeText = volumeText;  }  // Handle volume changes when the slider value is updated  protected override void OnValueChanged(float value)  {  // Update the audio source volume based on the slider value  AudioManager.Instance.musicSource.volume = value/100;  // Change the volume text display  UpdateVolumeText(value);  }  // Update the displayed volume percentage  private void UpdateVolumeText(float value)  {  volumeText.text = "Volume: " + $"{Mathf.RoundToInt(value)}" + "%"; // Display as a percentage (0–100)  }  // Save the current volume setting to PlayerPrefs  public override void SaveSetting()  {  PlayerPrefs.SetFloat("MusicVolume", value);  }  // Load the volume setting from PlayerPrefs  public override void LoadSetting()  {  value = PlayerPrefs.GetFloat("MusicVolume", 1.0f); // Default to 1.0 if no setting exists  OnValueChanged(value);  UpdateVolumeText(value); // Initialize text display with loaded value  }  }  **SFXVolumeSlider.cs**  using UnityEngine;  using TMPro;  public class SFXVolumeSlider : SliderBase  {  // Text component to display the volume value  private TextMeshProUGUI volumeText;  // Set the volume text display  public SFXVolumeSlider(TextMeshProUGUI volumeText)  {  this.volumeText = volumeText;  }  // Handle volume changes when the slider value is updated  protected override void OnValueChanged(float value)  {  // Update the audio source volume for sound effects  AudioManager.Instance.fxSource.volume = value/100;  // Refresh the volume text display  UpdateVolumeText(value);  }  // Update the displayed volume percentage  private void UpdateVolumeText(float value)  {  volumeText.text = "Volume: " + $"{Mathf.RoundToInt(value)}" + "%"; // Display as a percentage (0–100)  }  // Save the current SFX volume setting to PlayerPrefs  public override void SaveSetting()  {  PlayerPrefs.SetFloat("SFXVolume", value);  }  // Load the SFX volume setting from PlayerPrefs  public override void LoadSetting()  {  value = PlayerPrefs.GetFloat("SFXVolume", 1.0f); // Default to 1.0 if no setting exists  UpdateVolumeText(value); // Initialize text display with loaded value  }  }  **SettingsManager.cs**  using UnityEngine;  using TMPro;  using UnityEngine.UI;  public class SettingsManager : MonoBehaviour  {  private SliderBase musicVolumeSlider; // Slider superclass for music volume  private SliderBase sfxVolumeSlider; // Slider superclass for sound effects volume  private Slider musicSlider; //Actual unity slider object  private Slider sfxSlider; //Actual unity slider object  private void Awake()  {  // Locate the text components in the scene for volume displays  TextMeshProUGUI musicVolumeText = GameObject.Find("MusicVolumeText").gameObject.GetComponent<TextMeshProUGUI>();  TextMeshProUGUI sfxVolumeText = GameObject.Find("SFXVolumeText").gameObject.GetComponent<TextMeshProUGUI>();  //Using text components get the slider components  musicSlider = musicVolumeText.transform.parent.GetComponent<Slider>();  sfxSlider = sfxVolumeText.transform.parent.GetComponent<Slider>();  //DYNAMIC BINDING  // Initialize the volume sliders with the found text components  musicVolumeSlider = new MusicVolumeSlider(musicVolumeText);  sfxVolumeSlider = new SFXVolumeSlider(sfxVolumeText);  // Load saved settings for both volume sliders  musicVolumeSlider.LoadSetting();  sfxVolumeSlider.LoadSetting();  }  //Sets the actual Unity slider values  private void OnEnable()  {  musicSlider.value = musicVolumeSlider.Value;  sfxSlider.value = sfxVolumeSlider.Value;  }  // Method to set the music volume and save the setting  public void SetMusicVolume(float value)  {  //Sets value in in the slider subclass  musicVolumeSlider.Value = value;  //Saves new value  musicVolumeSlider.SaveSetting();  }  // Method to set the sound effects volume and save the setting  public void SetSFXVolume(float value)  {  sfxVolumeSlider.Value = value;  sfxVolumeSlider.SaveSetting();  }  // Getter for the current music volume  public float GetMusicVolume() => musicVolumeSlider.Value;  // Getter for the current sound effects volume  public float GetSFXVolume() => sfxVolumeSlider.Value;  }  **Mock Code**  Sliderbase SuperSlider;  TextMeshProUGUI MusicVolumeText;  Sliderbase SubSlider = new MusicSlider(MusicVolumeText);  Super Class: SliderBase  Sub Class: MusicSlider, SFXSlider  Virtual Function: OnValueChange, LoadSetting, SaveSetting  Choose a dynamically bound method. What method gets called now?  In SettingsManager.cs the sliders will call SFXVolumeSlider and MusicVolumeSlider to adjust volume rather than the SliderBase function.  Change the dynamic type. What method gets called now?  I’m not sure what this questions means. If you mean changing to a different dynamic type, then the functions would adjust the wrong volume in my code.  Pick a statically bound method. Which one would be called in each of the two previous cases?  The method being called with a statically bound method would be the SliderBase functions.  Show me an example of reuse in your code where you violate copyright law.    How does it violate copyright? Copies the style and design of another company’s game logo.  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.  I had to combine 2 fonts into 1, create the title, and add the image in a spot that doesn’t seem out of place. They would be able to take legal action to prevent me using the logo, possibly try to recoup “damages” through financial compensation, may request destruction or ownership over the logo.  4. One big or two small, well-chosen patterns.  Small Patterns = {Singleton, Private Class Data}  Which patterns did you choose?   1. Observer   2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Why did you choose each pattern? (Justify your use of it).  The observer pattern allows me to keep my code separate while allowing my UI to be told when things are happening. This was perfect for decoupling my code while still being able to interact between the information source and UI. The script detecting whether a player collides with an damaging object and if the player is drowning is able to send both signals to observers on whether the player is losing health or drowning.  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?  I don’t think anything would’ve worked quite as well as the observer pattern. I might have been able to use a state machine, but I think it would’ve made the code much more complex than the current version. The observer pattern would be bad to use when there needs to be a specific order to events. An example of this would be having 2 observers to one subject but one observer has to run before the other. With how the observer pattern works, both observers would be notified to start working at the same time. | /4  /3  /3  /4  /4 |