|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Hengyi Tian | Team | Green Sky Games | TL | 5 | Date | Apr. 13 | 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:  In the game named “Breaking Red”, my feature is responsible for designing the Scene Engine, including environment and audio systems. For environment system it contains a weather system and a background system, including rain, snow, fog, dust, green sky, tornado, and 7 different transition scenes. For audio system it contains background music and sound effects, including more than 11 dialogue sounds, 8 environmental sounds, 9 background sounds, and more than 8 sound effects. And I also created demo modes for all 5 levels.  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?  Until now, I have spent about 50 hours finishing the scene engine system. I initially estimated it would take about 56 hours, as my Gantt chart shows. First, I started from requirements collections. In the Unity Asset Store, I searched for horror background music and sound effects that matched the game’s story. Then, establishing the environment and weather system, such as rain, snow, fog, and dust, due to the level maps were built first. As the levels were generated, I began creating the audio system, including background music and sound effects. This part of the work was scattered and fragmented. Because I needed to understand the codes logic of all my teammates and add the audio code to their files. Such as all the buttons in all menus, the player’s actions, NPCs dialogues, collectible actions, sliding door and passaway opening, and triggers for victory or defeat. I took a huge amount of time to learn codes’ logic and communication. At the same time, I was constantly modifying and adjusting my code based on my teammates’ code and continuously testing it.  I learned that changes always go beyond plans, especially when working in a team. A good estimator needs to consider all the potential problems and unexpected situations, such as GitHub merging problems, codes overwritten by my teammates, debugging followed by others’ codes changing and so on. This definitely requires experience and good communication. Fortunately, after the journey of our course, I have gotten a lot of valuable experience as a team 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.)  Due to our TL3 did not get game test runner assembly operation. I am still trying to figure this part out. Thank you.  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: Audio Manager  The ReadMe file will give a basic introduction about the audio manager, including description, features, and requirements. Giving this file, I am trying to answer the question: How to manage audio resources in an efficient and reusable way when designing a Unity game?  People who work on Unity design, such as developers or maintainers would be asking this question. They need to know how to connect their own components to the audio systems. In addition, they might ask more detailed questions. Such as “How do I trigger a sound from my script by using this AudioManager prefab?” or “Can I loop the audio by using this AudioManager prefab?”  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: Weather Source Code**  **Sub Class: RandomWeather Source Code**  **Virtual Function: v\_getRandomWeather()**  // Provides a virtual method that can be overridden to return a random weather condition.  public class Weather  {  // This virtual function return a random weather condition. It can be overridden by subclasses to provide specific weather  public virtual int v\_getRandomWeather()  {  return 0; // No specific weather that can be overridden  }  }  // Subclass that overrides the virtual weather function to return a randomly generated weather value.  public class RandomWeather : Weather  {  // This override of the v\_getRandomWeather function generates a rondom weather type by returing either 0 or 1  public override int v\_getRandomWeather()  {  return Random.Range(0, 2); // 0 = rain, 1 = snow  }  }  Choose a dynamically bound method. What method gets called now?  Change the dynamic type. What method gets called now?  Pick a statically bound method. Which one would be called in each of the two previous cases?  The dynamically bound method is v\_getRandomWeather. If I use virtual and override keywords, dynamic binding happens. Because the actual type of \_myWeather = new RandomWeather(), the v\_getRandomWeather method in the subclass RandomWeather, will be called. When I remove the virtual and override keywords in superclass Weather and subclass RandomWeather, all methods become statically bound. Now the v\_getRandomWeather() method from the superclass Weather will be called. It means that the weather will no longer randomly change and it will always return the same value, which could be always raining.  Dynamic Binding\_WeatherManager\_Before and After.mp4 shows an example of what removing the keywords does  Show me an example of reuse in your code where you violate copyright law.  How does it violate copyright? I used a 30-second clip of the original soundtrack from the The Legend of Zelda as my game’s main menu background music. This is a well-known piece of copyrighted music owned by Nintendo.Therefore, this music falls under copyright law. But I did not have permission or license from Nintendo.  What did you have to do to integrate it with the code you wrote?  I edited the original soundtrack music and imported the first 30 seconds of the copyrighted music into my Unity game and added it to the AudioManager. When the main menu loads, the AudioManager plays the music as background music. I found it through a Google search.  What are the legal implications if you market your code with the re-used portion?  If I market or publish the game in the future, I would likely be infringing on Nintendo’s copyright. Cause I do not own the rights to the Zelda soundtrack and have not obtained a license. This could result in takedown notices, demonetization, or legal action, especially the music is used prominently and recognizably in the game.  Use fair use argue that you can use this anyway.  1. Amount and substantiality of the portion taken  I only used the first 30 seconds of the Zelda music, and it doesn’t include the most recognizable part of the theme. It’s more like background buildup, not the part that people usually remember. Because of that, I’d argue this factor leans more in favor of fair use. I’m not using the heart of the music.  2. Effect of the use upon the potential market  My game is made for mobile phones, while The Legend of Zelda is mainly for Nintendo consoles like the Switch. Since we’re on totally different platforms and scales, my game isn’t really competing with theirs. It’s unlikely that using this short clip would hurt Nintendo’s music sales or take away from their audience.  4. One big or two small, well-chosen patterns.  Small Patterns = {Singleton, Private Class Data}  Which patterns did you choose?  1. Singleton Pattern: AudioManager Class  2. Observer Pattern: BackgroundMusic Class  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?  1. Singleton Pattern: AudioManager Class  I chose the Singleton pattern for the AudioManager. Because in a Unity game, audio must remain consistent and centralized across all scenes. This pattern ensures that only one instance of the AudioManager exists and provide a global point of access to it. Without the Singleton pattern, each scene could create its own AudioManager, which might result in multiple overlapping audio musics.    Figure 1 Structure from sourcemaking.com Figure 2 Class Diagram for Singleton Pattern  Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?  Singleton is the most effective pattern for global audio system in Unity. The event-driven Observer pattern could also work in a relatively complex Audio system.  For bad time, if the project grows complex and has huge architecture, using Singleton Pattern could be hard to mock or replace in Unity testing. It can hide dependencies and make the project harder to scale.  2. Observer Pattern: BackgroundMusic Class  In the BackgroundMusic class, I used the Observer Pattern by subscribing the Unity’s built-in sceneLoaded event. This allows the class to detect scene transitions and play new scene’s music automatically. To be specifical, when a new scene is loaded, BackgroundMusic fades out the current playing music and fades in the appropriate background music for the new level scene.    Figure 3 Structure from sourcemaking.com    Figure 4 Class Diagram for Observer Pattern  Would something else have worked as well or better than this pattern? When would be a bad time to use this pattern?  In this situation, yes but it would not be better. I could manually call the playing code between each scene and the AudioManager instead of using Observer pattern. But it would not be as flexible as the Observer method.  For bad time, If my game only has 1 or 2 scenes and the background music is static. Using an event-driven observer method may not be necessary. A simple code, AudioManager.Play(), could be enough for each scene. | /4  /3  /3  /4  /4 |