|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Cory Clairmont | Team | BrainStew | TL | 1 | Date | 4/14/25 | Time | 1608.00 |

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 focus for this project was sound design. I created sound effects based on actions, animations, and collisions. I also managed background noises and music for bosses and levels. Besides this, I helped with enemy design, coding, and bug fixing.  **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?**  My gantt chart was very wrong. I made it when I had a poor understanding of how long different tasks would take so I spent a lot of time on small tasks and much less time on others. In the future, I will give a lot more leeway with time and simplify my tasks, so the prerequisites are less convoluted.  **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.)**  My test plan is to ensure that the volume features don’t break and that many audio sources at once can’t break the game. These are the main features which I am responsible for and the entire bounds of it. I did this by setting the volume to a way to high level to make sure it doesn’t break and to zero to make sure there is no divide by zero. This all went well, and I ran extra boundary testing based around the volume slider in the menus to ensure the tl6 in charge of menus does not cause issues.  A scenario where I tested to ensure that my future code wouldn’t break the game was in a similar way. By ensuring that volume doesn’t go to high or low and break the game (divide by zero error), my paralaxing 2d sound script ran perfectly without major errors, where it would have had a divide by zero error.  **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: AudioManager  In my prefab documentation, I was mostly trying to show where each audio source was used. For backgrounds I listed which scene each of them belonged to, and for sound effects I pointed to a spot which has an accurate updating instance of which audio source belongs where.  **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: AudioManagerVolume  Sub Class: DynamicAudioManagerVolume  Virtual Function: setVolume ()  public class AudioManagerVolume  {  public virtual float getVolume()  {  return 1f;  }  public virtual void setVolume(float VOLUME){}  }  public class DynamicAudioManagerVolume : AudioManagerVolume  {  public float volume = 1f;  public override float getVolume()  {  return volume;  }  public override void setVolume(float VOLUME)  {  volume = Mathf.Clamp(VOLUME, 0.01f, 1f);  }  }  **Choose a dynamically bound method. What method gets called now?**  manager = new DynamicAudioManagerVolume();  manager.getVolume();  since getVolume references the dynamic class, it will call the dynamic method.  **Change the dynamic type. What method gets called now?**  manager = new AudioManagerVolume();  manager.getVolume();  since getVolume is not referencing the Dynamic type, it will default to the base value set in AudioManager and will not pay attention to any of the dynamic instances.  **Pick a statically bound method. Which one would be called in each of the two previous cases?**  instance.audioVolume.setVolume(1f);   In my start function, I am referencing audioVolume. Since it is dynamic, it will be changeable and set to 1f in this instance. It also means that in other instances of this call that it will be able to get this volume return as well since the virtual variable is stored at runtime and override is called.  If this was referencing the not dynamic version, it would not be able to be changed like this. It would default to the value already set in the initial class.  **Show me an example of reuse in your code where you violate copyright law.**  **How does it violate copyright?**  I have multiple instances of music which are creatively owned by multiple different artists. It is the creative property of another person.  **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**.  The background music is instantiated at the beginning of each scene. The legal implication is that a copywrite claim could be given to the project. This may include removal of the game from distribution platforms or potential legal action depending on the rights holder. For fair use, I could argue that the popularity of the game and its soundtrack would greatly increase sales and streams of a song, directly benefiting it. Besides this, the use of the song can be considered transitive. This is because the experience of the game and actions in it provide a unique and interactive experience that is different than passively listening to a song.  **4. One big or two small, well-chosen patterns.**  **Small Patterns = {Singleton, Private Class Data}**  **Which patterns did you choose?**  1.One big Singleton  **Why did you choose each pattern? (Justify your use of it).**  I chose the singleton pattern because having a centralized manager for audio calls would drastically simplify making audio calls and managing volume. If audio calls were spread between everywhere that needs to make a call, it would need to reference the actual audio source every time, ensure that it has it, and then check for volume variables, and run the sound. This is drastically simplified in the audio Manager class, all that needs to be done is to call the class, with the sound type (which is provided in the description) and potentially a volume modifier.  **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?**  This is the perfect time to use this pattern. An instance where it might be bad to use a singleton would be any object that needs to get cloned over and over in the same scene. A prototype or factory type would be much better in this instance. | /4  /3  /3  /4  /4 |