Name	Lalith Aditya Chunduri	Team	Error 404	TL	5	Date	April 18	Time	8:00am

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: ***I created Whole Graphics for Our Game, created a Help Menu for the Game and created an animated key which we used for all three 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? ***It got me 36hrs, but I assumed It may take 37hrs. I am assuming my estimation was right. But I encountered and wasted some time creating the same things which I created due to some git issues.*** 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.) ***I Created an animated Key where the player needs to collect to go to next level or to win the game. If the teammate's participation incorrectly influences the key animation, an error in regression could occur. My Tl02 added a sphere as an key firstly before I added the Key animation later I added it but when I tested if it works or no it got different and the key is not being picked instead moving by the touch of player moment.***	/4
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: ***Help Menu***	/3
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.	/3

Super Class: **Pause Menu**
Sub Class: **Main and options menu**
Virtual Function: **public virtual void Resume()**

Choose a dynamically bound method. What method gets called now?
The variable dynamicMenu is declared with a static type of PauseMenuBase, but it's instantiated with a dynamic type of Pausemenu, which is a subclass of PauseMenuBase.

Change the dynamic type. What method gets called now?
If we change the dynamic type to a subclass that overrides the Resume() method, the overridden version of the method in the subclass will be called. Let's change the dynamic type to another subclass CustomPauseMenu that overrides the Resume() method /4

Pick a statically bound method. Which one would be called in each of the two previous cases?
Let's choose the Pause() method as the statically bound method. In both of the previous cases, since the Pause() method is not overridden in any subclass, the statically bound method will always be the one defined in the Pausemenu class.
Show me an example of reuse in your code where you violate copyright law.
How does it violate copyright?

I learnt most of my design or assets from web where it states for educational reasons only and we may use it for our personal or while learning things, and because they are free, I Assume we can download and use them in our game As they mention in their terms. /4

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?

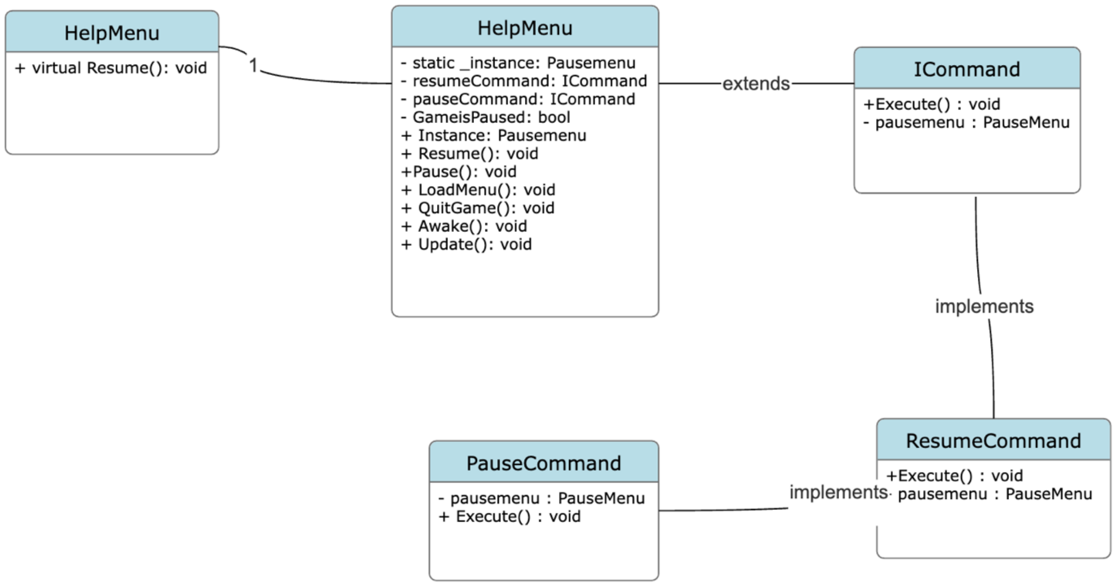
1. ***Singleton: Singleton Pattern: This pattern ensures that there's only one instance of a class throughout the application's lifecycle. In the code, the Pausemenu class is designed as a singleton to ensure there's only one instance of the pause menu system in the game.***

2. ***Command Pattern: The Command pattern encodes requests as objects, allowing clients to be customized via queues, requests, and actions. It encourages loose coupling by enabling instructions to be executed without explicitly linking the invoker and recipient. In the given code, the ICommand interface offers a single Execute() function, whereas concrete command classes (ResumeCommand and PauseCommand) contain activities such as resuming and halting the game. This approach allows for simpler extension and increases flexibility in handling game actions..***

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

1. **Singleton Pattern**:
    - **Justification**: Chosen to ensure there's only one instance of the pause menu system, preventing conflicts and ensuring consistent management of the game's pause state and UI elements.
2. **Command Pattern**:
    - **Justification**: Chosen to encapsulate the resume and pause actions as separate command objects, promoting decoupling between the invoker and the receiver, thus enabling flexibility and extensibility in managing game actions.

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?

For the Singleton pattern:

- **Alternatives**: Dependency injection or service locators may provide similar functionality without the constraints of a Singleton.
- **Bad Time to Use**: Avoid when dependency injection or global state management via service locators is preferred, or when it leads to tight coupling and global state issues.

For the Command pattern:

- **Alternatives**: Direct method calls or callbacks may suffice for simple scenarios, while the Strategy pattern might be more suitable for dynamic strategy selection.
- **Bad Time to Use**: Avoid when commands are simple and don't require separate objects, or when the overhead of creating command objects outweighs the benefits.