Solution Answers and Documentation

Contents

Answers & Documentation

Question1.

How would you implement this scenario in a modular and generic way with seamless transitions between the scenes?

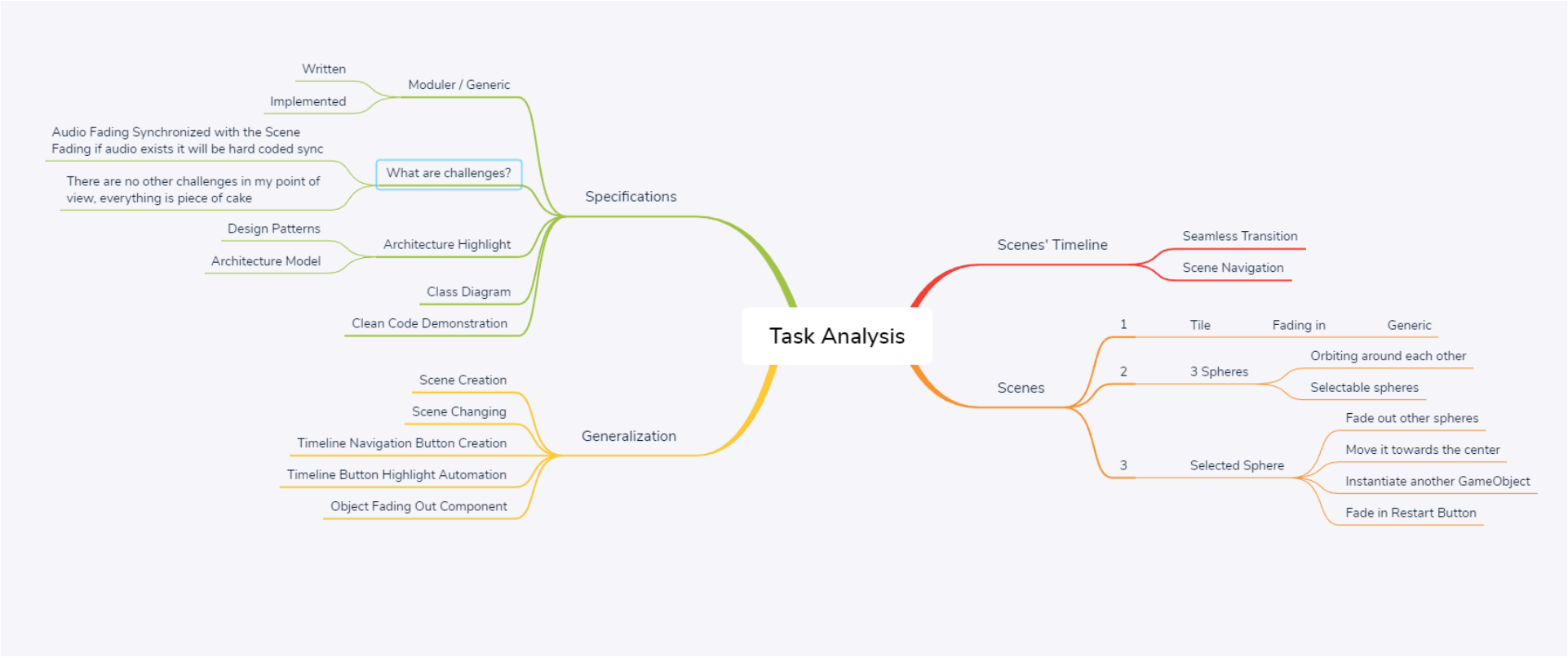
It is very simple to handle the transition between scenes with different fading out/in transitions.

I would implement it through using the animation system which is built in unity and will make a blank white/black image which will be a child of canvas that has highest layer priority on the screen that has an animator which at any time we are going to change the scene we will play the fade-out animation and wait until it finishes (fully black or white) then change the scene and at the On Loaded Scene event will play the fade-in animation of the same image. Using the same technique we can fade out specific object(s) which uses animation to fade out, but instead of changing the opacity of an image we can change the color or even the alpha of the color of the material if it was set to fade render mode.

Question2.

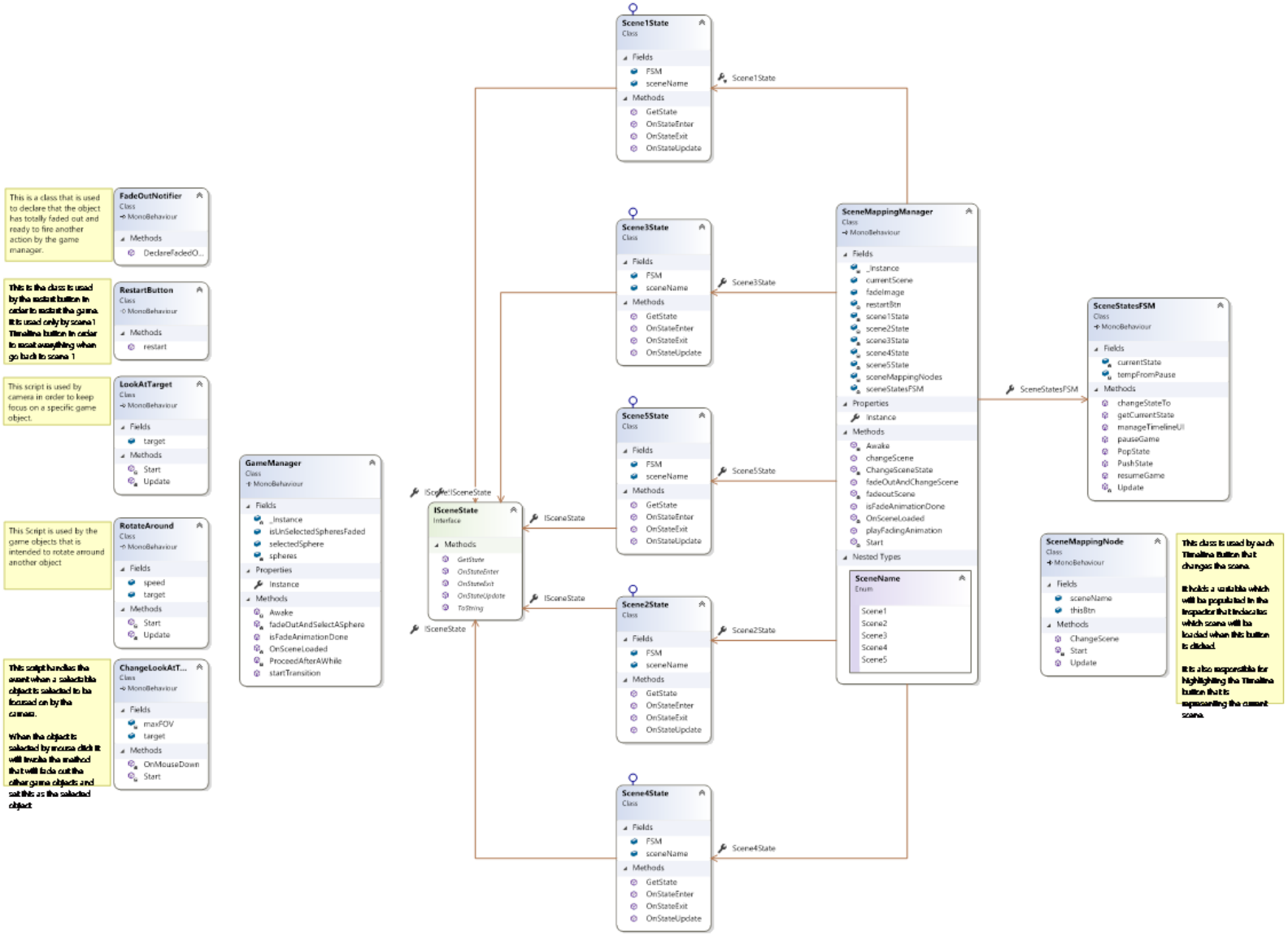
- Block out the basic architecture of your solution in code.

Task Analysis: (fig.1)

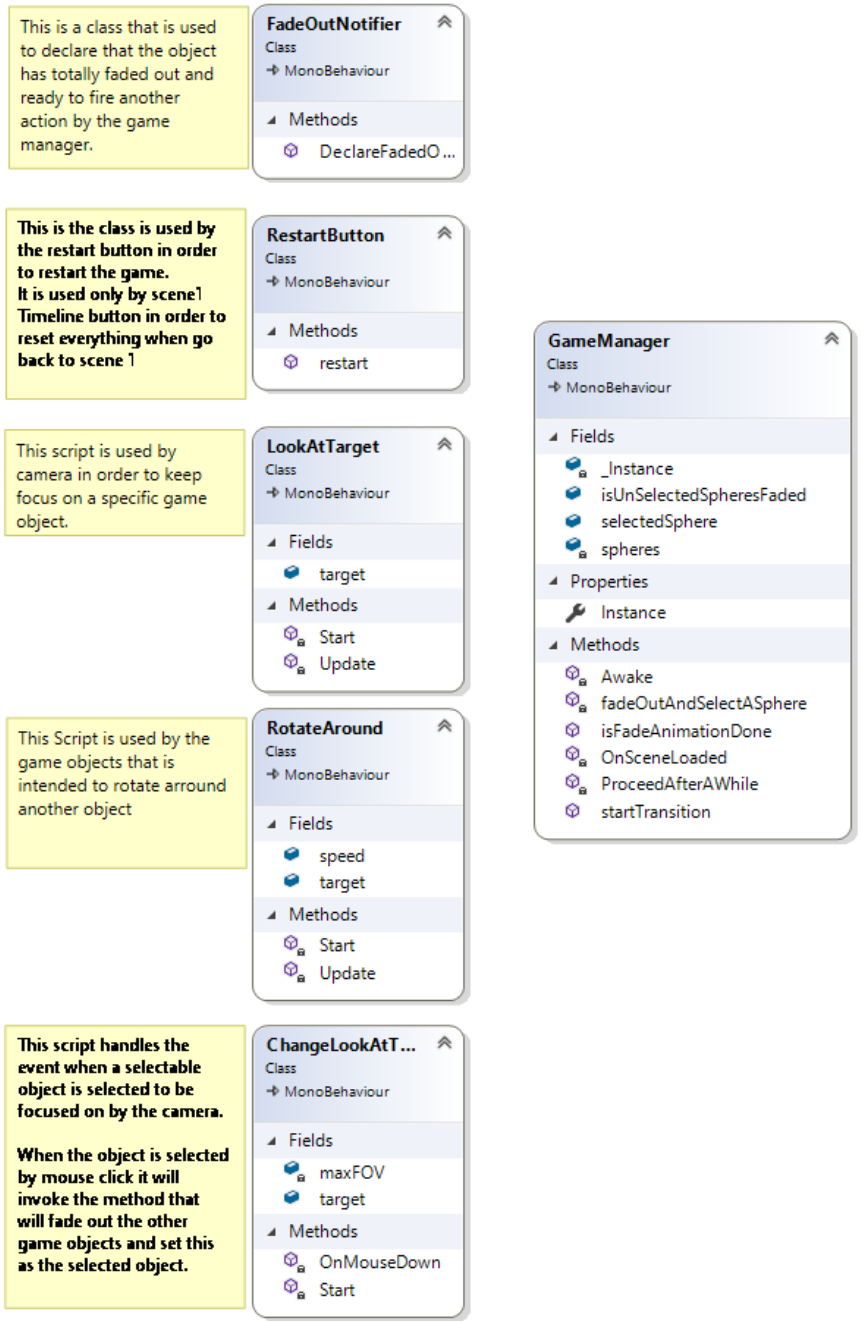


Solution Basic Architecture

An Overview: (fig.2)



Game Management and independent scripts: (fig.3)

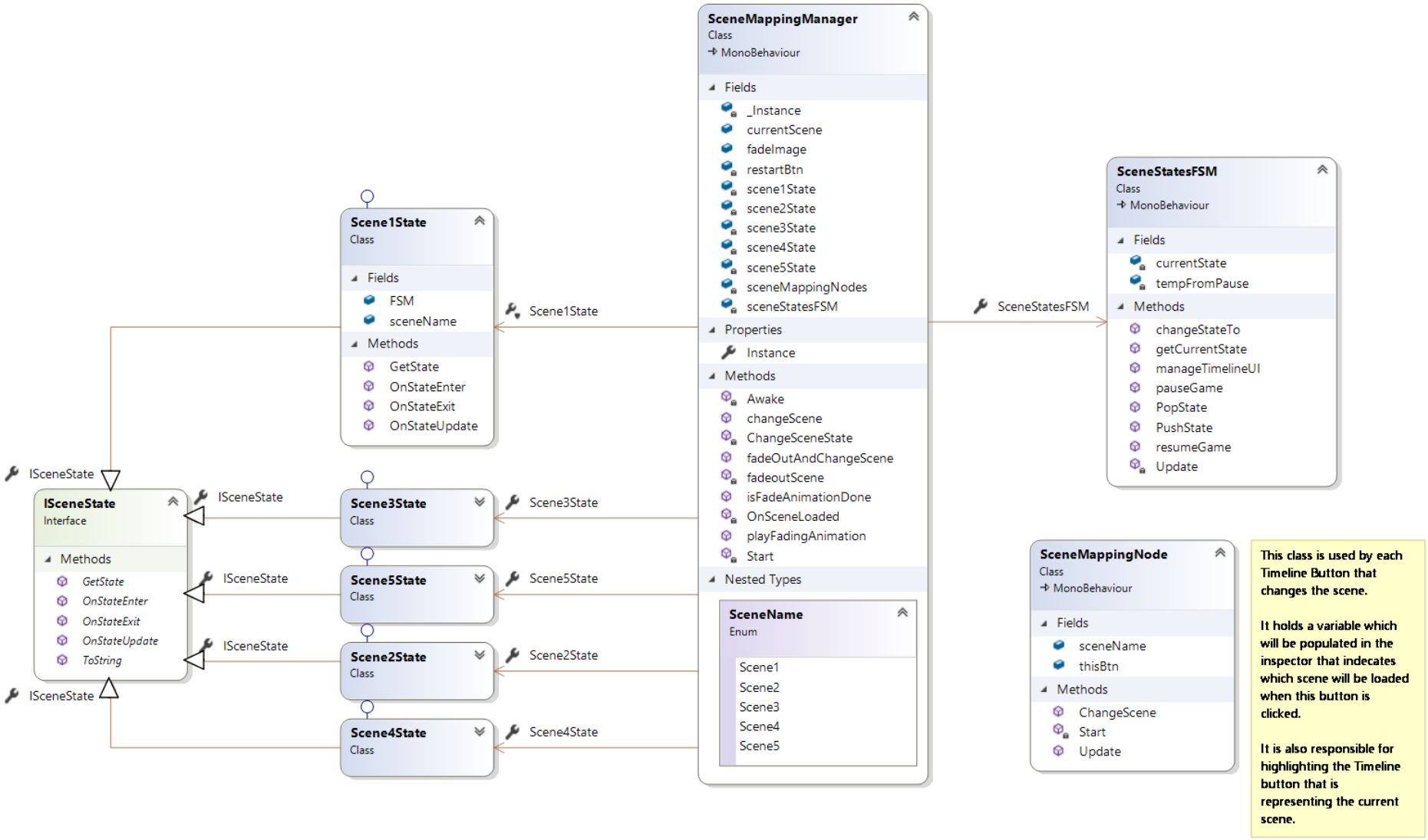


Game Management and independent scripts:

- GameManager: Doing general gameplay processes and holds the shared data members that needs to be shared.
- FadeOutNotifier: Notifies the game manager that specific game objects need to be faded out is completely faded.
- RestartButton: A script that is used by the Restart button to restart the game and the Timeline button of scene1 to reset the logic.
- LookAtTarget: Used by the camera to keep focus on a specific gameObject.
- RotateAround: Used by an object that is needed to rotate around another object.
- ChangeLookAtTarget: Reflects the mouse clicks on an object and perform a specific action including changing the Look at target to be the pressed object.

Note: GameManager isn't the best version of it but because it is not the subject of the test to enhance and design everything 100% so we will go with it as it is.

Scene Management & Scene States Finite State Machine (fig.4)



Scene Management & Scene States Finite State Machine

- ISceneState <<interface>>: standardize the state with a generic behavior.

- SceneXState: Implements the SceneState and the logic inside interface's methods, the scene state is made specifically to handle events needed to be handled dynamically in a very robust structured that fits different scenarios depending on the gameplay.

- SceneMappingManager: Doing general scene mapping and management and holds the shared data members that needs to be shared. Handles the general purpose fading mechanism.
- SceneMappingNode: it represents the Timeline button

- SceneName <Enum>: it holds all the scenes' names with exact same name of the scene at the project.

- SceneStatesFSM: the handler that change from scene state to another.

  Note:

  - Scenes must be added to Build Settings.
  - Timeline Buttons: Disabled/Enabled on purpose that fits the test scenario with no logical (e.g. un expected behavior out of the specified scenario)

Question3.

Challenges

- The Mix between the Generalization and the specification of each scene it is little bit holds details that need to be considered.

- ChangeSceneState() -> SceneMappingManager -> could be maintained to be generic and more professional, but it is not the subject of the test so no time for it now (Prioritization).
- Time isn't enough to me so I hadn't time to enhance the game manager, there should be an independent class that encapsulates the logic, sort of behavioral design pattern and then the Singleton Game Manager just uses it.

- Synchronizing audio with fading out animation it needs to be made by hand in order to be homogenous with fading out transition.
  Sometimes work deadlines doesn't give the space for the best software analysis and design.

- Off course the good design will make everyone life better but unfortunately it might not be very useful at the beginning, but it will take more time and effort and the effect at the beginning isn't much, which will make other "Game Developers" bothered because they always choose the easy choice, they're not a software engineers.

Question4.

Implement some methods in a way that you consider to be clean code.

Figure 4: each component in this figure is pretty clean and well designed. So check the code out of each of it.

Design pattern used:

- Singleton:
  - Game Manager.
  - Scene Mapping Manager.
- State Pattern:
  - ISCeneState.
  - Scene1State, Scene2State, Scene3State, Scene4State, Scene5State.
  - SceneStatesFSM.
- A concept from the bridge pattern:
  - SceneMappingManager -> Abstract Part.
  - SceneMappingNode -> Decoupled application, implementation.

Conclusion

Proof of Generalization and Modularity:

- Create a new scene.
- Rename it with the name that you want and copy it retaining case sensitivity.
- Add the scene name at the Enum -> SceneMappingManager.
- Create concrete class for the “new scene”state -> Implements ISceneState, add a reference to the SceneStatesFSM, add Enum variable of type SceneMappingManager.SceneName and initialize it with the scene name that we have just added it to the Enum -> SceneMappingManager. Determine what behavior will happen on the scene enter, update, exit.
- Create and object of the “new scene”state class at SceneMappingManager -> initialize it at the start method.
- ChangeSceneState() -> SceneMappingManager -> Add case to the switch that will check if the loaded scene is the newly created scene and add the line: sceneStatesFSM.changeStateTo(scene5State);
  In order to implement it at this case.
- At scene1 at the Management Scripts and Components -> Timeline Scene Navigation GameObject -> Panel -> Add a new connector and new Timeline button for the newly created scene, add a SceneMappingNode to the new Timeline button and choose from the inspector the intended scene. Add the script to the button actions and choose changeScene() -> SceneMappingNode.