Exam Theoretical Explanations

Nicholas Xynos 100782842
1 + 0 + 0 + 7 + 8 + 2 + 8 + 4 + 2 = 32 = EVEN
(Using the TA's starter project, and code as a reference)

I did not code any of this in, I'm explaining how I would do it *(TA's code was referenced)*

Question 5 Explanation:

I will be explaining how to increase the amount of robots in the scene (I'm using the TA's starter project, and she has robots instead of ducks in the scene), when running the game.

First I would create a script called "ObjectPooling.cs", and inside this script, I will give it the robot prefab and then instantiate how many robots I want in the scene (for an example I will set it 10), and then store this information in a list, because incase one doesn't spawn it will reference the list and spawn in one of the deactivated robots to the scene and always have 10 at a time (unless you manually place more, or increase the limit in the list).

Object Pooling benefits and greatly optimizes the scene because it not only reduces how much is being loaded individually (meaning it will check every object individually if you don't use object pooling), but it loads everything in at once which cuts down the load times. On top of this, it also saves time when placing objects individually, it doesn't need to only be for robots in this case, it can also be for say, platforms, or other environment assets needed in the scene, and it will save so much time.

Question 6 Explanation:

I will be explaining how I would have implemented how to change the player's looking controls when they miss hitting an enemy twice (the camera will be inverted if you miss an enemy, so looking up will make you look down, and looking down will make you look up)

First I would need to create a counter, for whenever the player misses the enemy, and to do this I would create a script called "EnemyDection.cs", which will have an OnTriggerEnter() function to determine whether or not the player hit an object with the tag "enemy" (I will put a tag on every single enemy), if it does, add 1 to a counter and

attach this script to the player.

Then I would start by implementing a public interface name “ICommand.cs”, which would have an abstract function called “Execute()”, that will be inheritable by other scripts and classes.

I would then make an invoker script named “CommandInvoker.cs” which would have a queue created in it for the list of commands that it will need to execute, and the queue will be called CommandBuffer. Still in CommandInvoker.cs, there would be an “AddCommand()” function that will dequeue a command (remove it from the queue) and then call the execute function from ICommand to then execute said command.

Moving on I would then create another script called “ControlInvert.cs”, and create a function called “InverseControls” and that function would have an if statement, and if the counter is greater than or equal to 2 (the counter to determine whether or not you hit the enemy), transform the position whenever you look to be inverted (looking up will make you look down, and looking down will make you look up).

This benefits the game and optimizes it because it would take some load off of the CPU, also for pure organization purposes, you wouldn’t be using tons of different scripts, and on top of this, things can easily be changed, modified, and expanded upon extremely easily with the command pattern, and will allow the user to add or remove however many functions they wish without it being messy or having to worry about optimization when it’s already been done.

Question 7 Explanation:

I will be explaining how I will change the colour of the robot enemies whenever they spawn into the scene and named “Robot2” using an observer pattern (again this is using the TA’s code and final exam template)

First, I will create a script called “RobotColour.cs”, and create a public function called “RobotMaterial()”, which will change the default robot colour to blue (RGB value would be 0,0,255), and this function can be inherited by other classes/scripts.

Then I would make “Observer.cs”, which is the most important script since it will tell the robots to change colour when the observer gets notified. The script will have the abstract class for the observers to inherit called “OnNotify()”, and it will inherit from RobotColour(), and override OnNotify() to change the Robot Colour to blue.

We will make a third script called “Subject.cs” which will inherit OnNotify, and notify all of the observers that an enemy has been spawning, and keep a list of all of the observers to know how many to notify/call whenever it needs to change the material on the robots.

“SpawnEnemyButton.cs”, which will have the function “SpawnRobot()”, which will have the prefab for the robot set into it, it will add an observer to it, which will send a notification to the observers attached to the robots (from the list we created in Subject.cs), and if it’s named to “Robot2” it will add the blue material set in the RobotColour.cs script, otherwise, leave the material to it’ default colour.

This benefits the game just by adding some variation to the enemies, this also allows it so that if you want to in the future, make it so that anything named “Robot2” can have specific attacks or different attributes, and they can easily be added onto the observer OnNotify() function which makes things a lot more efficient, and also saves time on spawning specific enemies.