Question 5

# Object Orientated principles in unity and game development

***Inheritance :*** This is the core concept of object orientation programming it allows the sharing of methods and cold and variables amongst super classes and its children are inherited members

Inheritance is omnipresent in all unity projects and in all game projects. Every script in unity inherits from a mono behaviour script which inherits multiple classes methods and variables useful within the engine. For example The transform component can be accessed by a transform variable we have functionality like vector 3 and start update et cetera

***Polymorphism:*** This concept allows us to group multiple instances of a given superclass together even though the objects themselves may not be the same class . for example we might have an item super class in which we have various information about the weight the description maybe some imagery of the object , add methods that allow us to pick it up drop it etc. Everything in the game is an item, But we may create classes inheriting from the item class such as weapon armour food drink et cetera . As there are items they all will have a weight, a description, all of the information that stored the super class item, But may have other information and methods such as DPS for weapons and armour rating for armour. Polymorphism allows us to create a single array or list of item class In which we could push all of the objects that inherit from item .

***Interface :*** Interface is and beings of demanding your requirement For how you interact with that object in a given context. The most obvious interface concept for games would be a health interface. Typically if you have something that has health you would expect to be able to damage that object thus reducing its health so if you were the damager you would need to communicate with the object to say take damage. So all an interface does is to set a requirement that a given method in this case take damage has to be included in the class if that class is using the interface health

***Abstract class*** the definition of an abstract class is a class that has An abstract method. An abstract method is a method that has no code only a declaration of a return type and the type of parameters. The idea behind this is that you create a an abstract class a class that can never exist in the game only through concrete inherited instances. For example, we could create an abstract class called character for our game this character might have movement code along the lines of if should turn left turn left if should turn right turn right it should move forward move forward et cetera et cetera .you would expect all characters should within reason all move in a similar fashion So the code to make them turn and move and walk and jump it should all be the same. The only thing that differs would be the reasons why they should move in turn and turn et cetera. So the player obviously would move and turn on key press and or mouse movements, whereas an npc would only move under the direction of some ai. So if we made the should turn left and should turn right and should move forward methods abstract and all we filled the code when we implemented them as the inherited player from character are the inherited NPC from character then All of the cold that works for making the move works in the base character and then we can rewrite the should move left should move right in any way we want

# Communication between scripts and unity

There are multiple ways that game objects can communicate with each other the most useful way is obviously through script communication.

Game objects can connect in a number of ways

* Collisions
  + If the objects have colliders then collisions may be detected,And through the collision parameter within the on collision enter method The collider and game object can be accessed, and from these the scripts of that object can be got through a get component
* Public variables
  + An object in the scene or a prefab can be directly dropped onto a public variable of a script, and this may be accessed directly from within the script
* Send message and broadcast message
  + These It can send a message to a specific game object looking for a specific method within that game object or to all game objects in a given set of objects again which contain a specified method
* Through find object
  + Similar to get component doesn’t require a search within a given game object but looks at all objects in a given scene so typically you could find an object of type <A given script> and it will return a list of all those scripts in the scene
* Caching and seeding at instantiation
  + When an object is instantiated by say a spawner or a manager the manager has access to that object and can pass information to that object informing that object that I am the manager. The new object can then cache that information and access the manager or communicate with the manager at a later date

# Group projects using unity and Github

# Initial setup

First the repository has to be created with Github. As we know it’s going to be a Unity project and a group project we ensure that the project is a public project and the Github is specifically designed to be a repository for a unity project we choose a unity git ignore file from the pull down menu. Once created we then get the url, and this allows us to clone our empty repository onto our desktop. Typically here we open a Git GUI application, choose clone repository then paste the URL Into source and select the location on our desktop that we want the repository to be located as our destination. And the empty folder should be created on our desktop mirroring our repository on Github.

An empty unity project is now created, Once this is done we close off unity and any IDE, (Visual Studio etc.) that we may have open. We then cut and paste the entire folder containing this unity project into our repository folder. This repository folder should already include a .gitignore file. This file needs to be moved into the root of the unity folder, In other words the same folder that you would see the asset folder in.

We reopen our git gui , and refresh. We should see a number of files here ready for staging, Typically in the 30s but if you have thousands please make sure that your git ignore file is in the correct place before proceeding. After this we <Stage Changed> <sign off ><commit> and then <push>

After the standard pull requests from master to main on our github account the project is set up for the group to contribute to.

If people wish to contribute to an open source project Firstly they will need the URL to the repository in question. So whoever has set up the unity project just above will distribute or make available the url of the repository.

Anyone who wishes to contribute to the project must firstly fork the project, This creates a linked copy of the repository in their own Github account. It is important when cloning the repository that the repository is cloned directly from their personal gitHub account and not from the group gitHub account.

The contributor should now have a perfectly synced copy of the newest version of the project on their machine and on their Github account. Work may now be done locally on the project.

Once work is done for the day, The contributor should close off all versions of unity and visual studio, then return to gitGUI and <Stage Changed> <sign off ><commit> and then <push>

This sends the work they’ve done up to their personal repository. They should now access their personal repository through a browser and make a pull request updating their main from their master. After the master branch is deleted their main branch is up to date with the work that they have just done. A pull request is made From the main of their repository to the original group repository.

# Typical work session

For typical work session we have already contributed to the project and we are assuming others are contributing as well.

We will assume that the previous day a pull request was sent to the group repository and that everything is up to date as if we had finished the initial setup above.

Before we can start working we have to ensure that we are up to date everywhere that means we are up to date with the group repository in both our local repository of our machine and on our github linked repo.

There are two ways of doing this we can update our local version first and then send our changes up to our Github repo or we can update our Github repo and bring our changes down to our local version

The easier of these two options is the latter which is the one we will cover here. To update our Github repo, we just need to sync it with the original fork, This is done through the sink fork button . Once this is done our github repo is up to date but our local repo is behind.

The easiest way to update as opposed to clone out local repo is as follows:

In Windows Explorer we need to locate the local repository and right click on it, This should open up a menu you may need to show more options but we need to select Open git bash here. This should open a command line window for Git .

We should perform a couple of quick checks here, Firstly is the directory as described in the grey bar at the very top of the window as you would expect. Secondly is there a blue master in brackets at the end of the first line in the command line interface. If so it is telling us that it recognises the folder that’s been selected as being a Github repository or a local Github repository.

It is possible that it could be linked to a different repository so to cheque this we type

git remote -v

And press return once we do this it should give us the URL that this repository this local repository is linked to in the cloud.

Typically you will see the word origin with the URL of your repository

To bring down changes locally from our Github repository we type

git fetch origin

Once you have done this you may see some lines of feedback and typically the very last thing on the last line that will be written will usually look something like “origin/main”

To incorporate these changes into our local working copy we need to merge them to do this we type

git merge origin/main

Whatever we type at the end here as in origin/main will be the last part of the last line of the feedback as discussed above.

Once this is done all changes will be updated on your master copy on your local machine and you are free to work.

At the end of the day once all work is done we repeat the processes above.

The contributor should close off all versions of unity and visual studio, then return to gitGUI and <Stage Changed> <sign off ><commit> and then <push>

This sends the work they’ve done up to their personal repository. They should now access their personal repository through a browser and make a pull request updating their main from their master. After the master branch is deleted their main branch is up to date with the work that they have just done. A pull request is made From the main of their repository to the original group repository.

# Management, pull requests and merge conflicts

At the end of the day or more likely throughout the day, The owner of the repo or the manager or the project manager will log on to their group repo account. All pull requests sent to him will be available on the pull request tab.

It is the manager’s responsibility to review each of these pull requests and to determine whether to accept them or not. If the manager wishes to accept the request there may still be issues that have to be addressed. At this stage it would be assumed that code would work so the biggest issue that managers would have to face here at this stage would be conflicts. Because we’re working on a group project multiple people accessing same files it is possible that one person might want to for example call a variable or rename a variable X for another person might want to rename the variable Y, this is all really simplified but effectively you have two suggestions for the same place in code or two possibilities for the same place in court and it’s up to the manager to side which one if any to accept. Merge conflicts in code are usually fairly straightforward , An examination of the code and provided by both will usually lead to a very simple solution where one or both are used .

The nightmare scenario is when multiple versions of unity may be used amongst the contributors and in this circumstance the number of conflicts can increase exponentially because it’s computer generated code that is producing the conflict as opposed to people these are situations which we want to and would try to avoid at all costs For a manager the easiest thing at this situation would be to just completely drop a pull request and possibly send a message saying update or use this version of unity before you submit a pull request again