# Assignment 1 – Game Design Patterns

## INFR 3110U – Game Engine Design & Implementation

Bill Ko 100590491

Stephen Saunders 100621189

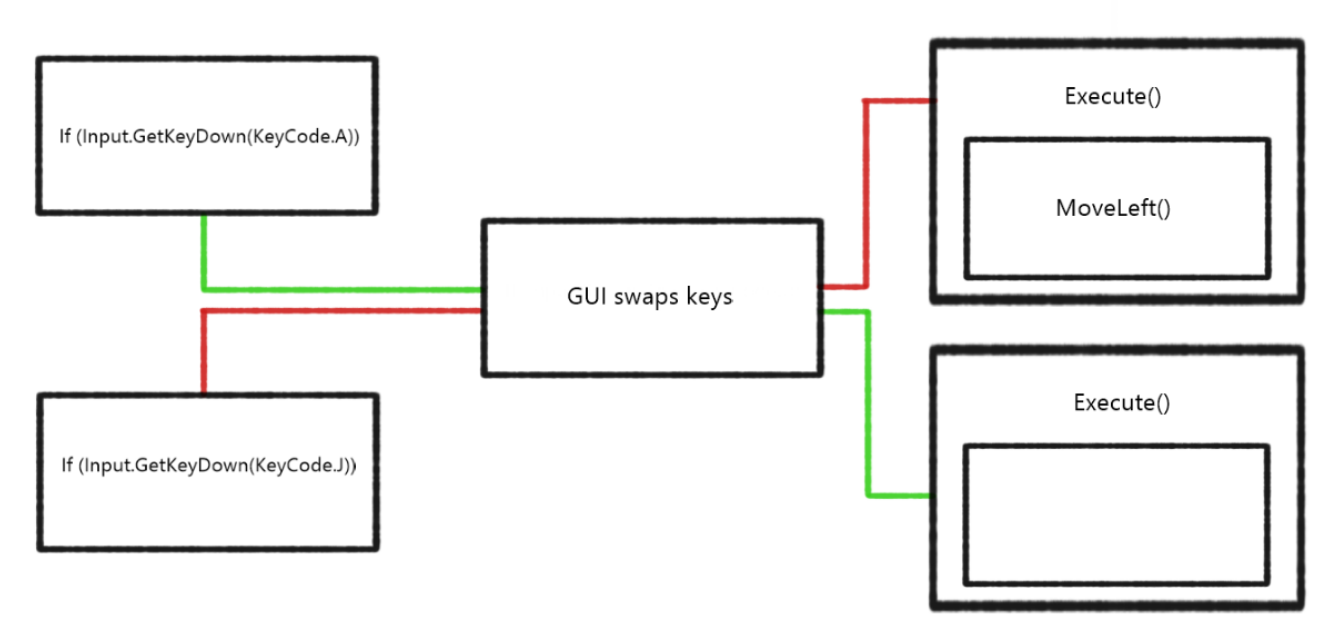
## Command Design Pattern

The command design pattern is formed when you take a request and encapsulate it as an object. In this specific case we used it to log requests into a list, allowing us to undo or replay any changes. By encapsulating the request to move the player (for example; ‘A’ key pressed = move player left), we were able to allow the user to change the key bound to that object. To do this we needed a ‘Do Nothing’ class to allow for the switch of the keys.

Before:



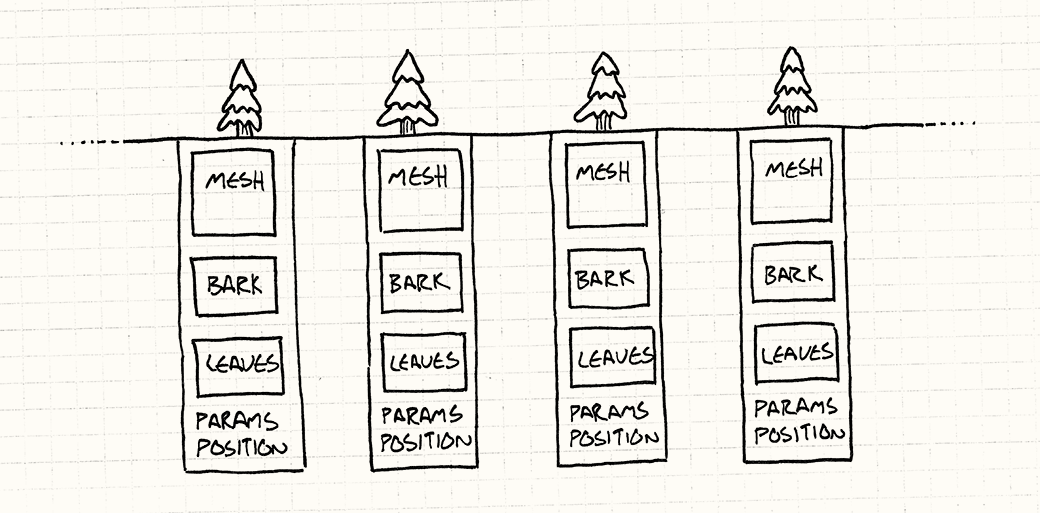
After:



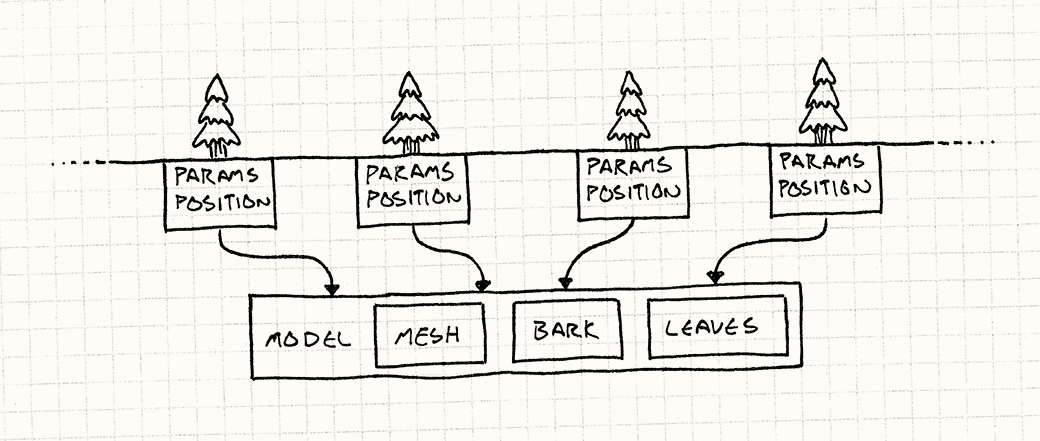
In this example, the first picture is before the command pattern was implemented. In it the MoveLeft() function is simply called from an if statement for the “A” button being pressed. Since we wanted the user to be able to swap the keys, we encapsulated the MoveLeft() into one object, and made a class that does nothing when it is called. This way, when they change it in the GUI, the key that is no longer being used has somewhere it is directed to. Now that MoveLeft() is encapsulated, it is easy for us to add calls to this request into a list, allowing us to undo to previous movements.

## Flyweight Design Pattern

The flyweight design pattern is used to save memory by sharing information between large numbers of objects that are nearly identical. A good visual example I found is from <http://gameprogrammingpatterns.com/flyweight.html>.



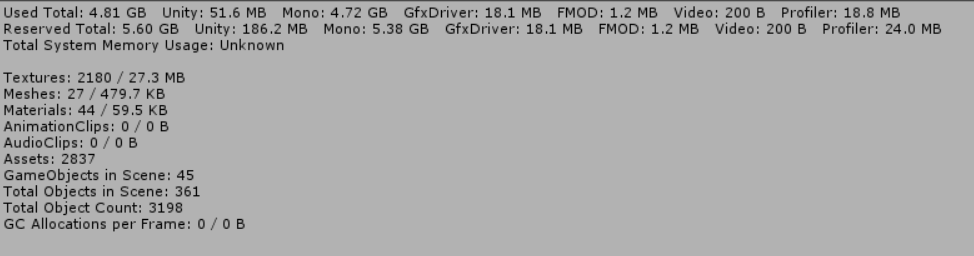
This is how they describe creating multiple objects without the use of the flyweight design pattern; each object has its own mesh and textures despite the mesh and textures being the same. Why needlessly keep re-initializing the mesh and textures when you can do it once and share it between the objects? This is the philosophy behind the flyweight design pattern and can be visualized as such;



In this example, each tree has a different location, so that information cannot be shared, however, they all share the same model and textures, so that information was compiled, now only being initialized once for each instance of the object, allowing us to save a lot of memory.

The effects of the flyweight design can be seen within our own tests, pay attention to the “Used Total” and “Total Object Count” in each screen-capture.

Before:



After:

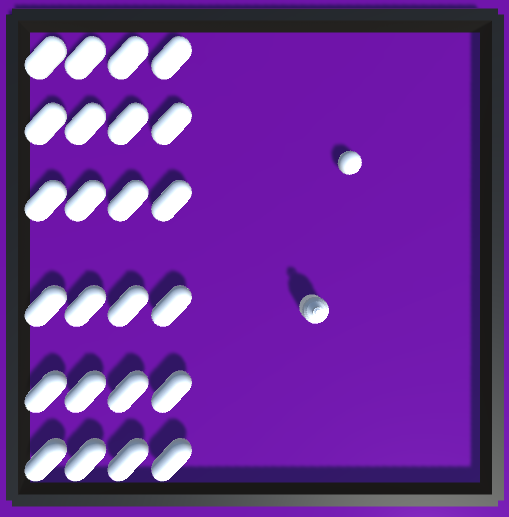
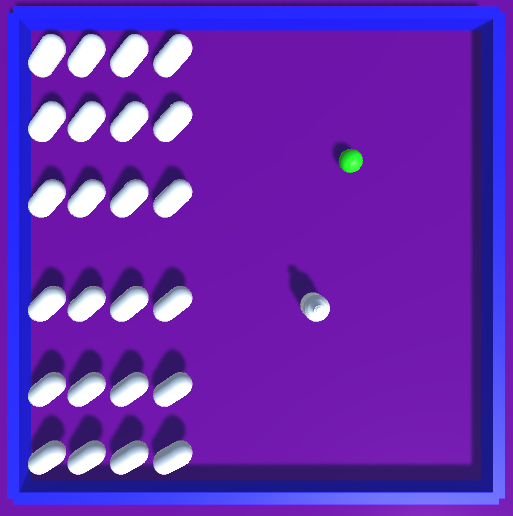


Without the flyweight design pattern, at an object count of 3198, it consumed about 4.81GB, which is a stark contrast to the 109.5MB that was used for 3179 objects while using the flyweight pattern. To test this in Unity we had it create multiple empty objects, which is why it isn’t visually represented in the game, but instead only in the memory analytics.

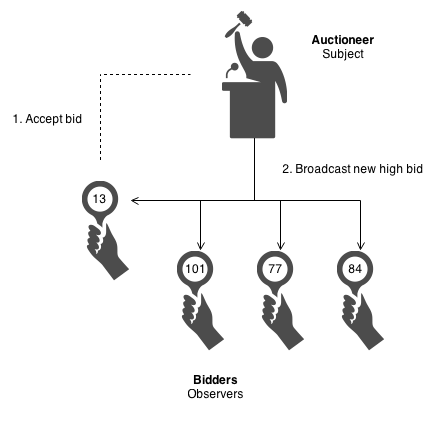
## Observer Pattern

The observer pattern is exactly as it sounds; an object/subject is ‘observed’ by its dependents, and whenever the state of the subject is changed, the observers/dependents are notified. In our assignment, the state of the player controlled sphere is changed; its color by simply pressing Ctrl. The observers watching the sphere are notified about the change, or more accurately, the sphere (otherwise known as the subject) updates the observers on the change.

Once the color-state of our player-controlled sphere is changed, it updates its determinants, and as a result, the color of the outside border of our scene changes.

In these screen-captures you can see the changed game-state of each object. The sphere is the subject, that has its color changed to green, and the border is a determinant of the sphere, that is updated at the change and as a result also changes its color; this time to blue.

 A good example of the observer pattern comes from <https://sourcemaking.com/design_patterns/observer>, in which they describe an auction. In the example the auctioneer is the subject, being observed by the bidders. The auctioneer (subject) updates the bidders (observers) on the changed state, being a new highest bid. As a result of hearing a new highest bid, the bidders may also change their state; raising the bid.

## Pseudocode/UML

The pseudocode for the Observer pattern is as follows;

**class Subject{**

**list of Observers;**

**void attachObserver(Observer1){**

**Observer1 attaches itself to the list;**

**}**

**void Notify(){**

**for each (Observer in list){**

**run Observer.response();**

**}**

**}**

**}**

**class Observer{**

**Observer(){**

**Attach to list;**

**}**

**Respond(){**

**Do whatever it was supposed to;**

**In our specific case, change colour of an object;**

**}**

**}**

**class Game{**

**create Subject;**

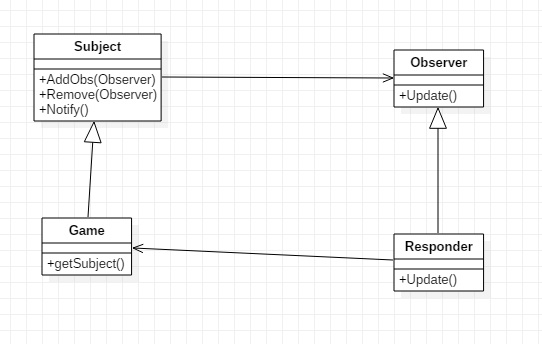
**if (X is true;){**

**Subject.Notify();**

**}**

**}**

And the UML Diagram is also as follows;



## Citations

The majority of the code we used for the command, flyweight, and observer patterns came from the below citations, respectively, and were edited and elaborated upon for the final submission of the project.

Command Pattern Tutorial:

Nordeus, E. (n.d.). Game programming patterns in Unity with C# - 1. Command Pattern. Retrieved September 29, 2018, from <https://www.habrador.com/tutorials/programming-patterns/1-command-pattern/>

Flyweight Pattern Tutorial:

Nordeus, E. (n.d.). Game programming patterns in Unity with C# - 2. Flyweight Pattern. Retrieved September 29, 2018, from <https://www.habrador.com/tutorials/programming-patterns/2-flyweight-pattern/>

Observer Pattern Tutorial:

Nordeus, E. (n.d.). Game programming patterns in Unity with C# - 3. Observer Pattern. Retrieved September 29, 2018, from https://www.habrador.com/tutorials/programming-patterns/3-observer-pattern/

The base of the project, including the interactive game scene, was created with the aid of an official Unity tutorial;

Unity. (2015, April 17). Unity 5 - Roll a Ball game - 1 of 8: Setting up the Game - Unity Official Tutorials. Retrieved September 29, 2018, from <https://www.youtube.com/watch?v=W_fAidYRGzs>

Any models or meshes used were either default meshes already a part of Unity or our original product, namely the potion used in the scene.

The version control we used for this project was GitHub, and can be downloaded at this link; https://desktop.github.com/