# EECS 3311: Software Design (Section A)

Name: Rushilkumar Patel

Student ID: 216870057

EECS Login: rushildp

For Page 1  and Page 2 content check docs folder of the submitted project.

Page 3:

## 1.Section: Enemy Action

All the enemies have preemptive and non-preemptive actions.

In my implementation I have a deferred ENEMY class with deferred features preemptive action, action_when_starfighter_is_seen, and action_when_starfighter_is_not_seen, update_can_see_starfighter, update_seen_by_starfighter, move_enemy.

All the enemy class inherit the ENEMY class and implement the deferred features in their own way. The ETF_MODEL class has a HASH_TABLE of ENEMY class in which I dynamically store different types of enemy. This table is only created once in the entire life span of the program.

The preemptive actions are based on starfighter's action, so according to the starfighter's action different enemies will behave differently. The preemptive action is always performed before enemy's actual action. And according to action enemy will behave differently whether it sees the starfighter or not or whether its turn has ended or not. After the preemptive action if enemy's turn has ended then that enemy cannot perform non-preemptive action for that turn.

After both preemptive and non-preemptive action enemy's vision will be update and turn ended Boolean (is_turn_ended) will be set to false.

The Information Hiding design principle is satisfied in this implementation because the user only need to know the feature to call which is not hidden and stable. While user may not be concerned with the implementation of each enemy behavior. All the implementation details are hidden and can be changed if necessary and user doesn't need to know that.

The Single Choice Principle is also satisfied, because only one HASH_TABLE of enemy is created in ETF_MODEL.

Each class has common set of attributes which are defined in the ENEMY class and each class inherit those attributes and set values according to the requirement. This is where cohesion is satisfied.

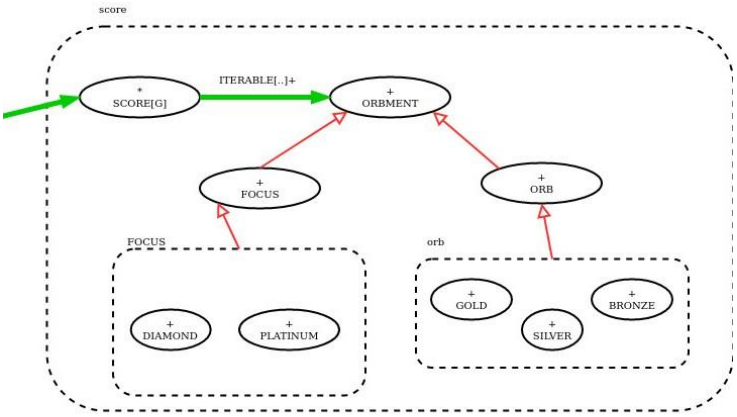
The Programming from Interface is also satisfied. There is only one ENEMY class which is deferred, and all its feature are implemented its child classes.

*The snippet of deferred features is provided below*

```
110
111 feature --Actions
112     preemptive_action(sf_act : INTEGER)
113         deferred
114         end
115     action_when_starfighter_is_not_seen
116         deferred
117         end
118     action_when_starfighter_is_seen
119         deferred
120         end
121     move_enemy(steps:INTEGER)
122         deferred
123         end
124 end
125
```

## 2.Section: Scoring of Starfighter

The score contains two different kinds of elements ORB and FOCUS. ORB has a fixed value based on its type. Whereas as FOCUS is a linear array of fixed size which can contain ORB or FOCUS, thus making it recursive in nature.




As shown in the above diagram Score is a deferred class which has client supplier relationship with ORBMENT class. There is SCORE object in ETF_MODEL which satisfies the single choice principle.

This approach is a Composite pattern approach

The ORB has three different types: GOLD, SILVER and BRONZE. As seen in the diagram all three are child classes of ORB. Each of the three types have different score values.

Similarly FOCUS has two different types: DIAMOND and PLATINUM. DIAMOND is array of size 4 with its first place occupied by gold orb. And platinum is an array of size 3 with its first place occupied by bronze orb.

The Starfighter has its own focus which is infinite and stores orbments. Before getting the score of a turn, in each turn we must add the orb or focus based on the type of enemy got destroyed.

In order to add we need recursion. Let say a grunt is destroyed, which drops silver orb, to store that orb we will first find the empty slot in the focus stored in starfighter. Let say the first slot is occupied by a different orb and the second place is occupied by a focus which has an empty space in it. So, we will traverse through the focus recursively and add it there.

Now to get the total score we must visit each slot and add the values. For this we must visit all focus that are also stored. The recursive feature will run till the array's size.

The design principle Information Hiding is satisfied because user is not aware how the score is calculated. He/she just need to call get_score method which will recursively get the score.

The Cohesion principle is also satisfied because all the classes are closely related.

The Programming from the Interface is also satisfied as there is one parent class ORBMENT which has two effective class FOCUS and ORB, and these classes have their child classes.

Here is the small code snippet for the adding the orb into focus

```
Algorithm add
        If sf_focus is Empty, then
                Add the orb to the array
        Else if contains focus then
                add orbment to focus if not full
        Else
                Add the orb in first empty slot
        End if
End Algorithm
```

The get score is like add instead of adding we are retrieving.