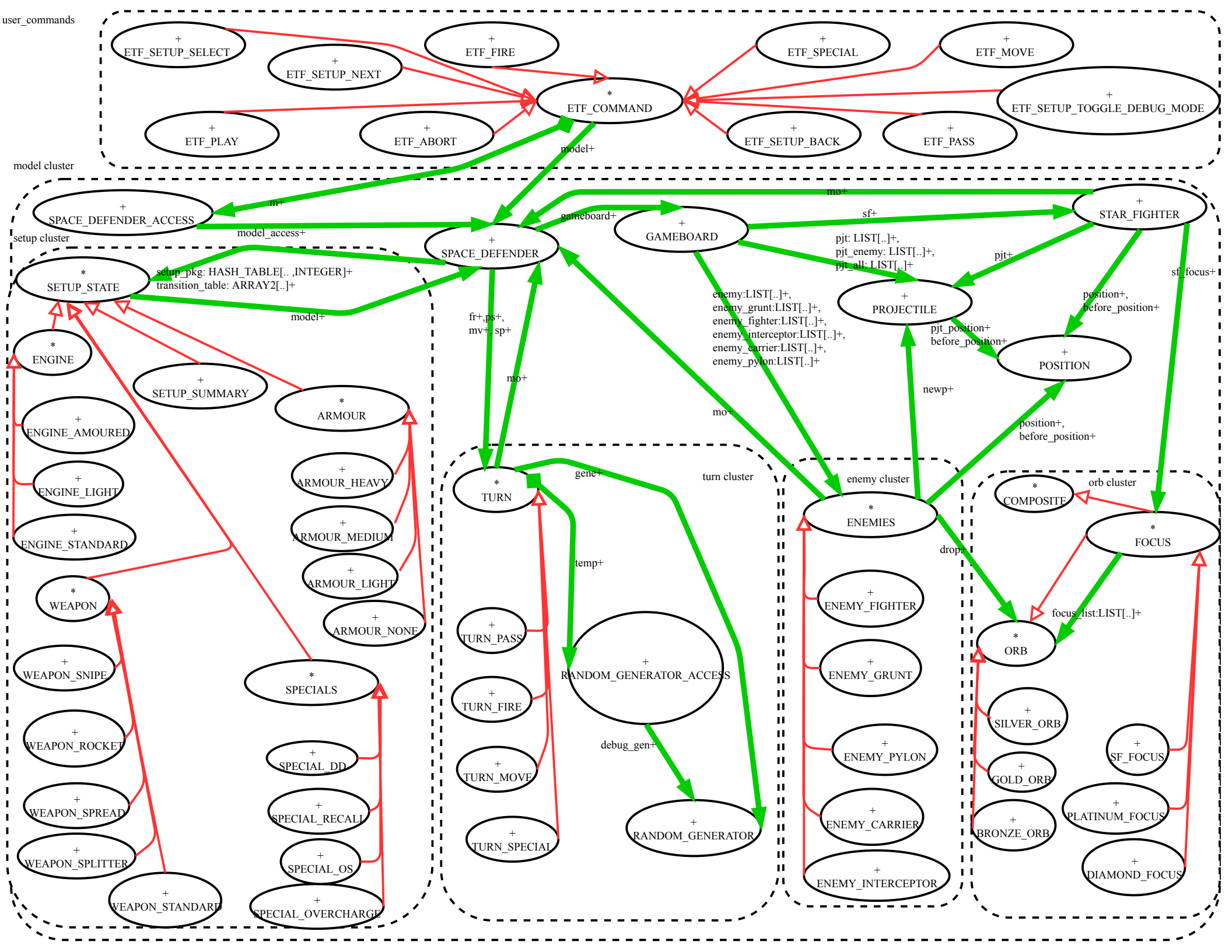
Project Report

EECS3311
2020WIN
ZIWEI WANG
WANGZW

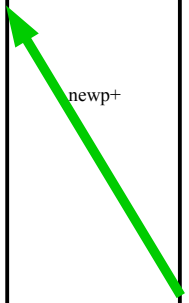

user_commands
+ ETF_SETUP_SELECT
+ ETF_SETUP_NEXT
+ ETF_PLAY
+ ETF_FIRE
+ ETF_ABORT
* ETF_COMMAND
+ ETF_SPECIAL
+ ETF_SETUP_BACK
+ ETF_MOVE
+ ETF_PASS
+ ETF_SETUP_TOGGLE_DEBUG_MODE
model+
model cluster
m+
+ SPACE_DEFENDER_ACCESS
model_access+
setup cluster
+ SPACE_DEFENDER
gameboard+
+ GAMEBOARD
mo+
sf+
+ STAR_FIGHTER
pjt: LIST[..]+,
pjt_enemy: LIST[..]+,
pjt_all: LIST[..]+
pjt+
sf_focus+
* SETUP_STATE
setup_pkg: HASH_TABLE[.. ,INTEGER]+
transition_table: ARRAY2[..]+
model+
fr+,ps+,
mv+, sp+
mo+
enemy:LIST[..]+,
enemy_grunt:LIST[..]+,
enemy_fighter:LIST[..]+,
enemy_interceptor:LIST[..]+,
enemy_carrier:LIST[..]+,
enemy_pylon:LIST[..]+
+ PROJECTILE
pjt_position+
before_position+
position+,
before_position+
+ POSITION
newp+
mo+
position+,
before_position+
* ENGINE
+ ENGINE_AMOURED
+ ENGINE_LIGHT
+ ENGINE_STANDARD
+ SETUP_SUMMARY
* ARMOUR
+ ARMOUR_HEAVY
+ ARMOUR_MEDIUM
+ ARMOUR_LIGHT
+ ARMOUR_NONE
* WEAPON
+ WEAPON_SNIPE
+ WEAPON_ROCKET
+ WEAPON_SPREAD
+ WEAPON_SPLITTER
+ WEAPON_STANDARD
* SPECIALS
+ SPECIAL_DD
+ SPECIAL_RECALL
+ SPECIAL_OS
+ SPECIAL_OVERCHARGE
turn cluster
* TURN
gene+
temp+
+ TURN_PASS
+ TURN_FIRE
+ TURN_MOVE
+ TURN_SPECIAL
+ RANDOM_GENERATOR_ACCESS
debug_gen+
+ RANDOM_GENERATOR
enemy cluster
* ENEMIES
drop+
+ ENEMY_FIGHTER
+ ENEMY_GRUNT
+ ENEMY_PYLON
+ ENEMY_CARRIER
+ ENEMY_INTERCEPTOR
orb cluster
* COMPOSITE
* FOCUS
focus_list:LIST[..]+
* ORB
+ SILVER_ORB
+ GOLD_ORB
+ BRONZE_ORB
+ SF_FOCUS
+ PLATINUM_FOCUS
+ DIAMOND_FOCUS

PROJECTILE +

feature
make
feature --attribute
pjt_mov:**INTEGER**
pjt_position:**POSITION**
before_position:**POSITION**
id:**INTEGER**
damage:**INTEGER**
collide:**BOOLEAN**
symbol:**CHARACTER**
feature -- command
pjt_forth -- move the projectile
require
pjt_is_valilid: is_valid
ensure
pjt_has_moved: pjt_position.row ~ before_position.row ∨ pjt_position.col ~ before_position+pjt_move
change_damage(da:**INTEGER**) --change the damage of projectile
require
valid_damage: da /=0
pjt_is_valilid: is_valid
ensure
damage_has_changed: damage=da

change_pjt_mov(mv:INTEGER) --change the move step of projectile
require
move_is _valid: mv /=0
pjt_is_valilid: is_valid
ensure
move_has_changed: pjt_mov=mv

set_position(pos:**POSITION**) --change the position of projectile
require
valid_position: pos.row <=gameboard.board_row ∨ pos.col <= gameboard.board_column ∨ pos.row > 0 ∨ pos.col > 0
ensure
pjt_has_moved: before_position ~ old pjt_position ∨ pjt_position ~ pos



ENEMIES *

feature -- attribute
seen_by_Starfighter :**BOOLEAN**
can_see_Starfighter:**BOOLEAN**
id:**INTEGER**
position:**POSITION**
symbol:**CHARACTER**
actual_health:**INTEGER**
move:**INTEGER**
feature -- command
execute -- execute the enemy action
require
enemy_is_valid:is_valid

check_for_collision(tmppos:**POSITION**)
require
enemy_is_valid: is_valid
position_is_valid: tmppos.row <=gameboard.board_row ∨ tmppos.col <= gameboard.board_column ∨ tmppos.row > 0 ∨ tmppos.col > 0

fire_new -- fire a new projectile
require
enemy_is_valid:is_valid
ensure
new_pjt_added_to_pjt_all : old gameboard.pjt_all.count +1 = gameboard.pjt_all.count
new_pjt_added_to_pjt_enemy: old gameboard.pjt_enemy.count +1 = gameboard.pjt_enemy.count

check_vision
require
enemy_is_valid:is_valid

feature --deferred commands
action
require
enemy_is_valid:is_valid

execute_prp_action
require
enemy_is_valid:is_valid

Section 1 : Enemy Actions

Preemptive action:
Enemy action happens in feature phase_e of deferred class TURN. Because different command will trigger the preemptive action of different enemy, so in the deferred class ENEMIES, there is a feature “trigger_prp_action” that will set the variable “prp_action” to True. When the enemy been generated each time, add it into its own type of enemy list and also add it to the list that contains all the enemy that has been generated so far(for example if a carrier is generated, add it into the list “enemy_carrier” in the GAMEBOARD, then add it into list “enemy” as well). Then if the current command will trigger the preemptive action of a certain type of enemy, across the list that stored that type of enemy, and for each one, call “trigger_prp_action”. Then in phase_e, across the “enemy” list, if this enemy is not been destroyed, and its “prp_action” set to True, also the starfighter is not destroyed at this time, call “execute_prp_action”

1.Grunt
The grunt’s preemtive action will be triggered by pass or special. If the current command is pass, prp_pass will be called, then change the health accordingly, add the message the grunt’s action message

2.Fighter
If the current command is fire, just call prp_fire to change the armour. If pass, regen first, call “check_for_collision (tmppos:POSITION)” each step it supposed to move, then change the move step and damage of projectile and call the “fire_new” to create a new projectile. Call “check_for_collision (tmppos:POSITION)” for this new projectile ,add the message the fighter’s action message, set “turn_over” = True so that the following non-preemptive action will not happen.

3. Carrier
Depends on the current command, call the according feature. If it is pass, regen first, after moving the carrier is not destroyed, create 2 interceptor, check collision for each of them and add the message accordingly, set “turn_over” = True so that the following non-preemptive action will not happen.

4.interceptor
If the command is fire, regen first. Check the position of starfighter, check collision before moving, add the message accordingly, set “turn_over” = True so that the following normal action will not happen.

Non-preemptive action

After execute the preemptive action for the certain enemy, its “execute” feature will be called to perform the non-preemptive action. Also when update the enemy vision in phase_d, the “check_vision” feature been called and update the move step of enemy, the move step of the projectile it fires(if the enemy can fire projectile) and the damage of the projectile, depends on whether the starfighter can be seen by this enemy.

```
feature--check vision
    check_vision
        do
            check_can_see_Starfighter
            check_seen_by_Starfighter
        end
    check_can_see_Starfighter
        do
            if position.distance_between(gameboard.sf.position )  <= vision then
                can_see_Starfighter:= true
                move:=move_after
                pjt_damage:=pjt_damage_after
                pjt_mv:=pjt_mv_after
            else
                can_see_Starfighter:=false
                move:=move_before
                pjt_damage:=pjt_damage_before
                pjt_mv:=pjt_mv_before
            end
        end
    check_seen_by_Starfighter
        do
            seen_by_Starfighter:=position.distance_between(gameboard.sf.position ) <=
            gameboard.sf.vision
        end
```

1.Grunt

Because the turn of Grunt does not end for both preemptive actions, so first it will regen. Since the move step, move step of the projectile it fires, the damage of the projectile it fires, have already been update in phase_d, so in the “action” feature, it will just call “check_for_collision (tmppos:POSITION)” for each step it supposed to move, if grunt collide with friendly projectile at “tmppos”, remove the friendly projectile, the damage it take will be max(0, damage of friendly projectile-armour), update the “health” of grunt by calling “change_actual_health()”, add the message the grunt’s action message. If the “health” is below 0, “collide” will be set to true, then “toggle_flag_has_moved” so that grunt will not move anymore because it is already been destroyed, add the grunt’s orb to starfighter’s focus, add the message the grunt’s action message

```
--if collide with friendly pjt
across gameboard.pjt is p
loop
    if p.valid and not p.collide_by_others then
        if p.pjt_position ~ tmppos then
            p.toogle_collide_by_others
            t1:=p.damage-armour
            if t1<0 then t1:= 0 end
            change_actual_health(-t1)
            current.set_collide_with_others_msg ("friendly projectile",
            p.id, p.pjt_position, "taking "+t1.out+" damage")
            if collide then
                toggle_flag_has_moved
                set_position (tmppos)
                set_destroy_others_msg("      The "+name+" at location ["
                +tmppos.correct_row.out+","+tmppos.col.out+"] has been destroyed.%N")
                gameboard.game_board[tmppos.row,tmppos.col]:='_'

                gameboard.sf.sf_focus.add (drop)
            end
        end
    end

end
```

Then check whether grunt collide with enemy projectile at “tmppos”, if so, remove the enemy projectile, add the damage of the enemy projectile to the health of grunt, add the message to the grunt’s action message

```
--if collide with enemy pjt
across gameboard.pjt_enemy is p
loop
    if p.valid and not p.collide_by_others then
        if p.pjt_position ~ tmppos then
            if p.flag_has_moved then
                p.toogle_collide_by_others
            else
                p.toogle_collide
            end
            change_actual_health(p.damage)
            current.set_collide_with_others_msg ("enemy projectile",
            p.id, p.pjt_position, "healing "+p.damage.out+" damage")

        end
    end
end
```

Then check whether grunt collide with another enemy at “tmppos”, if so, set the position of grunt one step back, then “toggle_flag_has_moved” then so that grunt will not move anymore

```
-- if collide with another enemy
across gameboard.enemy is e
loop
    if e.valid then
        if e.position ~ tmppos and e.id /= id then
            toggle_flag_has_moved
            if flag_is_interceptor then interceptor_collide_with_enemy:=true
                flag_is_interceptor :=false
            else
                set_position (e.position.to_right(1).deep_twin)
            end
        end
    end
end
```

Then check whether grunt collide with starfighter at “tmppos”, if so, set “collide ” = True, add the grunt’s orb to starfighter’s focus, update the health of starfighter, then “toggle_flag_has_moved” then so that grunt will not move anymore because it already been destroyed, add the according message to grunt’s action message. Then check whether the starfighter is destroyed after this collision, if so, add the according message to grunt’s action message.

```
--if collide with sf
if tmppos~ gameboard.sf.position then
    toogle_collide_by_others
    set_position (tmppos.deep_twin)
    gameboard.sf.change_actual_health (-actual_health)
    set_collide_with_others_msg("Starfighter",0,tmppos.deep_twin,"trading "
        +actual_health.out+" damage")
    toggle_flag_has_moved
    set_destroy_others_msg("      The "+name+" at location ["
        +tmppos.correct_row.out+","+tmppos.col.out+"] has been destroyed.%N")
    gameboard.sf.sf_focus.add (drop)
    if gameboard.sf.destroyed then
        set_destroy_others_msg("      The Starfighter at location ["
        +tmppos.correct_row.out+","+tmppos.col.out +"] has been destroyed.%N")
    end
end
```

After checking all the steps for grunt and if the “flag_is_moved” is not triggered, means it is safe to go, otherwise the position of grunt has already been set. then fire a new projectile if it is not destroyed.

```
action
    local
        tmppos:POSITION
        tmp:INTEGER
    do
            regen
        if not gameboard.sf.destroyed or collide then
            from tmp:=position.col
            until tmp= position.col-move-1 or collide or flag_has_moved
            loop
                create tmppos.make (position.row,tmp )
                check_for_collision(tmppos)
                tmp:=tmp-1
            end
        end

        if valid  and not flag_has_moved then
            backward
            current.toggle_flag_has_moved
        end
        generate_action_msg
        if flag_has_moved and valid and not gameboard.sf.destroyed then
            fire_new
        end
    end
```

2.Fighter

First check whether the turn for fighter is ended, if not, regen. Then same as grunt, before making any move, call “check_for_collision (tmppos:POSITION)” each step it supposed to move, if collision happened, add message to action message. If the fighter is still valid after this, call “fire_new” to fire a new projectile.

3.Carrier

First check whether the turn for fighter is ended, if not, regen. If Starfighter is not seen, then same as before, check for collision before moving. If Starfighter is seen, check for collision before moving, and after this if carrier is still valid, create an interceptor at its left, store it into the list “enemy” and list “enemy_interceptor”, check collision for this new interceptor, add message to carrier’s action message, update vision for this new interceptor.

4.Interceptor

First check whether the turn for fighter is ended, if not, regen. If Starfighter is not seen, then same as before, check for collision before moving. If Starfighter is seen, check for collision before moving.

5.Pylon

First check whether the turn for fighter is ended, if not, regen. If Starfighter is not seen, check for collision before moving. If after moving it is still valid, across the enemy list, calculate the distance between it and this pylon, if the distance is smaller than the pylon's vision, increase its health. If Starfighter is seen, check for collision before moving , if the fighter is still valid after this, fire a new projectile by calling "fire_new".

If star fighter is destroyed during any of the place described above, the following movement will stop.

Information Hiding

This principle is satisfied because in the ENEMIES class, some features are likely to change, like the make feature for each descendant, so it is hidden from the client. All the feature used to generate action message is also hidden from the clients because client does not need to know the detail of how enemy moves. "fire_new" is also hidden from the client because only the descendant of ENEMIES will use this feature. Some features are unlikely to change, like "execute" , "execute" is defined in the class ENEMIES and it will not be changed by its descendants, so it's safe for client to just call this feature, therefore it is not hidden. Feature like "check_vision","toggle_collide" is not hidden because client like TURN need to update vision for each enemy, and when projectile or starfighter is collide with this enemy, client should be able to set this enemy to collide, so these are not hidden.

Single Choice Principle

This principle is satisfied because in the phase_e of TURN, it will across the list of enemy and call e.execute_prp_action and e.execute to do the preemptive action and non-preemptive action. Beacause different enemy have different preemptive action and non-preemptive action, so each decendant of ENEMIES will inherit the deferred feature "execute_prp_action" and "action" and implement it in its own class. So if any of the enemy action need to be modified, just modify the according feature in each descendant class like ENEMY_CARRIER, ENEMY_FIGHTER, ENEMY_GRUNT, ENEMY_INTERCEPTOR, ENEMY_PYLON, the rest can remain untouched. And if a new type of enemy been introduced, just create a new class that inherit from the ENEMIES, implement the feature "execute_prp_action" and "action" in its own class, other class will remain untouched.

Cohesion

Class ENEMIES defined the feature "execute", "execute_prp_action"," check_for_collision"," generate_action_msg"," check_vision"," set_position", etc , these feature will be inherit by the descendant class verbatim so that the descendant class will not have to redefine then again. in the descendant class ENEMY_CARRIER, ENEMY_FIGHTER, ENEMY_GRUNT, ENEMY_INTERCEPTOR, ENEMY_PYLON, each redefine the feature "action", that can let

each enemy to perform action accordingly, The feature “fire_new” is defined in ENEMIES, descendant class such as ENEMY_GRUNT, and ENEMY_FIGHTER can call it when they need to fire a new projectile. But class like ENEMY_CARRIER, in certain situation it will spawn an enemy instead of projectile, so “fire_new” is redefined in that class. Therefore each class only contains the feature that related to themselves.

Programming from the Interface, Not from the Implementation

When creating the enemy list, all the static type of the attributes like enemy, enemy_grunt, enemy_fighter, enemy_interceptor, enemy_carrier, enemy_pylon is LIST[ENEMIES], in order to instantiate this attributes, use the effective class of ENEMIES.

```
--enemy
        enemy:LIST[ENEMIES]
        enemy_grunt:LIST[ENEMIES]
        enemy_fighter:LIST[ENEMIES]
        enemy_interceptor:LIST[ENEMIES]
        enemy_carrier:LIST[ENEMIES]
        enemy_pylon:LIST[ENEMIES]
```

```
--enemy
            create {ARRAYED_LIST[ENEMIES]}enemy.make (1)
            create {ARRAYED_LIST[ENEMIES]}enemy_grunt.make (1)
            create {ARRAYED_LIST[ENEMIES]}enemy_fighter.make(1)
            create {ARRAYED_LIST[ENEMIES]}enemy_interceptor.make (1)
            create {ARRAYED_LIST[ENEMIES]}enemy_carrier.make (1)
            create {ARRAYED_LIST[ENEMIES]}enemy_pylon.make (1)
```

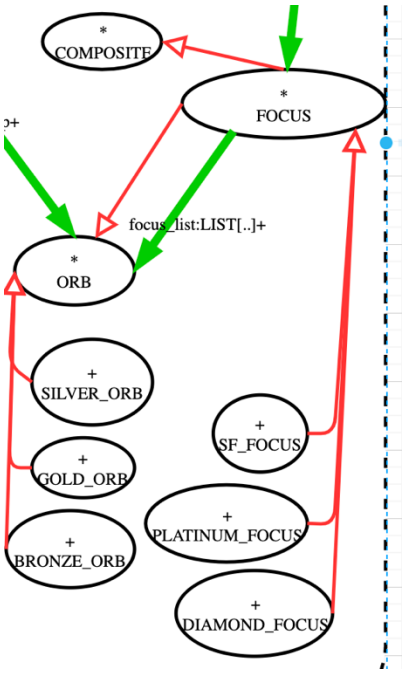
Also ENEMIES class contains deffered feature “action”,” execute_prp_action”, etc., so the client only need to know these feature will trigger the action of an enemy, how each enemy will move depend on the effective feature in their own class.

Section 2 : Scoring of Starfighter

Use the composite pattern learned in the class. ORB is a deferred class that has attribute score, “flag_full”, and slot, feature “get_score” can return the score. Class COMPOSITE has attributes “focus_list”. The effective class FOCUS inherits the ORB and COMPOSITE, ORB has another 3 descendants GOLD_ORB, BRONZE_ORB and SILVER_ORB, FOCUS has 3 descendants SF_FOCUS, DIAMOND_FOCUS and PLATINUM_FOCUS. ORB has slot 1, DIAMOND_FOCUS has slot 4, PLATINUM_FOCUS has slot 3.

If the enemy drop a new orb/focus, starfighter will call “add()” to across the current “focus_list” to check whether any element in it does not show “flag_full”, because for the GOLD_ORB, BRONZE_ORB and SILVER_ORB, there “flag_full”, are set to True by default, so this element that does not show “flag_full”, must of dynamic type of the descendent of FOCUS, therefore call add() on this element. If all the element in the current “focus_list” has “flag_full”, means this new orb/focus has to be add to the next slot of current “focus_list”. After adding this new orb/focus, call “check_flag_full” to check whether the current “focus_list” is full. Therefore, the new orb/focus can be add to a place in the descendant “focus_list” recursively.

When calculate the score, let starfighter call “get_score”, then it will go through the starfighter’s “focus_list”, feature “get_score” is defined in class ORB and redefined in class FOCUS, in class ORB, “get_score” simply return the “score”, in class FOCUS, “get_score” will go through the current “focus_list” and add the score of each element together. So the score of the descendant focus within a focus will be calculated recursively.



```
check_flag_full
    local
        f:BOOLEAN
    do
        f:=true
        across focus_list is li
        loop
            f:= f and li.flag_full
        end

        flag_full := f and focus_list.count = slot

    end
add(sc:ORB)
    local
        flag:BOOLEAN
    do
        flag:=false

            across focus_list is li
            loop
                if not li.flag_full then
                    if attached {FOCUS}li as l then
                        l.add (sc)

                        flag:=true
                    end
                end
            end
            if not flag then
                focus_list.extend (sc)
            end

        check_flag_full
    end
```

```
get_score:INTEGER
    do

        across focus_list is li
        loop
            result:=result + li.get_score
        end
        if focus_list.count = slot then
            if slot/=1 then
                result:=result * (slot-1)
            end

        end
    end
```

Information Hiding

The static type of the attributes focus_list in class COMPOSITE is likely to change, so it is hidden from then client, because client does not need to know the type of the focus_list , if it change later, clients should not been affect, therefore this is hidden from the client. Feature like “check_flag_full” and “ get_score” in class FOCUS is stable and unlikely to change, because the way to determine whether the current “focus_list” is full is unlikely to change, so does “get_score”, client can call theses feature without knowing how this can be done, therefore they are not hidden.

Single Choice Principle

GOLD_ORB, BRONZE_ORB and SILVER_ORB inherit from class ORB, class SF_FOCUS, DIAMOND_FOCUS and PLATINUM_FOCUS inherit from FOCUS, how the descendant class behave is defined in its own class, so if we need to change the feature of any effective class, like change the slot number for PLATINUM_FOCUS, we only need to modify this class, the rest can remain unchanged.

If a new type of focus or orb needed, we can just create a new descendant class of FOCUS or ORB that inherit all the feature from them, it can then be declared as dynamic type, the rest of the structure can remain untouched.

Cohesion

The GOLD_ORB, BRONZE_ORB ,SILVER_ORB , SF_FOCUS, DIAMOND_FOCUS , PLATINUM_FOCUS inherit the attribute score, “flag_full”, slot verbatim. In GOLD_ORB, BRONZE_ORB ,SILVER_ORB just set the number of slot and the number of score, the “flag_full” is true because orb cannot add orb or focus. In DIAMOND_FOCUS , PLATINUM_FOCUS they also need to instantiate the “focus_list” and the type of orb in their first slot. Each class only contain the feature that strictly related to them, so the cohesion principle is satisfied.

Programming from the Interface, Not from the Implementation

The static type of “focus_list” in the class COMPOSITE is LIST, then when instantiate it, use LINKED_LIST, an effective class that is the descendant of LIST.

```
deferred class
    COMPOSITE[T]

feature {NONE}
    focus_list:LIST[T]
```

```
class
    PLATINUM_FOCUS
inherit
    FOCUS
create
    make_focus
feature
    make_focus
        local
            bronze_orb:ORB
        do
            create  {BRONZE_ORB}bronze_orb.make
            flag_full:=false
            create {LINKED_LIST[ORB]}focus_list.make
            focus_list.force (bronze_orb)

            slot:=3

        end
```

orb and focus all have feature “get_score”, when the client calls this feature, it will call the according “get_score” in each class, the client is able to get the total of scorse without knowing the detail.

The attribute “drop” in deferred class ENEMIES is defined of type ORB, in each descendant class of ENEMIES, “drop” is instantiated as the descendant of ORB, such as DIAMOND_FOCUS. When the detail in class ORB has changed, the client will remain unaffected.