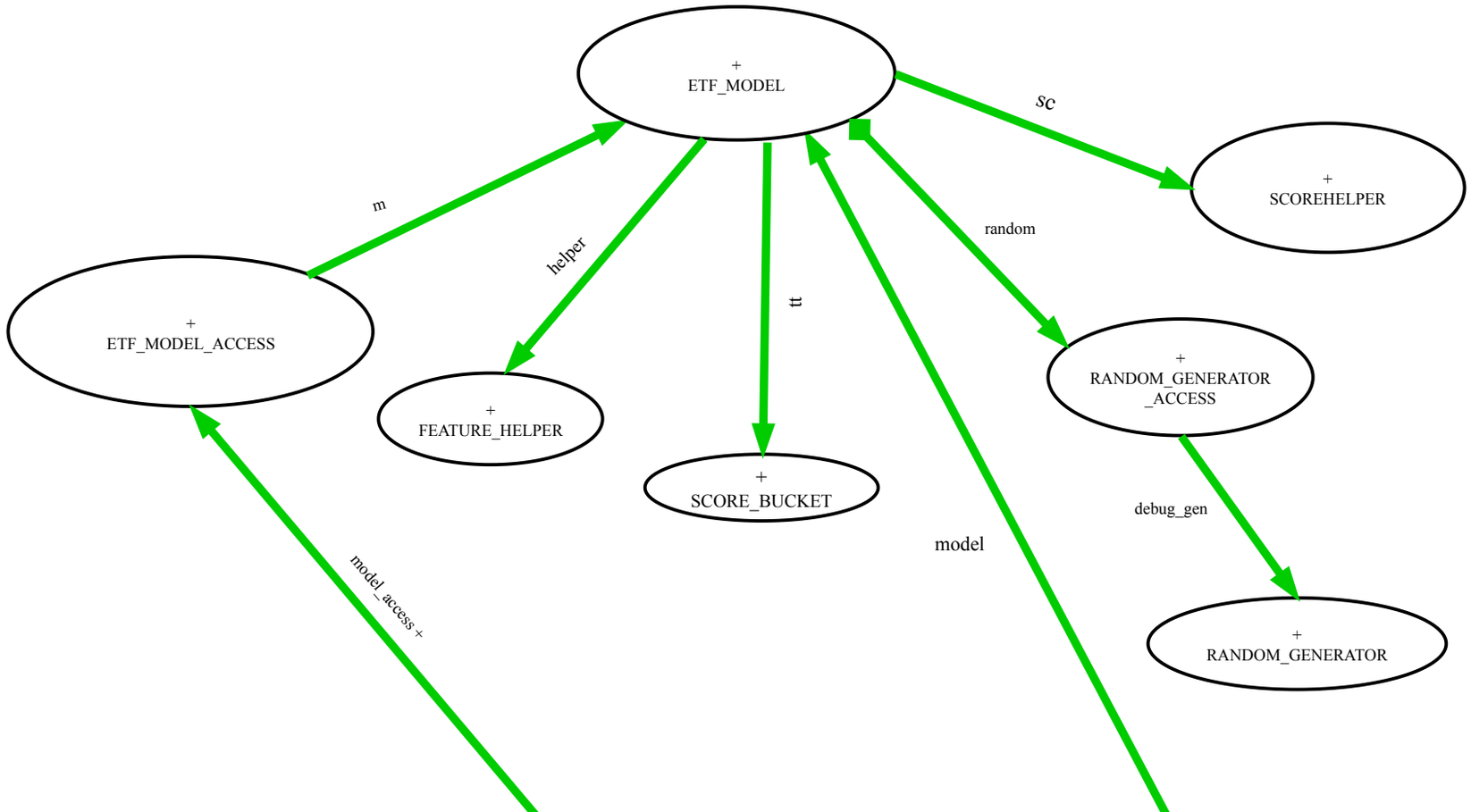
COURSE – EECS 3311 PROJECT REPORT

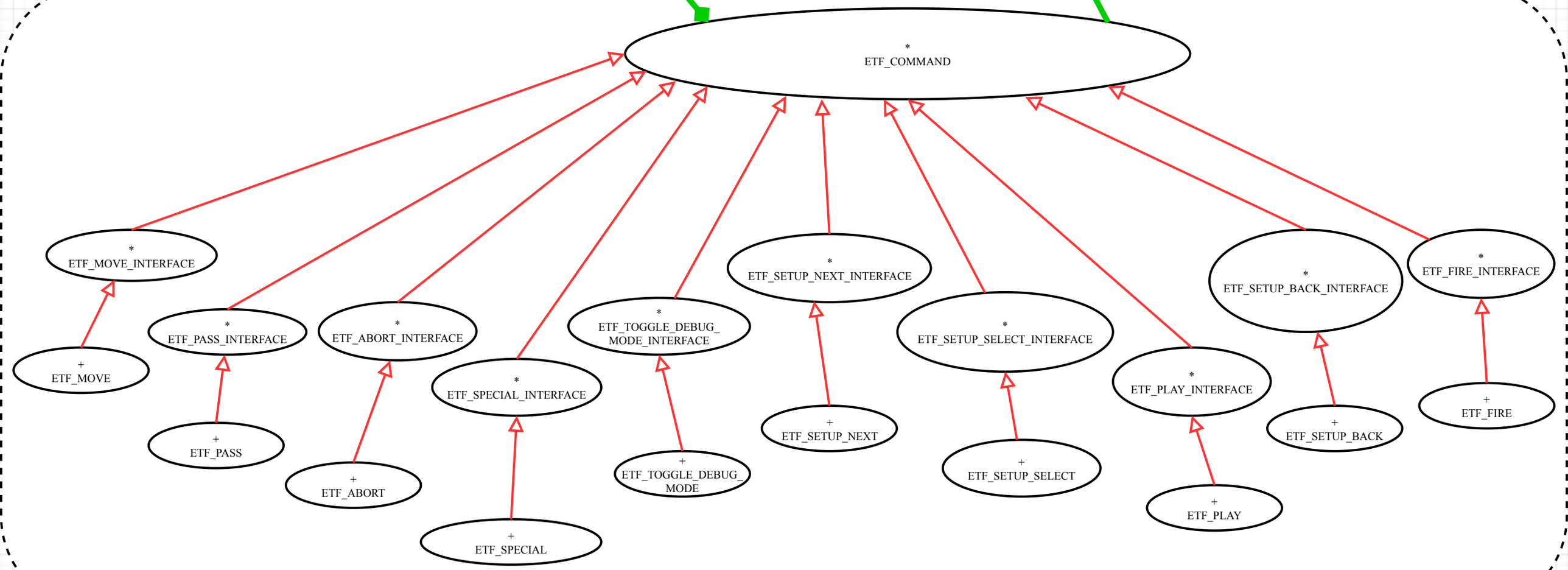
SEMESTER – FALL 2020

PRISM LOGIN -- samayak

MODEL



USER_INTERFACE



MODEL

ETF_MODEL +

```
note
   description: "A default business model."
   author: "Jackie Wang"
   date: "$Date$"
   revision: "$Revision$"

class interface
   ETF_MODEL

create {ETF_MODEL_ACCESS}
   make

feature -- model operations

   abort

   canseenby (start_row: INTEGER_32; start_col: INTEGER_32; end_row: INTEGER_32; end_col: INTEGER_32; vision: INTEGER_32): BOOLEAN
         -- from current location of enemy to fighter of both vision

   change_armour_name (sel2: STRING_8)

   change_engine_name (sel3: STRING_8)

   change_tolprojdamage (i19: INTEGER_32)

   change_tolregene (i14: INTEGER_32)

   change_tolregenh (i13: INTEGER_32)

   change_tolvision (i16: INTEGER_32)

   change_weapon_name (sel1: STRING_8)

   collide_pass: BOOLEAN

   converttruefalse (b1: BOOLEAN; b2: BOOLEAN): TUPLE [csf: STRING_8; sbf: STRING_8]

   debug_mode

   default_update
         -- Perform update to the model state.

   enemy_spawn

   enemyatspawnproje (row4: INTEGER_32; col4: INTEGER_32): BOOLEAN

   enemyatspwanprojf (rowfinal2: INTEGER_32; colfinal2: INTEGER_32; number: INTEGER_32): BOOLEAN
         -- remove projf if enemy collided

   enemycollidefighter (row3: INTEGER_32; colstart3: INTEGER_32; colend3: INTEGER_32; number: INTEGER_32): BOOLEAN
         -- enemy move and collide fighter

   enemycollideprojf (row3: INTEGER_32; colstart3: INTEGER_32; colend3: INTEGER_32; damage: INTEGER_32; number: INTEGER_32): BOOLEAN

   enemyfightervisionupdate

   enemyput (efirstrow: INTEGER_32; espawncol: INTEGER_32; sign: STRING_8; id: INTEGER_32)
         -- first 2 spawn are at 1,c

   entire_grid: STRING_8
         -- create entire grid

   fighter_vision (fighterrow: INTEGER_32; fightercol: INTEGER_32)
         -- update ? not S

   fightercollideenemy (mrow2: INTEGER_32; mcol2: INTEGER_32)

   fightercollideproj (mrow: INTEGER_32; mcol: INTEGER_32)

   fire

   hasenemyin (enemystartrow: INTEGER_32; enemystartcol: INTEGER_32; enemyendrow: INTEGER_32; enemyendcol: INTEGER_32): BOOLEAN

   incvisionhealth (row6: INTEGER_32; col6: INTEGER_32; number: INTEGER_32)

   ingamesetup

   move (mrow: INTEGER_32; mcol: INTEGER_32; ls_flag: STRING_8)

   num_steps (start_row: INTEGER_32; start_col: INTEGER_32; end_row: INTEGER_32; end_col: INTEGER_32): INTEGER_32

   pass

   play_setup (row: INTEGER_32; column: INTEGER_32; g_threshold: INTEGER_32; f_threshold: INTEGER_32; c_threshold: INTEGER_32; i_threshold: INTEGER_32; p_threshold:
INTEGER_32)
         -- only for initial setup / intialisation first print in setup next

   premactions (prem: STRING_8)

   projatenemyspawn (row5: INTEGER_32; col5: INTEGER_32; number: INTEGER_32)

   projatspawnprojf (rowfinal: INTEGER_32; colfinal: INTEGER_32; number: INTEGER_32)

   projeatspawnproj (row3: INTEGER_32; col3: INTEGER_32; number: INTEGER_32): BOOLEAN
         --check if there proj at spawn

   projecollideproje (rowstart5: INTEGER_32; colstart5: INTEGER_32; colend5: INTEGER_32; number: INTEGER_32): BOOLEAN

   projecollideprojf (row3: INTEGER_32; colstart3: INTEGER_32; colend3: INTEGER_32; number: INTEGER_32): STRING_8

   projenemycollide (row1: INTEGER_32; colstart: INTEGER_32; colend: INTEGER_32): BOOLEAN
         -- check if any in between for horizontal moving projectiles

   projfcollideenemy (mrow2: INTEGER_32; mcol2: INTEGER_32; number: INTEGER_32): STRING_8

   projfcollideproje (row3: INTEGER_32; colstart3: INTEGER_32; colend3: INTEGER_32; id: INTEGER_32; damage: INTEGER_32; number: INTEGER_32): STRING_8

   projfcollideprojetemp (row3: INTEGER_32; colstart3: INTEGER_32; colend3: INTEGER_32; id: INTEGER_32; damage: INTEGER_32; number: INTEGER_32): STRING_8

   projfightercollide (row2: INTEGER_32; colstart2: INTEGER_32; colend2: INTEGER_32): BOOLEAN

   projfoutofboard (prow: INTEGER_32; pcol: INTEGER_32; inc_by: INTEGER_32; number: INTEGER_32): BOOLEAN

   remainenergyfunc (call: STRING_8): INTEGER_3
         -- check how much health is remaining
   special

   update_allproj (premup: STRING_8)
         -- move projectile

   update_fighterproj (premup: STRING_8)
         -- projloc.put not movlov.inc := 2*movlov.inc

   updateenemy (prem: STRING_8)

   verticlecollideprojf (rowstart: INTEGER_32; ls_col: INTEGER_32; rowend: INTEGER_32; ine: INTEGER_32)

feature -- queries

   out: STRING_8
         -- New string containing terse printable representation
         -- of current object

end -- class ETF_MODEL
```

tt

SCORE_BUCKET +

```
class interface
   FEATUREHELPER

create
   make

feature

   container_size: INTEGER_32

   insert (f: SCORE_BUCKET)

   isfocus: BOOLEAN

   make (name: STRING_8)

   ornaments: LINKED_LIST [SCORE_BUCKET]

   remain_capacity: INTEGER_32

   times: INTEGER_32

   tolpoints: INTEGER_32

   val: INTEGER_32

end -- class SCORE_BUCKET
```

Q- Working of phase 5

For the enemy's data, I am maintaining a linked list of the tuple which consists of data such as its current health, total health and its location for each enemy. For applying the regeneration to the enemies, I iterate through by enemy linked list to regenerate their health.

For the preemptive actions I iterate through my linked list to check the type of enemy I am having in a particular iteration and check for the command such (as pass, fire etc.) that invoked my feature for preemptive actions and based on that analysis for a particular enemy I do its preemptive action if an appropriate command to trigger the preemptive action is invoked by the player.

For the normal actions of the enemy, Once the preemptive actions of the enemy are finished, I once again iterate through my enemy linked list to first check if they can see the enemy or not and based on that I decide their movement actions, whether they will fire or spawn new enemies. While I iterate through my linked list I checked for collision for the enemies with different entities after moving the enemy as per the seen and not seen condition. if there is no collision or the enemy is still alive then only, I fire or spawn enemies

For the satisfaction of --

1) Information hiding - When I implemented the enemy action the clients have to use getters and setters to get/set the value of the attributes in my enemy collision class and they do have the access to directly modify the data structures , other attributes in the class. All my enemy action attributes are being used by only the supplier class which handles the enemy action not by any other client class.

Example feature -

```
change_tolhealth(i11 : INTEGER)
    do
        tolhealth := i11
    end
change_tolenergy(i12 : INTEGER)
    do
        tolenergy := i12
    end
```

2) single choice principle - For the implementation of the regeneration and preemptive action I have no repetition of code and when to check for the collision of each enemy with other entities I have a shared feature that checks the collision in enemy spawning, preemptive action as well as in normal actions of the individual enemies.

Code Example feature in my code --

enemycollideprojf(row3:INTEGER;colstart3:INTEGER;colend3:INTEGER;damage:INTEGER;number:INTEGER) : BOOLEAN

3) cohesion - For the implementation of the enemy action movements I satisfy cohesion by keeping the code for the collision in a separate class called helper feature that takes care of all the collision so that it's easier to debug and used dedicated features in the class to that particularly handle 1 action at a time instead of keeping everything in the same feature.

Code Example -- Feature projfcollideenemy(mrow2:INTEGER;mcol2:INTEGER;number:INTEGER) : STRING

is specifically for collision of fighter projectiles

4) programing from the interface - To satisfy the programming from the interface I used the interfaces to declare the static types of my data structures such as for enemy data or projectiles locations storing, I am using List instead of linked list directly while declaring the data structures static type.

Code Example -- To store my all the projectiiles I have following declaration :

projloc : LIST[TUPLE[prow:INTEGER;pcol:INTEGER;id:INTEGER;sign:STRING;incby:INTEGER]]

Q - Working of Scoring in-game

I am using a composite design pattern for the implementaion of score

To check for increasing the score I maintain a linked list of the tuple in which I have a parameter called dead which becomes true only when an enemy dies by interacting with other entities such as after colliding with starfighter or after colliding with the starfighter projectile.

After the enemy action is finished for each enemy I check if it's dead or not and if it's dead I further check for the type of the enemy to add the score into the linked list tree structure. If the enemy died is carrier or pylon while I iterate through my linked list of enemies then I add a new focus into the linked tree structure and add new orbs into this new focus until its full

For finding the sum of the score I recursively check If the focus count is reached then do a multiplier as per the focus multiplier.

For the satisfaction of --

1) Information hiding - When implementing the scoring composite pattern all the attributes of the class are only accesible by main model class and testing class only

and they are hidden from the clients so that they can not make any changes to the code or to the data structure directly.

Code example -

feature {ETF_MODEL,SCORE_BUCKET,TESTSCORE}

val : INTEGER

ornaments: LIST[SCORE_BUCKET]

isfocus : BOOLEAN

container_size,times: INTEGER

2) single choice principle - To satisfy the single choice principle I am not calculating the score again and again when an enemy dies instead I am storing a variable in my tuple to indicate if an enemy is dead or alive then once the enemy action phase 5 is finished, I check for the type of enemy and store the relevant focus or score in my linked list and then combine the score later using the feature called tolpoints in my SCORE_BUCKET class.

Feature example -- tolpoints in my SCORE_BUCKET class

3) cohesion - To satisfy the cohesion principle for scoring I am using a separate class for calculating the score of the enemies who died by interacting with other entities in the game in which I am interacting though my enemy data to find the score/ focus to add for different enemies

Code example - class

SCORE_BUCKET

feature

insert

4) programing from the interface - The data structure I used for enemy data storage is implemented using their interfaces in their static types declaration instead of using the child classes directly in their static types.

Code Example -

ornaments: LIST[SCORE_BUCKET]