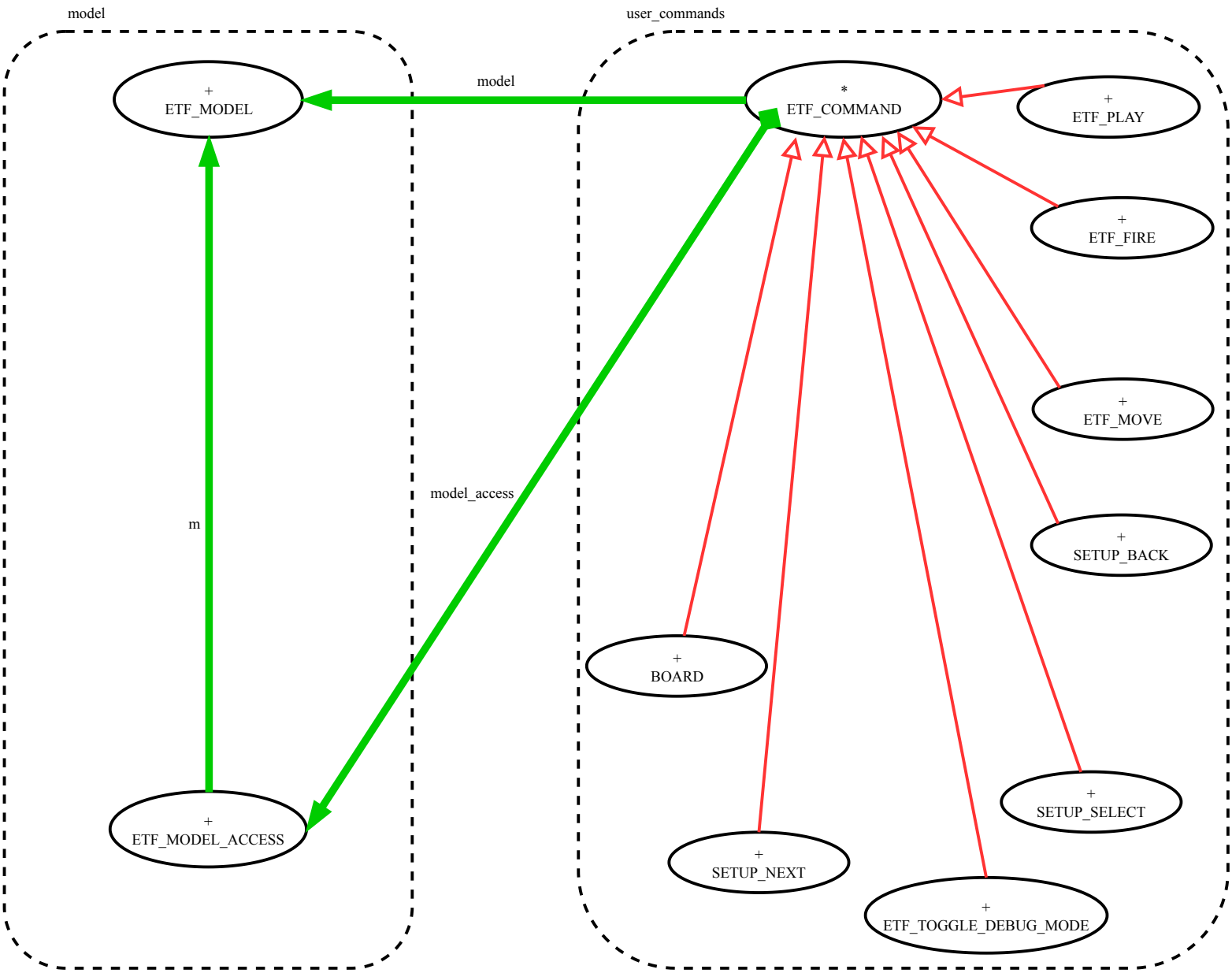
EECS 3311 - Project Bon Diagrams

Varuhn Ruthirakuhan - 215634140

Design diagram for relationship between relevant classes in space_defender_2



EECS 3311 - Project Bon Diagrams

Varuhn Ruthirakuhan - 215634140

Detailed view of toggle_debug_mode and fire class with inheritance relation

ETF_MODEL+

inherit
ANY
feature { NONE } -- Initialization
-- Initialization of integer and string values
feature -- model attributes
-- private features
feature -- model operations
abort++
-- ends the game immediately
fire++
-- fires a projectile from 1 spot in front of Starfighters location
-- weapon_type decides how projectiles are shot and how much damage is dealt
move++(row: INTEGER_32 ; column : INTEGER_32)
-- moves the ship to the designated spot by (row, column)
pass++
-- decide not to move anything during the turn
-- Starfighter energy regenrates
-- enemy health regenerates
play++ (row: INTEGER ; column: INTEGER ; g_threshold: INTEGER ; f_threshold: INTEGER ; c_threshold: INTEGER ; i_threshold: INTEGER ; p_threshold: INTEGER)
-- starts the game with designated grid size of length column and width of size row.
toggle_debug_more++
-- removes all fog from the board
-- additional messages such as Projectile movements are displayed
setup_next(state: INTEGER)++
-- displays next stage (weapon, armour, engine, or power setup)
-- state decides how far forward to go in setup stage or to start game
-- ensure (1 <= state<= 5)
setup_select(state: INTEGER)++
-- while still in setup stage, allows user to decide which attributes they would like from each setup
-- ensure (1 <= state<= 5)
setup_back(state: INTEGER)++
-- if the user would like to go back and change initialized setup
-- ensure (1 <= state<= states modified)

feature -- out
-- displays attributes of the game to the interface for the user to observe and modify

toggle_debug_mode+

feature {NONE} -- Initialization
game_started : INTEGER -- if game is running (play command called set to 1)
debug_value : INTEGER -- sets debug mode to either on or off
game_ended : INTEGER -- if game has been aborted or lost, value will be set to 1
debug_output : STRING -- display starting message depending on game state

feature -- queries
if (game_started == 1) && (game_ended == 0) && (debug_value == 0) then set debug_value to 1
--turns debug_mode on

elseif (game_started == 1) && (game_ended == 0) && (debug_value == 1) then set debug_value to 0
--turns debug_mode off

elseif (game_ended == 1) then set debug_output message to game has ended

model

*
ETF_COMMAND

\+
ETF_FIRE

\+
ETF_TOGGLE_DEBUG_MODE

fire+

feature {NONE} -- Initialization
game_started : INTEGER -- if game is running (play command called set to 1)
debug_value : INTEGER -- gotten from toggle_debug_mode
ship_x : INTEGER -- ships location on the x-axis
ship_y : INTEGER -- ships location on the y-axis
board : ARRAY2[character] -- board used for game

feature -- queries
fire
--nested across loops used to check for existing projectiles to move first on board.

board.put('*') --ship_x position and ship_y position will be taken to fire projectile one spot to the right
-- location of projectile is then stored in p_x and p_y

if (game_started == 1) and (debug_value == 1)
-- display advanced messages
else
-- cover board with fog and display minimal messages

EECS 3311 – Fall 2020

Varuhn Ruthirakuhan – 215634140

Prism login: vruth11

Dec 9, 2020

1. Enemy Actions

One preempted action enemies may undergo on the board, is the moving of their projectiles that have already been shot. For example, at the start of any next turn assuming that an enemy such as the Grunt, 'G', is still on the board, two across loops will be run to check for the enemies projectiles as shown below in figure one.

```
across 1 |..| board.height is h1 loop
    across 1 |..| board.width is w1 loop

        if board[h1.item,w1.item].item ~ '<' and w1-4 <= 1 then

            board.put ('<', h1, w1-4)
            board.put ('_', h1, w1)

        end

    end

end

board.put ('<', str_x2, str_y2)
str_sx2 := str_x2+1
str_sy2 := str_y2
```

Fig 1 – double across loop used to check for existing projectiles fired by enemies

At the start of any turn such as if the fire feature were to be called, then these two nested across loops will be used. This board searches from the top left to the bottom right of the board to ensure no missiles get missed. If the board was searched in another order such as from the bottom right to the top left, some missiles may get overlapped and either forgotten about or moved forward additional spaces. Starting from the top left, if an enemy projectile gets located, the nested across loop will locate it by using the if statement where they check to ensure any element of the board matches the '<' symbol. Then the loop will check if the projectile can move four spaces to left to ensure the missile does not produce a valid_column error. If this is the case, the projectile will move forward by four spaces assuming the pre-condition before the loop that it was fired by a grunt is proven true. Then its old spot where the missile once existed will be replaced with a '_' using the board.put command. After this, the coordinates of the projectile will be stored for potential output printing if toggle_debug_mode is turned on.

What is left hidden for possible future change here is the position of newly fired projectiles from the enemy such as a Grunt. This is done because the enemy may get hit by a Starfighter projectile, or the Starfighter itself. This was also done to make it easier to store locations of the projectiles on the board. For example, if every projectile was to be stored on the board over many turns their may be too many projectiles to store. So instead the new projectile is spawned one spot to the left of the ship where that projectile will be stored. Depending on the number of turns on the board and if the Grunt or projectile

gets hit, more projectiles will spawn in front of that first one. If three turns have passed where the grunt was on the board and it has not received any action, the two existing projectiles from the previous turn will move left by four spaces and a new projectile will spawn in front of the enemy.

The process of moving these projectiles helps satisfy information hiding. Newly fired projectiles are primarily left hidden to prevent any errors on the board. Since their position is not yet stable they may or may not be shown on the board. Their positions are initially hidden to in the case where they may interact with a starfighter. For example, if the starfighter collides with one of the enemy projectiles but still has enough health points to remain in the game, then the enemy projectile gets destroyed and will no longer be displayed or moved on the board. At the end of a turn however, the positions of the moved missiles will be set if the grunt still has health points, it will remain on the board in a stable state. This location of the Grunt is not hidden because it is needed to help display where the newest enemy projectile will be fired which is one space to the left of the Grunt.

A non-preemptive action that is done by the ship is its movement. This specifically follows single-choice principle by using the board class. This board class will be called by each feature and printed directly at the end of a call. This way when a modification needs to happen to the board, it only needs to happen within the board class and not within every individual method. For instance, when a new move is made, the board must check if the grunt can move via a non-preemptive method. In the case of a Grunt, it will ensure no other enemy ship is directly in front of it to spawn using if statement. After that is checked, potential movements of starfighter projectiles and collisions are also checked for the enemy ships health. If the enemies health bar reaches zero, it will be removed from the board. If not however, the enemy ship will enter an across loop like the one below seen in figure 2.

```
across board.height |..| 1 is h1 loop
    across board.width |..| 1 is w1 loop

        if board[h1.item,w1.item].item ~ 'G' and w1-2 <= 0 then

            board.put ('G', h1, w1-2)
            board.put ('_', h1, w1)

        end

    end

end

board.put ('<', str_g, str_gy)
str_g := str_g+1
str_gy := str_gy
```

Figure 2 – movement of Grunt enemy ship

This time the board will be checked from the top right to the bottom left for the movement of the Grunt. This is done to prevent the grunt from overlapping its fired projectiles. Granted that the Grunt is still alive, it will move two spots to the left where the new position of the enemy ship will be stored to help with printing and for the placement of a potential new firing projectile '<'.

Now with regards to the enemies regenration capabilites. Enemies only move a fixed distance each turn so they do not expend any energy. Enemies only gain or lose health. In this case I will be referring to the Grunt once again. If the special feature is called and the teleport back to spawn special ability has to be used, the Grunt will get a free turn to move forward. In addition to this, the Grunt will gain 20 extra health points. This includes even increasing the enemies health bar as seen below in figure 3.

```
grunt_hp := grunt_hp + 20
grunt_bar := grunt_hp_old+20
```

Figure 3 – Grunt health points are increased during special feature call.

Here the grunts health bar is set to a default value of one hundred where it is also stored in the grunt_hp_old variable. This is so that if the Grunt does exist and the special feature is called, the health bar will increase by twenty health points which raise the health bar from a value of 100 to a value of 120. The Grunt will also gain twenty health points from this call. Similar to this, if the pass feature gets called, the Grunts health will increase by 10 in the exact same way as seen in figure three above. This means that if the special feature were to be called and then the pass feature was to be called while the Grunt took no damage, the enemy Grunt will now have reached a total of 130 health points and the Grunts health bar would have increased to 130 health points as well.

The use of setting these Grunt health points in the features special and pass help satisfy cohesion. This is because the Grunts health is not implemented within every method to gain health points. The Grunts health points are only set to change within the relevant features. For example, in the move feature, the Grunt cannot gain any health points so two calls similar to figure three will not be used. This helps by only storing the relevant concepts in a single place to satisfy cohesion. The Grunt can only gain twenty health points if the special feature is called and can only gain ten health points if the pass feature is called so there is no need to fill up other features with these calls. If these calls were included within every feature, it would pollute with unrelated operations which could also confuse the program to printing incorrect values.

2. Scoring of starfighter

Scoring in this game will depend on health of the enemies ships. Throughout this game the position of enemy ships such as the Grunt will be checked. For instance, if the Starfighter fires a projectile towards the Grunt. Across loops will be checked to see the location of all the projectiles on the board first. Since the enemy projectiles deal less damage, the Starfighter projectiles will break through the Grunt projectiles dealing less damage each time. So, if the Starfighter fires a standard projectile and there are two enemy Grunt projectile '<' in front of it, the Starfighter Projectile will end up dealing 40 damage to the Grunt. Regardless, if the damage of the projectile is greater than the Grunts health points, the Grunt will be destroyed. At the end of every turn, a method will be called to check if the Grunts health points have reached zero or negative. This is displayed in figure four below.

```
grunt_hp := grunt_hp - projectile_damage_new

if grunt_hp <= 0 then
    score := score+2
end
```

Figure 4 – checking of Grunts health points and point distribution

The damage of the projectile is gotten after checking its movement distance with the object in front of it. As it is seen, if the Grunts health points reach a negative or zero value, two points will be added towards the Starfighters score. The two points is because Grunts drop a silver orb upon destruction. These silver orbs are collected straight away once the Grunt is destroyed giving the Starfighter two points. There are additionally different enemy ships that drop different orbs. For example, more advanced ships with more health points may drop a gold orb upon destruction granting the user with three points.

There are also many ways for the user to collect points in the game. Points can be obtained by any means to destroy the enemy ships. This could be from firing projectiles using the 'fire' feature, using the Orbital strike in the 'special' feature which deals damage to all enemies in the board, or by simply moving the ship using the 'move' feature and colliding with the enemy ship. Even if the ship is moved using the 'move' feature and collides with the enemy ship, statements like the one shown in figure four will be checked at the end of the turn to see if points are granted. Although a Starfighter may get destroyed causing the game to end, points will still be granted by the if-statement as long as the enemy ship gets destroyed.

This supports programming from the interface because essentially the user can decide how to get points to increase their score. Since the game is never ending, the user may decide to move the ship to collide with the Grunt. The user can decide to set their own coordinates to collide with the enemy ship knowing that they will be granted points upon the destruction of the enemies ship. This is done to support programming from the interface and not just the implementation where the ships are only free moving.