EECS3311-W19 — Project Report

Submitted electronically by:

Team members	Name	Prism Login	Signature
Member 1:	Martin Zlatkin	mzlatkin	Martin Zlatkin
Member 2:	Paul Owe	pcowe	Paul Owe
*Submitted under Prism account:		mzlatkin	

* Submit under **one** Prism account only

Also submit a printed version with signatures in the course Drop Box

Contents

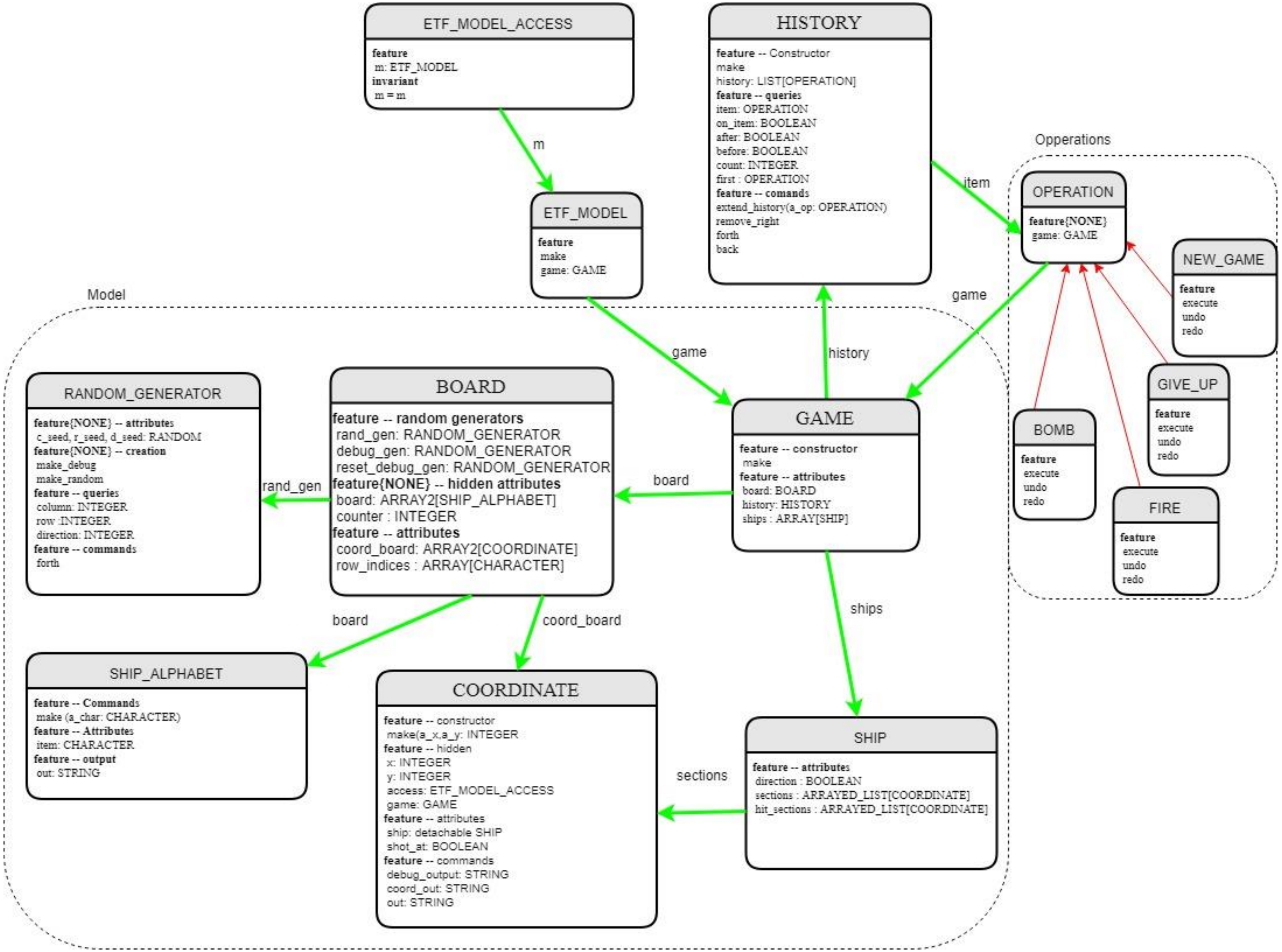
Documentation must be done to professional standards. See OOSC2 Chapter 26: *A sense of style*. Code and contracts must be documented using the Eiffel and BON style guidelines and conventions. *CamelCase* is used in Java. In Eiffel the convention is *under_score*. Attention must be paid to using appropriate names for classes and features. Class names must be upper case, while features are lower case. Comments and header clauses are important. For class diagrams, use the BON conventions, and use clusters as appropriate. Use the EiffelStudio document generation facility (e.g. text, short, flat etc. RTF views), suitably edited and indented to prevent wrapping, to help you obtain appropriately documentation (e.g. contract views). Each diagram must be at the appropriate level of abstraction. Use Visio for the BON class diagrams.

Your signature attests that this is your own work and that you have obeyed university academic honesty policies. Academic honesty is essentially giving credit where credit is due, and not misrepresenting what you have done and what work you have produced. When a piece of work is submitted by a student it is expected that all unquoted and uncited ideas and text are original to the student. Uncited and unquoted text, diagrams, etc., which are not original to the student, and which the student presents as their own work is considered academically dishonest.

1. Requirements for Project "Battleship"

Our instructor provided us with the following statement of their needs: The subject is to play a battleship game. A new game may be started in one of two ways (either Debug mode or Normal mode). The main difference between these two modes is that debug mode will allow the player to view the placement of the ships on the board whereas in normal mode, the ship placements are not visible. A game can be played on different difficulty settings ranging from easy, medium, hard to advanced. Our battleship game also allows a user to have custom games with custom_setup_test or custom_setup. Custom_setup_test is used in debug mode whereas custom_setup is used in normal mode. Custom games provide you the ability to determine the size of the grid, the number of fire shots, the number of ships, as well as the number of bomb shots. A fire command has the potential to hit only one coordinate. A bomb command will hit two coordinates. A ship is only sunk if all its coordinates are hit or bombed. This battleship game also provides numerous error handling messages in the event a player runs commands incorrectly. For instance, an "Invalid Coordinates" error message is returned in the event a player provides a coordinate that is out of the board's range.

See *battleship.ui.txt* for the grammar of the user interface. The acceptance tests *at1.expected.txt* and *at2.expected.txt* describe some of the input-output behavior at the console for this project.


ETF_MODEL_ACCESS
feature
m: ETF_MODEL
invariant
m = m
m
ETF_MODEL
feature
make
game: GAME
HISTORY
feature -- Constructor
make
history: LIST[OPERATION]
feature -- queries
item: OPERATION
on_item: BOOLEAN
after: BOOLEAN
before: BOOLEAN
count: INTEGER
first : OPERATION
feature -- comands
extend_history(a_op: OPERATION)
remove_right
forth
back
item
Opperations
OPERATION
feature{NONE}
game: GAME
NEW_GAME
feature
execute
undo
redo
GIVE_UP
feature
execute
undo
redo
BOMB
feature
execute
undo
redo
FIRE
feature
execute
undo
redo
game
Model
game
history
RANDOM_GENERATOR
feature{NONE} -- attributes
c_seed, r_seed, d_seed: RANDOM
feature{NONE} -- creation
make_debug
make_random
feature -- queries
column: INTEGER
row :INTEGER
direction: INTEGER
feature -- commands
forth
rand_gen
BOARD
feature -- random generators
rand_gen: RANDOM_GENERATOR
debug_gen: RANDOM_GENERATOR
reset_debug_gen: RANDOM_GENERATOR
feature{NONE} -- hidden attributes
board: ARRAY2[SHIP_ALPHABET]
counter : INTEGER
feature -- attributes
coord_board: ARRAY2[COORDINATE]
row_indices : ARRAY[CHARACTER]
board
GAME
feature -- constructor
make
feature -- attributes
board: BOARD
history: HISTORY
ships : ARRAY[SHIP]
board
coord_board
ships
SHIP_ALPHABET
feature -- Commands
make (a_char: CHARACTER)
feature -- Attributes
item: CHARACTER
feature -- output
out: STRING
COORDINATE
feature -- constructor
make(a_x,a_y: INTEGER
feature -- hidden
x: INTEGER
y: INTEGER
access: ETF_MODEL_ACCESS
game: GAME
feature -- attributes
ship: detachable SHIP
shot_at: BOOLEAN
feature -- commands
debug_output: STRING
coord_out: STRING
out: STRING
sections
SHIP
feature -- attributes
direction : BOOLEAN
sections : ARRAYED_LIST[COORDINATE]
hit_sections : ARRAYED_LIST[COORDINATE]

3. Table of modules — responsibilities and information hiding

1	board	**Responsibility**: includes all game pieces and board	**Alternative**: none
	Module	**Secret**: game states	

1.1	GAME	**Responsibility**: tracks game states and values, also builds and outputs the game board on screen	**Alternative**: none
	Concrete	**Secret**: game states	

1.2	COORDINATE	**Responsibility**: to store if a specific coordinate is hit and if it has a ship on it	**Alternative**: none
	Concrete	**Secret**: coordinate values	

1.3	SHIP	**Responsibility**: record data of a ship, output ship values	**Alternative**: none
	Concrete	**Secret**: none	

1.4	BORAD	**Responsibility**: builds the game board and places ship every new game, holds the 2d array of coordinates	**Alternative**: none
	Concrete	**Secret**: implemented in contiguous memory amortized over constant time re-allocation	

1.5	HISTORY	**Responsibility**: Iterator object responsible for keeping track of operations in history	**Alternative**: none

	Concrete	**Secret**: none	

2	OPERATION	**Responsibility**: abstract class of all operations, also sets some operation messages generic to all operations	**Alternative**: None
	Abstract	**Secret**: check_hit_or_win: checks if there was a ship hit this turn and if the game is over	

2.1	BOMB	**Responsibility**: to check bomb validity and to call the main GAME to bomb the coordinates	**Alternative**: None
	Concrete	**Secret**: bomb validation	

2.2	FIRE	**Responsibility**: to check fire validity and to call the main GAME to fire on the coordinates	**Alternative**: None
	Concrete	**Secret**: fire validation	

2.3	NEW_GAME	**Responsibility**: to track new_game calls and new_game validity	**Alternative**: None
	Concrete	**Secret**: new game validation	

2.4	GIVE_UP	**Responsibility**: to track give_up calls	**Alternative**: None
	Concrete	**Secret**: None	

4. Expanded description of design decisions

GAME Module

The game module is responsible for all the main functions of the battleship program. Every time a user makes a new command a new OPERATION object is made and that object uses game to execute its main functions.

GAME has many features that are used to track game states, these include features things like score and ships which is a list of SHIP objects that is used for end of game checks. It also keeps track of whether or not the game is currently running in debug mode. If it is an important value that is likely going to be used by another class then it would be in game. In the same vain, GAME is the only way to access the BOARD object thereby enforcing the singleton pattern.

Other classes use some features from game to determine their own validity, such as FIRE or BOMB using the game_over feature from game. This was done so that all important game states can be found in a single class.

The commands in GAME include new_game, give_up, fire, and bomb, these are all external facing commands that are called by OPERATIONS to execute or undo their respective functions. Other commands include reset_score, and reset_game, these functions provide a way for GAME itself to reset a game when required and to change the score when needed.

new_game takes several parameters provided by the operation NEW_GAME to initialize the values for a new game to be played, it also recreates the BOARD object with the new values.

reset_game resets all relevant values to their defaults, ensuring that the new game does not keep any unnecessary information from the old game.

give_up requires that the game has started and that the game is not over and that the user has not already given up, after that the function reverts all scores back to the way they were before the game started and allows the user to make a new game without saving data from the old game.

reset_score updates scores based off of ship statuses. At first scores were calculated per ship hit, but, that was incorrect, instead score calculation was delegated to its own function. In this function I check every ship to see if they have been sunk, if they have I update the score based on the ships size. I also keep track of separate scores, debug_score and the “normal” score, this way when the user starts a new game in either mode I can erase or store old data.

fire and bomb are very similar, they both begin by validating the coordinates that are passed in to make sure that they are in fact valid coordinates. They also take a Boolean value that tracks whether or not the function needs to be undone or executed. These functions update several game state features such as hit_this_turn which is used by OPERATIONS to create their respective messages. In these functions I have COORDINATES set to being hit or “unhit” and the same for ships if they are on that coordinate. These functions end by calling the reset_score function which recalculates the scores.

GAME also includes several queries that are used by different functions in GAME like validate_coordianates or check_game_status that will return whether or not the game has ended.

5. Significant Contracts (Correctness)

SHIP and GAME have the most significant contracts

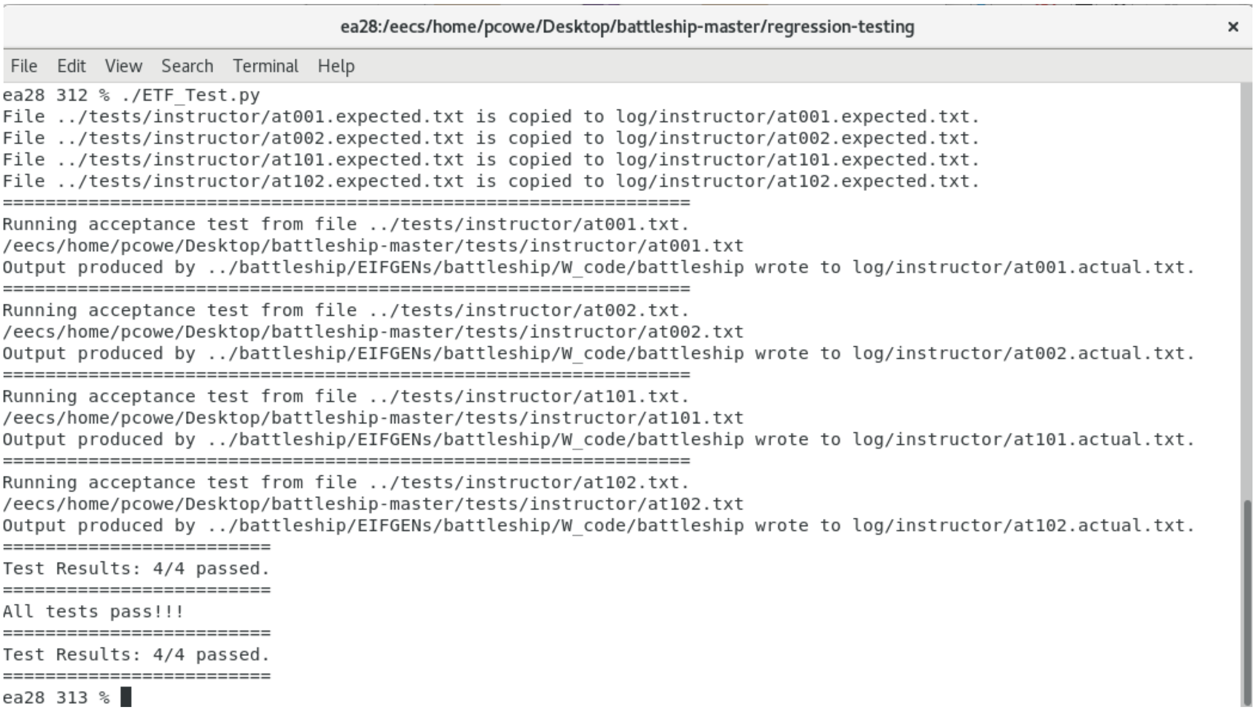
In SHIP it is important that if a section of that ship is being hit that section is actually part of that ship. The contracts in ship make sure that the COORDINATES showing up in hit_sections are also in sections, this ensures that a ship can never be sunk unless it actually has been properly hit.

GAME includes contracts that validate coordinates for fire and bomb and also requires that games are started and not over before someone can give up.

6. Summary of Testing Procedures

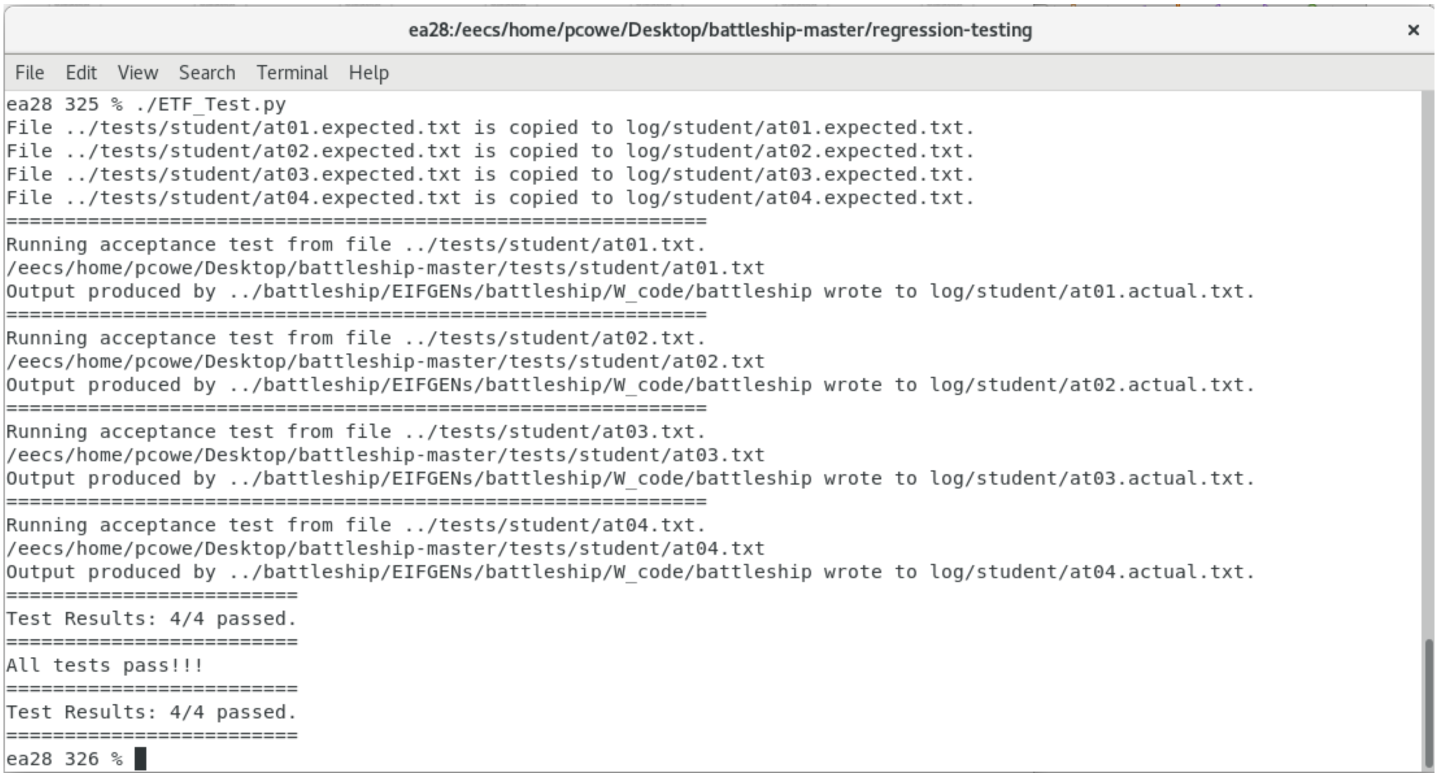
Test File	Description	Passed
at01.txt	undo/redo test	PASSED
at02.txt	give_up test	PASSED
at03.txt	debug_test levels	PASSED
at04.txt	bomb test	PASSED
at001.txt	Instructor test 1	PASSED
at002.txt	Instructor test 2	PASSED
at101.txt	Instructor test 3	PASSED
at102.txt	Instructor test 4	PASSED

SCREENSHOT OF INSTRUCTOR TESTS


ea28:/eecs/home/pcowe/Desktop/battleship-master/regression-testing
×
File
Edit
View
Search
Terminal
Help

```
ea28 312 % ./ETF_Test.py
File ../tests/instructor/at001.expected.txt is copied to log/instructor/at001.expected.txt.
File ../tests/instructor/at002.expected.txt is copied to log/instructor/at002.expected.txt.
File ../tests/instructor/at101.expected.txt is copied to log/instructor/at101.expected.txt.
File ../tests/instructor/at102.expected.txt is copied to log/instructor/at102.expected.txt.
================================================================
Running acceptance test from file ../tests/instructor/at001.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/instructor/at001.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/instructor/at001.actual.txt.
================================================================
Running acceptance test from file ../tests/instructor/at002.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/instructor/at002.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/instructor/at002.actual.txt.
================================================================
Running acceptance test from file ../tests/instructor/at101.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/instructor/at101.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/instructor/at101.actual.txt.
================================================================
Running acceptance test from file ../tests/instructor/at102.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/instructor/at102.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/instructor/at102.actual.txt.
=========================
Test Results: 4/4 passed.
=========================
All tests pass!!!
=========================
Test Results: 4/4 passed.
=========================
ea28 313 % █
```

SCREENSHOT OF STUDENT TESTS


ea28:/eecs/home/pcowe/Desktop/battleship-master/regression-testing
×
File
Edit
View
Search
Terminal
Help

```
ea28 325 % ./ETF_Test.py
File ../tests/student/at01.expected.txt is copied to log/student/at01.expected.txt.
File ../tests/student/at02.expected.txt is copied to log/student/at02.expected.txt.
File ../tests/student/at03.expected.txt is copied to log/student/at03.expected.txt.
File ../tests/student/at04.expected.txt is copied to log/student/at04.expected.txt.
============================================================
Running acceptance test from file ../tests/student/at01.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/student/at01.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/student/at01.actual.txt.
============================================================
Running acceptance test from file ../tests/student/at02.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/student/at02.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/student/at02.actual.txt.
============================================================
Running acceptance test from file ../tests/student/at03.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/student/at03.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/student/at03.actual.txt.
============================================================
Running acceptance test from file ../tests/student/at04.txt.
/eecs/home/pcowe/Desktop/battleship-master/tests/student/at04.txt
Output produced by ../battleship/EIFGENs/battleship/W_code/battleship wrote to log/student/at04.actual.txt.
=========================
Test Results: 4/4 passed.
=========================
All tests pass!!!
=========================
Test Results: 4/4 passed.
=========================
ea28 326 % █
```

7. Appendix (Contract view of all classes)

```
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {GAME}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        GAME

create {ETF_MODEL}
        make

feature -- creation

        make

feature -- attributes

        board: BOARD

        history: HISTORY

        game_board_print: STRING_8

        debug_mode: BOOLEAN

        set_debug_mode_to (to_debug_mode: BOOLEAN)

        bombs_total: INTEGER_32

        bombs_shot: INTEGER_32

        fires_total: INTEGER_32

        fires_shot: INTEGER_32
                        --game over information

        gave_up: BOOLEAN

        game_over: BOOLEAN

        started: BOOLEAN
                        -- check if the first ever game has been started

        set_started

        ship_sank_this_turn: ARRAY [SHIP]

        first_shot: BOOLEAN

        set_first_shot_to (this: BOOLEAN)

feature -- game_start

        new_game (board_size, ships_number, max_shots, max_bombs: INTEGER_32)

        set_up_game_board (size, ship_number: INTEGER_32)

        reset_game

feature -- commands

        give_up
                require
                                started ~ True and not game_over and not gave_up
```

```
    reset_score

    fire (x, y: INTEGER_32; undo: BOOLEAN)
        require
                validate_coordinate (x, y)

    bomb (l_x, l_y, r_x, r_y: INTEGER_32; undo: BOOLEAN)
        require
                validate_coordinate (l_x, l_y) and validate_coordinate (r_x, r_y)

feature -- queries

    was_a_ship_hit_this_turn: BOOLEAN

    validate_coordinate (x, y: INTEGER_32): BOOLEAN

    check_game_status: BOOLEAN

    check_ship_status: BOOLEAN

    check_fire_status: BOOLEAN

feature -- output

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

    update_game_board: STRING_8

end -- class GAME
            -- Generated by ISE Eiffel --
            -- For more details: http://www.eiffel.com --
```

```
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {BOARD}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        BOARD

create
        make

feature -- random generators

        rand_gen: RANDOM_GENERATOR
                        -- random generator for normal mode
                        -- it's important to keep this as an attribute

        debug_gen: RANDOM_GENERATOR
                        -- deterministic generator for debug mode
                        -- it's important to keep this as an attribute

        reset_debug_gen: RANDOM_GENERATOR

feature -- attributes

        coord_board: ARRAY2 [COORDINATE]

        Row_indices: ARRAY [CHARACTER_8]

        size: INTEGER_32
                        -- size of board

feature -- creation

        make
                        -- Initialization for Current.

feature -- query

        get_coord_board: ARRAY2 [COORDINATE]

feature -- utilities

        reset_game (r_size: INTEGER_32)

        generate_ships (is_debug_mode: BOOLEAN; board_size: INTEGER_32; num_ships: INTEGER_32;
was_debug_mode: BOOLEAN): ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]]
                        -- places the ships on the board
                        -- either deterministicly random or completely random depending on debug
mode
```

```
        collide_with_each_other (ship1, ship2: TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]): BOOLEAN
                        -- Does ship1 collide with ship2?

        collide_with (existing_ships: ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32;
col: INTEGER_32; dir: BOOLEAN]]; new_ship: TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]): BOOLEAN
                        -- Does new_ship collide with the set of existing_ships?
                ensure
                                Result = across
                                        existing_ships as existing_ship
                                some
                                        collide_with_each_other (new_ship, existing_ship.item)
                                end

        set_square (x, y: INTEGER_32)

feature --ship placement

        place_new_ships (new_ships: ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]])
                        -- Place the randomly generated positions of new_ships onto the board.
                        -- Notice that when a ship's row and column are given,
                        -- its coordinate starts with (row + 1, col) for a vertical ship,
                        -- and starts with (row, col + 1) for a horizontal ship.
                require
                                across
                                        new_ships.Lower |..| new_ships.upper as i
                                all
                                        across
                                                new_ships.Lower |..| new_ships.upper as j
                                        all
                                                i.item /= j.item implies not collide_with_each_other
(new_ships [i.item], new_ships [j.item])
                                        end
                                end

end -- class BOARD
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {COORDINATE}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        COORDINATE

create
        make

feature -- make

        make (a_x, a_y: INTEGER_32)
                        -- may not be a valid square
        feature -- hidden

        x: INTEGER_32

        y: INTEGER_32

        Access: ETF_MODEL_ACCESS

        Game: GAME
        feature -- attributes

        ship: detachable SHIP assign set_ship

        shot_at: BOOLEAN
        feature -- commands

        set_shot_at (hit: BOOLEAN)

        set_coords (a_x, a_y: INTEGER_32)

        set_ship (a_ship: detachable SHIP)

        debug_output: STRING_8
                        -- String that should be displayed in debugger to represent Current.

        coord_out: STRING_8

        out: STRING_8
                        -- New string containing terse printable representation
                        -- of current object
        end -- class COORDINATE
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "History operations for undo/redo design pattern"
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        HISTORY

create
        make

feature -- queries

        item: OPERATION
                        -- Cursor points to this user operation
                require
                                on_item

        on_item: BOOLEAN
                        -- cursor points to a valid operation
                        -- cursor is not before or after

        after: BOOLEAN
                        -- Is there no valid cursor position to the right of cursor?

        before: BOOLEAN
                        -- Is there no valid cursor position to the left of cursor?

        count: INTEGER_32

        first: OPERATION
        feature -- comands

        extend_history (a_op: OPERATION)
                        -- remove all operations to the right of the current
                        -- cursor in history, then extend with a_op
                ensure
                                history [history.count] = a_op

        remove_right
                        --remove all elements
                        -- to the right of the current cursor in history

        forth
                require
                                not after

        back
                require
                                not before
        end -- class HISTORY
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com --
```

```
                       -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {FIRE}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        BOMB

create
        make

feature -- queries

        l_x: INTEGER_32

        l_y: INTEGER_32

        r_x: INTEGER_32

        r_y: INTEGER_32

        valid_bomb: BOOLEAN
        feature -- commands

        validate_bomb: BOOLEAN

        execute

        undo

        redo
        end -- class BOMB
                       -- Generated by ISE Eiffel --
                       -- For more details: http://www.eiffel.com --
                       -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {FIRE}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        FIRE

create
        make

feature -- queries

        x: INTEGER_32

        y: INTEGER_32
        feature -- commands

        validate_fire: BOOLEAN

        execute

        undo

        redo
        end -- class FIRE
                       -- Generated by ISE Eiffel --
                       -- For more details: http://www.eiffel.com --
                       -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {FIRE}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        GIVE_UP

create
        make

feature -- commands

        execute

        undo

        redo
        end -- class GIVE_UP
                       -- Generated by ISE Eiffel --
                       -- For more details: http://www.eiffel.com --
                       -- Automatic generation produced by ISE Eiffel --
note
       description: "Summary description for NEW_GAME. the most useless calss since it can never
be undone or redone"
       author: ""
       date: "$Date$"
       revision: "$Revision$"

class interface
       NEW_GAME

create
       make,
       make_custom

feature -- queries

       board_size: INTEGER_32

       ships_number: INTEGER_32

       max_shots: INTEGER_32

       max_bombs: INTEGER_32

       debug_mode: BOOLEAN
       feature -- commands

       validate_new_game: BOOLEAN

       execute

       undo

       redo
       end -- class NEW_GAME
                       -- Generated by ISE Eiffel --
                       -- For more details: http://www.eiffel.com --
```

```
                        -- Automatic generation produced by ISE Eiffel --
note
        description: "Summary description for {OPERATION}."
        author: ""
        date: "$Date$"
        revision: "$Revision$"

deferred class interface
        OPERATION

feature -- queries

        state: INTEGER_32

        post_state: INTEGER_32

        next_message: STRING_8

        first_shot: BOOLEAN

        append_message (a_message: STRING_8)

        prepend_message (a_message: STRING_8)

        set_next_message (a_message: STRING_8)
        feature -- deferred commands

        execute

        undo

        redo
        end -- class OPERATION
                        -- Generated by ISE Eiffel --
                        -- For more details: http://www.eiffel.com –
```