EECS3311-W19 — Project Report

Submitted electronically by:

Team members	**Name**	**Prism Login**	**Signature**
Member 1:	Meet Patel	meet2610	M.R.Patel
Member 2:	Amir Younesi	kxn	AY
*Submitted under Prism account:		meet2610	

* 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.

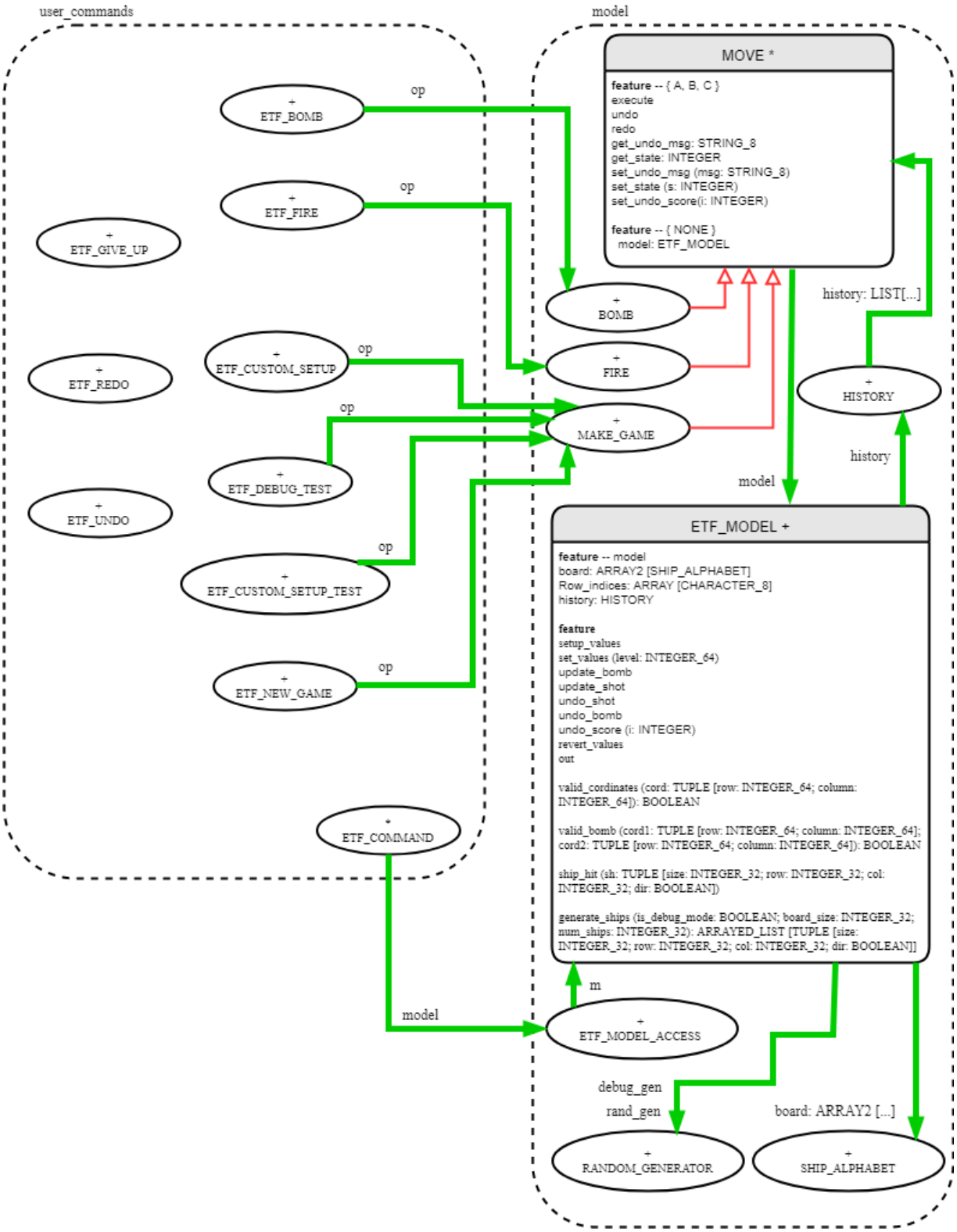
1.Requirements for Project Battleship

a brief outline to Project Battleship

Our customer was bored with playing connect 4 on their own and decided to spice things up and required a game of battleship for which they had full functionality over the game: Being able to choose what game mode they want to play (Easy, Medium, Hard, Advanced or even Custom).So depending on how the customer feels, they're able to choose a game of their own choice. Dependent on difficulty , we are given specification of each level, for easy gets a 4x4 grid and gets 8 shots and 2 bombs for 2 ships; and medium get a 6x6 grid, 16 shots 3 bombs for 3 ships, and custom can choose the NxN grid, N shots and N bombs for N ships, where n is the customers choice depending on what they choose. More on this topic can be read over at battleship.ui.txt. A feature that has been implemented in this game of battleship is being able to Undo the most recent move, or maybe even Undo to the entire beginning so you can get a fresh restart, along with the feature of being able to Redo moves that you have had undone, there many more commands that partake in this game of battleship, which as earlier mentioned can be read in battleship.ui.txt. A console-based application is used to handle the user input specifications for the game and their bounds and if any error to occur when using these commands, the correct way to use command is given just in case the user has no idea on how to correctly use the command.

See battleship.ui.txt in the appendix for additional grammar of the user interface.
And there is a list of acceptance tests: at1.expected.txt, at2.expected.txt, at3.expected.txt and at4.expected.txt

2. Requirements for Project Battleship



3. Table of modules — responsibilities and information hiding

1	ETF_BOMB	**Responsibility**: fire bomb at valid coordinates if a game is in progress	**Alternative**: none
	Concrete	**Secret**: none	

2	ETF_FIRE	**Responsibility**: fire shot at valid coordinates if a game is in progress	**Alternative**: none
	Concrete	**Secret**: none	

3	ETF_CUSTOM_SETUP	**Responsibility**: create a game, if no game is in progress, with a valid user input of board size, ships, shots, bombs	**Alternative**: none
	Concrete	**Secret**: none	

3	ETF_CUSTOM_SETUP_TEST	**Responsibility**: create a game, if no game is in progress, with a valid user input of number of boardsize, ships, shots, bombs with debug mode ON	**Alternative**: none
	Concrete	**Secret**: none	

5	ETF_DEBUG_TEST	**Responsibility**: create a game, if no game is in progress, with a valid level of difficulty with debug mode ON	**Alternative**: none
	Concrete	**Secret**: none	

6	ETF_NEW_GAME	**Responsibility**: create a game, if no game is in progress, with a valid level of difficulty	**Alternative**: none
	Concrete	**Secret**: none	

7	ETF_GIVE_UP	**Responsibility**: End current game and restore model to end of previous game	**Alternative**: none
	Concrete	**Secret**: none	

8	ETF_UNDO	**Responsibility**: if move has been made, undo previous move	**Alternative**: none
	Concrete	**Secret**: none	

9	ETF_REDO	**Responsibility**: redo the undo move, if any	**Alternative**: none
	Concrete	**Secret**: none	

10	MOVE	**Responsibility**: record information about each user command	**Alternative**: none
	abstract	**Secret**: none	

10.1	FIRE	**Responsibility**: records information about what ETF_FIRE has done, and how to undo it	**Alternative**: none
	Concrete	**Secret**: none	

10.2	BOMB	**Responsibility**: records information about what ETF_BOMB has done, and how to undo it	**Alternative**: none
	Concrete	**Secret**: none	

10.3	MAKE_GAME	**Responsibility**: records information about what ETF_DEBUG_TEST, ETF_NEW_GAME, ETF_CUSTOM_SETUP, ETF_CUSTOM_SETUP_TEST has done, and how to undo it	**Alternative**: none
	Concrete	**Secret**: none	

11	HISTORY	**Responsibility**: holds each MOVE in a LIST	**Alternative**: none
	Concrete	**Secret**: none	

11.1	LIST	**Responsibility**: a sequence of moves in history	**Alternative**: ARRAY
	Concrete	**Secret**: none	
12	ETF_MODEL_ACCESS	**Responsibility**: create a singleton access to ETF_MODEL	**Alternative**: none
	Concrete	**Secret**: none	

13	ETF_MODEL	**Responsibility**: create board, generate ships, handle game state changes, handle error messages, produce output	**Alternative**: none
	Concrete	**Secret**: board implemented using ARRAY2[SHIP_ALPHABET] Ships generated using RANDOM_GENERATOR	

13.1	ARRAY2	**Responsibility**: see ETF_MODEL Used to implement the board	**Alternative**: none
	Concrete	**Secret**: none	

14	RANDOM_GENERATOR	**Responsibility**: help generate ships the same way as the oracle	**Alternative**: none
	Concrete	**Secret**: none	

15	SHIP_ALPHABET	**Responsibility**: used to only allow specific symbols on the board	**Alternative**: none
	Concrete	**Secret**: none	

4. Expanded description of design decisions

The most significant module in this project would be the *ETF_MODEL*, It's the most crucial class that is involved, because of the responsibilities it holds such as: create board, generate ships, handle game state changes, handle error messages and produce outputs. The design of this class isn't the best, it is seen as a superman class where everything has been shoved inside, it wasn't designed to look like this at the start of the design but ended up as the result. Which could have been reworked at the beginning and make sub-classes to clean up the class since its not very reusable.

Given that, starting off with how the board was created we used *ARRAY2* because it best used effectively in this scenario of generating a grid of nxn (2D array) It would have been very difficult in creating an alternative of an implementation for the gameboard, Then depending on the amount of space the user needs, the columns and rows are labelled correctly by using an array to display the titles.

Next comes to the ship generation, which was given at the start as well as the board somewhat, had ships that were generated somewhere on the board with a valid coordinate, along with the length of the ship and their head and tail location. This was implemented with a *LIST* part of the *ARRAY_LIST* class, we decided to keep the default implementation in an array, since this design made most sense for keeping track of all this information, making sure the coordinates valid and making sure the ships don't collide on the board required for the information to be retrieved in constant time which is what was able to be achieved.

There's not much to the design of how we handle game state changes, such as undo bomb, undo shot, undo score, update bomb update shot update state, these are all features that have simple implementations with a straightforward constraints for example, undoing a bomb would require to decrease the bomb thrown count by 1 but if there is 0 thrown bombs to begin with and 0 thrown shots then this situation would be handled, that logic can be used for the rest of the features to explain the implementation for the rest of these features.

5. Significant Contracts (Correctness)

(only for the module with the most significant contracts)

.

The module with the most significant contract in our design is the HISTORY module.

There is a pre-condition if someone is trying to access the item: MOVE at current cursor position. The pre-condition requires the cursor to be on_item.

There is a post-condition for the wipe_out feature. The post-condition ensures that the history count is 0. Meaning all the objects in the LIST are removed.

There is a post-condition for extend_history(a_op: MOVE). The post-condition ensures that the a_op has been added to the end of the LIST

There is a pre-condition for forth. The pre-condition requires not after, which makes sure that there is an item after the current cursor position.

There is a pre-condition for forth. The pre-condition requires not before, which makes sure that there is an item before the current cursor position.

This is a significant module as it is required to implement the undo/redo design pattern. User moves of FIRE, BOMB, and MAKE_GAME is stored in history. The contracts ensure that it functions properly without issue.

6. Summary of Testing Procedures

Validity checks	At01.txt At02.txt At03.txt At04.txt At05.txt At06.txt At07.txt At08.txt	Testing valid fire/bombs in easy game mode
	At10.txt	Testing valid fire/bombs in advanced game mode
	At11.txt	Testing a loss in medium game mode
	At12.txt	Testing multiple debug games and firing in-between
Failure checks	At13.txt At14.txt At15.txt	Making sure two games can't be run at the same time
	At16.txt At17.txt At18.txt	Testing game commands when game isn't running
	At17.txt At15.txt	Testing a fire on the same coordinate
Undo/redo checks	At18.txt At19.txt At21.txt At20.txt	Undoing/redoing a state that doesn't exist
Give-up checks	At22.txt At23.txt	Series of games being given up (never started to begin with)
Custom tests	At24.txt At22.txt At21.txt	Series of games being built using custom setup
Normal game runs	At25.txt At26.txt	Games that are being played out normally

7. Appendix (Contract view of all classes)

note

description: "Summary description for {BOMB}."

author: ""

date: "$Date$"

revision: "$Revision$"

class interface

BOMB

create

make

feature

execute

get_bomb_valid: BOOLEAN

-- returns if move was valid

get_state: INTEGER_32

-- returns the state at which this move occured

get_undo_msg: STRING_8

```
                -- returns the output message from ETF_MODEL that was stored

        get_undo_score: INTEGER_32
                -- returns how much the score changed by with this move

        redo

        set_bomb_valid (b: BOOLEAN)
                -- stores information regarding if this bomb was valid or not

        set_state (i: INTEGER_32)
                -- stores the state number at which this move occured

        set_undo_msg (msg: STRING_8)
                -- stores the message output of ETF_MODEL for the given move

        set_undo_score (i: INTEGER_32)
                -- stores the change in score that occured by this move

        undo

end -- class BOMB
```

```
class interface
	ETF_MODEL

create {ETF_MODEL_ACCESS}
	make_empty

feature

	bad_bomb (b: BOOLEAN)
			-- Signals to output that Bomb or Shot hit one or more ship

	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
		ensure
			Result = across
				existing_ships as existing_ship
			some
				collide_with_each_other (new_ship, existing_ship.item)
			end

	collide_with_each_other (ship1, ship2: TUPLE [size: INTEGER_32; row: INTEGER_32; col: INTEGER_32; dir: BOOLEAN]): BOOLEAN

	default_update
```

generate_ships (is_debug_mode: BOOLEAN; board_size: INTEGER_32; num_ships: INTEGER_32): ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col: INTEGER_32; dir: BOOLEAN]]

-- Ship generation

giveup (b: BOOLEAN; msg: STRING_8)

-- Signals to output there are no more shots

-- Signals to output that Bomb or Shot missed all ships

make_empty

-- create an empty board

-- create an empty history

-- instantiate attributes that defer per game

out: STRING_8

-- New string containing terse printable representation

-- of current object

reset

revert_stuff

-- reverts game score to end of previous game if game was given up using ETF_GIVE_UP

-- updating game values

-- decreases bomb thrown count by 1

-- decreases shot thrown count by 1

revert_values

set_bomb_msg (b: BOOLEAN)

set_game_active (b: BOOLEAN)

set_go_msg (b: BOOLEAN)

set_hit (b: BOOLEAN)

-- Checks if the ship that has been hit, is no sunked or not

-- Sets output message if ship has sunk

-- Updates score if ship has sunk

-- Checks if game is over or not

set_invalid_cord (b: BOOLEAN)

-- Signals to output game not started message

set_miss (b: BOOLEAN)

set_new_game (b: BOOLEAN)

set_shot_msg (b: BOOLEAN)

set_undoredo (b: BOOLEAN; msg: STRING_8)

set_values (level: INTEGER_64)

-- Setting up game values

-- Sets shots, bombs, and score value for game depending on difficulty level of ETF_NEW_GAME and ETF_DEBUG_TEST

setup_valid (b: BOOLEAN; msg: STRING_8)

-- Signals to output invalid bomb message

-- Signals to output game has already started

-- Signals to output cordinates are invalid

-- Signals to output there are no more bombs

setup_values (ms1: INTEGER_32; mb1: INTEGER_32; score1: INTEGER_32)

-- Sets shots, bombs, score values depending on the values given by ETF_CUSTOM_SETUP_TEST and ETF_CUSTOM_SETUP

ship_hit (sh: TUPLE [size: INTEGER_32; row: INTEGER_32; col: INTEGER_32; dir: BOOLEAN])

-- output producer

-- Produces output that matches the oracle character for character

ships_out: STRING_8

-- Produces ship output for non debug_mode

stats: STRING_8

-- Produces ship output for debug_mode

sunk_ship: STRING_8

undo_bomb

-- increases shot thrown by 1

undo_score (i: INTEGER_32)

undo_shot

-- increases game state by 1

-- checking for valid shot and bomb

-- checks for valid cordinates for ETF_FIRE and ETF_BOMB

-- checks for repeat_fire

undoredo_change (b: BOOLEAN; msg: STRING_8)

update_bomb

update_shot

valid_bomb (cord1: TUPLE [row: INTEGER_64; column: INTEGER_64]; cord2: TUPLE [row: INTEGER_64; column: INTEGER_64]): BOOLEAN

-- Message and Error handling

-- Recieves message if undo or redo changed state of the game

-- Recieves message if there was nothing to undo or redo

-- Recieves message if ETF_CUSTOM_SETUP_TEST or ETF_CUSTOM_SETUP had invalid setups

-- Recieves message if ETF_GIVE_UP ended a game in progress

valid_cordinates (cord: TUPLE [row: INTEGER_64; column: INTEGER_64]): BOOLEAN

-- checks if bomb cordinates from ETF_BOMB are adjacent

-- checks for repeat_fire

z: INTEGER_32

feature -- attributes

bad_bomb_msg: BOOLEAN

board: ARRAY2 [SHIP_ALPHABET]

board_s: INTEGER_32

bomb_msg: BOOLEAN

debug_mode: BOOLEAN

destroyed_ship: INTEGER_32

game_active: BOOLEAN

game_count: INTEGER_32

game_over: BOOLEAN

game_started: BOOLEAN

give_up_msg: BOOLEAN

give_up_out: STRING_8

go_msg: BOOLEAN

history: HISTORY

hit: BOOLEAN

invalid_cord: BOOLEAN

max_bombs: INTEGER_32

max_score: INTEGER_32

max_shots: INTEGER_32

max_total_score: INTEGER_32

miss: BOOLEAN

new_game: BOOLEAN

no_bombs_shots: BOOLEAN

no_game: STRING_8

no_more_bombs: BOOLEAN

no_more_shots: BOOLEAN

no_undoredo: BOOLEAN

no_undoredo_msg: STRING_8

played_move: BOOLEAN

prev_game_revert: BOOLEAN

repeat_fire: BOOLEAN

Row_indices: ARRAY [CHARACTER_8]

score: INTEGER_32

ship_1: STRING_8

ship_2: STRING_8

ship_count: INTEGER_32

ship_msg: STRING_8

ship_sunk: BOOLEAN

ships: ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col: INTEGER_32; dir: BOOLEAN]]

shot_msg: BOOLEAN

sunk: BOOLEAN

switch_debug: BOOLEAN

thrown_bombs: INTEGER_32

thrown_shots: INTEGER_32

```
        total_score: INTEGER_32

        undoredo: BOOLEAN

        undoredo_msg: STRING_8

        valid_setup_msg: BOOLEAN

        valid_setup_out: STRING_8

feature -- random generators

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

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

end -- class ETF_MODEL
```

```
expanded class interface
	ETF_MODEL_ACCESS

create
	default_create

feature

	M: ETF_MODEL

invariant
		M = M

end -- class ETF_MODEL_ACCESS

note
	description: "Summary description for {FIRE}."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	FIRE

create
```

make

feature

execute

-- perform some update on the model state

get_shot_valid: BOOLEAN

-- returns if move was valid

get_state: INTEGER_32

-- returns the state at which this move occured

get_undo_msg: STRING_8

-- returns the output message from ETF_MODEL that was stored

get_undo_score: INTEGER_32

-- returns how much the score changed by with this move

redo

set_shot_valid (b: BOOLEAN)

-- stores information regarding if this shot was valid or not

set_state (i: INTEGER_32)

```
                -- stores the state number at which this move occured

        set_undo_msg (msg: STRING_8)
                -- stores the message output of ETF_MODEL for the given move

        set_undo_score (i: INTEGER_32)
                -- stores the change in score that occured by this move

        undo
                --                      model.set_hit (false)
                --                              model.set_hit (true)
                --                              model.ship_hit (s.item)
                --
        model.board[coordinate1.row.as_integer_32, coordinate1.column.as_integer_32] :=
create {SHIP_ALPHABET}.make ('_')
                --                              model.set_hit (true)
                --                              model.ship_hit (s.item)
                --                      model.set_hit (false)
                --                              model.set_hit (true)
                --                              model.ship_hit (s.item)
                --                              model.set_hit (true)
                --                              model.ship_hit (s.item)

end -- class FIRE
```

```
note
	description: "History operations for undo/redo design pattern"
	author: "JSO"
	date: "$Date$"
	revision: "$Revision$"

class interface
	HISTORY

create
	make

feature -- comands

	back
		require
				not before

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

```
        forth
                require
                        not after

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

        wipe_out
                ensure
                        history.count ~ 0

feature -- queries

        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

        item: MOVE
                -- Cursor points to this user operation
                require
```

on_item

on_item: BOOLEAN

-- cursor points to a valid operation

-- cursor is not before or after

end -- class HISTORY

note

description: "Summary description for {MAKE_GAME}."

author: ""

date: "$Date$"

revision: "$Revision$"

class interface

MAKE_GAME

create

make

feature

execute

```
	get_state: INTEGER_32
			-- returns the state at which this move occured

	get_undo_msg: STRING_8
			-- returns the output message from ETF_MODEL that was stored

	get_undo_score: INTEGER_32
			-- returns how much the score changed by with this move

	redo

	set_state (s: INTEGER_32)
			-- stores the state number at which this move occured

	set_undo_msg (msg: STRING_8)
			-- stores the message output of ETF_MODEL for the given move

	set_undo_score (i: INTEGER_32)
			-- stores the change in score that occured by this move

	undo

end -- class MAKE_GAME
```

```
note
	description: "Summary description for {MOVE}."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

deferred class interface
	MOVE

feature

	execute

	get_state: INTEGER_32

	get_undo_msg: STRING_8

	get_undo_score: INTEGER_32

	redo

	set_state (s: INTEGER_32)

	set_undo_msg (msg: STRING_8)
```

```
	set_undo_score (i: INTEGER_32)

	undo

end -- class MOVE



	description: "[
		The RANDOM_GENERATOR class is used to generate
		coordinates to place ships on the board. Each
		set represents a new ship and can be attained
		by calling forth.
	]"
	author: "Joshua Phillip"
	date: "June 18th, 2018"
	revision: "1"

class interface
	RANDOM_GENERATOR

create
	make_debug,
	make_random
```

```
feature -- commands

        forth
                -- sets the row, column and direction variables forward
                -- should be called for a new ship or if there is a collision

        revert

feature -- queries

        column: INTEGER_32
                -- returns a random variable used to generate column coordinates

        direction: INTEGER_32
                -- returns a random variable used to generate direction

        row: INTEGER_32
                -- returns a random variable used to generate row coordinates

end -- class RANDOM_GENERATOR
```

```
note
	description: "Alphabet allowed to appear on a battleship board."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	SHIP_ALPHABET

create
	make

feature -- Attributes

	item: CHARACTER_8

feature -- Commands

	make (a_char: CHARACTER_8)

feature -- output

	out: STRING_8
			-- Return string representation of alphabet.
```

```
invariant
        allowable_symbols: item = '_' or item = 'h' or item = 'v' or item = 'O' or item = 'X'

end -- class SHIP_ALPHABET
```