EECS3311-W2019 — Project Report

Submitted electronically by:

Team members	Name	Prism Login	Signature
Member 1:	Taehoon Kim	tkim28	Taehoon Kim
Member 2:			
*Submitted under Prism account:		tkim28	

Contents

1. Requirements for Project Battleship

Battleship project allows 2 users to play battleship game under a complex system. It involves many of error checking as well as multiple behaviors that come from a combination of different commands and user inputs. Because of that, it's important to design properly in order to run the system smoothly. As the specification explained, the key design principles that must be focused are information hiding, modularity, abstraction and separation of concerns. Battleship project basically contains the following commands: commands that allow users to have multiple game mode options (debug_test, new_game, custom_setup, custom_setup_test); commands that attack ships such as fire and bomb; utility commands which allow users to undo, redo and give up the current game.

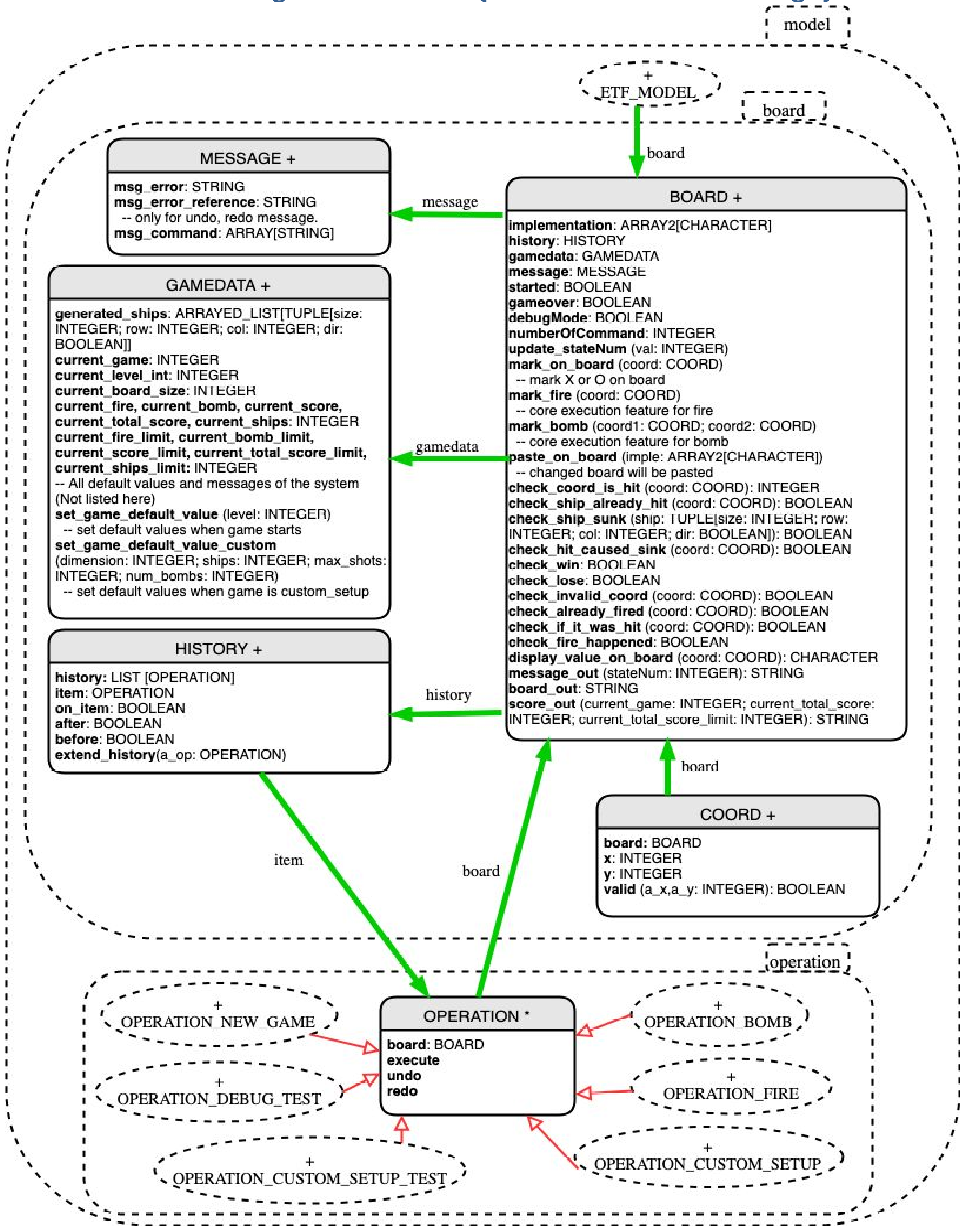
The main goal of the game is to conquer all ships on the board by firing shots or dropping bombs. The board is in a perfect square and the size might be vary depending on the difficulty of the game. When starting a game, the user can choose between normal and debug mode. Debug mode is more like a developer's mode and it generates ships in a fixed order. Otherwise, ships are generated in a random order using RANDOM_GENERATOR class. Custom setup modes have functionality that some variables of the game can be chosen by users. For example, a number of ships can be entered as an argument but the system checks whether it's a reasonable value or not using a specified formula provided in battleship.ui.txt.

Once the game is started, the user should decide whether to fire, bomb, undo, redo or give up the game. All commands are counted as state and attacking commands are saved in history. The history list is later used to track undo and redo commands.

Error detection must be done every time commands are executed. Error messages are provided in battleship.error.txt and conditions for errors must be the same as the oracle.exe program.

Please refer battleship.ui.txt to find more about commands and their conditions.

2. BON class diagram overview (architecture of the design)


model
+ ETF_MODEL
board
board
MESSAGE +
msg_error: STRING
msg_error_reference: STRING
-- only for undo, redo message.
msg_command: ARRAY[STRING]
message
BOARD +
implementation: ARRAY2[CHARACTER]
history: HISTORY
gamedata: GAMEDATA
message: MESSAGE
started: BOOLEAN
gameover: BOOLEAN
debugMode: BOOLEAN
numberOfCommand: INTEGER
update_stateNum (val: INTEGER)
mark_on_board (coord: COORD)
-- mark X or O on board
mark_fire (coord: COORD)
-- core execution feature for fire
mark_bomb (coord1: COORD; coord2: COORD)
-- core execution feature for bomb
paste_on_board (imple: ARRAY2[CHARACTER])
-- changed board will be pasted
check_coord_is_hit (coord: COORD): INTEGER
check_ship_already_hit (coord: COORD): BOOLEAN
check_ship_sunk (ship: TUPLE[size: INTEGER; row: INTEGER; col: INTEGER; dir: BOOLEAN]): BOOLEAN
check_hit_caused_sink (coord: COORD): BOOLEAN
check_win: BOOLEAN
check_lose: BOOLEAN
check_invalid_coord (coord: COORD): BOOLEAN
check_already_fired (coord: COORD): BOOLEAN
check_if_it_was_hit (coord: COORD): BOOLEAN
check_fire_happened: BOOLEAN
display_value_on_board (coord: COORD): CHARACTER
message_out (stateNum: INTEGER): STRING
board_out: STRING
score_out (current_game: INTEGER; current_total_score: INTEGER; current_total_score_limit: INTEGER): STRING
GAMEDATA +
generated_ships: ARRAYED_LIST[TUPLE[size: INTEGER; row: INTEGER; col: INTEGER; dir: BOOLEAN]]
current_game: INTEGER
current_level_int: INTEGER
current_board_size: INTEGER
current_fire, current_bomb, current_score, current_total_score, current_ships: INTEGER
current_fire_limit, current_bomb_limit, current_score_limit, current_total_score_limit, current_ships_limit: INTEGER
-- All default values and messages of the system (Not listed here)
set_game_default_value (level: INTEGER)
-- set default values when game starts
set_game_default_value_custom (dimension: INTEGER; ships: INTEGER; max_shots: INTEGER; num_bombs: INTEGER)
-- set default values when game is custom_setup
gamedata
HISTORY +
history: LIST [OPERATION]
item: OPERATION
on_item: BOOLEAN
after: BOOLEAN
before: BOOLEAN
extend_history(a_op: OPERATION)
history
board
COORD +
board: BOARD
x: INTEGER
y: INTEGER
valid (a_x,a_y: INTEGER): BOOLEAN
item
board
operation
+ OPERATION_NEW_GAME
+ OPERATION_DEBUG_TEST
+ OPERATION_CUSTOM_SETUP_TEST
OPERATION *
board: BOARD
execute
undo
redo
+ OPERATION_BOMB
+ OPERATION_FIRE
+ OPERATION_CUSTOM_SETUP

The main design concern is to make it simple and to separate functions as possible. This way, it’s easier to keep track of bugs as well as adding new functions. Since the final program must be executed like oracle.exe, it’s mandatory to test oracle.exe to find out conditions and relationships between commands and functions.

Class BOARD is where the basic game environment is set up. To prevent this to be a superman class, I have separated roles by creating additional classes such as GAMEDATA and MESSAGE which will be described later. BOARD class keeps track of basic game status whether the game is started or in debug mode and etc. This class holds the main execution process of fire and bomb commands. This process could be added either ETF classes or OPERATION classes but I designed this way because it makes other classes much simpler and easier to reuse the same execution code for other classes. Another important role of this class is to check if the game status should be changed. For example, check_hit_caused_sink feature checks if the hit caused the ship to be sunk. Lastly, this class displays all necessary texts like command messages, board, and game status.

Class MESSAGE is where all messages are stored and retrieved. All messages include errors, hit status, sink status and also guides what users should do next. Command messages are formed differently from errors and undo, redo reference messages because, by the specification of the project, we expect that command messages could be more than 1.

Class GAMEDATA is where most of the game variables are stored. Those variables include current shots, bombs, score, ships as well as all game messages. It also saves ships locations generated when the game started. This class is the main storage class where all data come and go. It makes controlling game data simpler and changing game messages easier.

Class HISTORY is where all commands are stored for mainly undo and redo. It’s a simple class that handles basic functions related to the data type LIST.

Class COORD is the class where coordinates are handled. With this, there is no need to pass x and y coordinates between features and classes. Also, it does a basic validation check whether the coordination is a reasonable value.

Class OPERATION is the data type of commands. It’s functionality deals with execution of the command, undo and redo. For undo and redo message display, it stores all messages after execution.

Also, there are classes inherited from OPERATION. They have the same titles of features but their processes work differently (designed by the definition of inheritance).

3. Table of modules — responsibilities and information hiding

1	BOARD	**Responsibility:** creates game data, history and message, executes attack commands, checks game status, displays on board	**Alternative:** none
	Concrete	**Secret:** game status data to prevent other functions to change game status. Attack commands execution to allow only OPERATION class have access to it. Display features to allow only ETF_MODEL can display game.	

2	MESSAGE	**Responsibility:** stores all game messages	**Alternative:** none
	Concrete	**Secret**: none	

3	GAMEDATA	**Responsibility:** stores all game variables and its default values, message templates.	**Alternative:** none
	Concrete	**Secret**: none	

4	HISTORY	**Responsibility:** stores operations that are executed.	**Alternative:** none
	Concrete	**Secret**: none	

4.1	LIST[OPERATION]	**Responsibility:** See HISTORY	**Alternative:** LINKED_LIST or ARRAYED_LIST might be working.
	Concrete	**Secret**: none	

5	COORD	**Responsibility:** formulate coordination of user input.	**Alternative:** none
	Concrete	**Secret**: none	

6	OPERATION	**Responsibility:** command execution, undo, redo. Also it saves game values after execution and board status.	**Alternative:** none
	Abstract	**Secret**: Each inherited classes has their own game variables that are protected from each other.	

4. Expanded description of design decisions

The most important module in this system is the BOARD. From BOARD, data storages like HISTORY and GAMEDATA are expanded and formulates proper messages to display. More importantly, it executes actual attack commands and checks if the game status is changed. It's like a command center of the system. Let's go over briefly how design pattern is done in this module.

When the BOARD is first created, I needed to decide whether to initialize RANDOM_GENERATOR or not. My decision was 'NO'. Then I put the initialization feature in ETF_MODEL which creates BOARD. This is because of the behavior of oracle.exe. When debug_test runs, the user finishes the game and start debug_test once again, generated ships must be different from previous. So if I initialize RANDOM_GENERATOR when BOARD is created, it'll keep repeating the same pattern of ships.

As a way to describe the board, ARRAY2 is used because the board is a 2-dimensional perfect square. It makes more sense than other 1-dimensional data types. Because it's 2 dimensional, it is never needed to think hard about coordination. Simply, x is x and y is y.

BOARD module handles attack command executions such as fire and bomb. It was definitely possible to handle those in either ETF classes or OPERATION classes. However, to make it reliable across the system, it was needed to put this function in one place where it's connected to almost every other classes in order to easily retrieve data. But at the same time, it was not necessary for all other classes to gain access to this function. That's why protection is added for these features so that only OPERATION classes can use them. To make the code as simple as possible, error checking is separated and placed in ETF classes. The basic procedures of this function are: marking X or O on the board depending on hit or miss conditions; changing game variables; checking if this changes game status such as a win or lose.

As mentioned above, it checks if the command caused the game status to be changed. Conditions related to this function are also added in BOARD. For example, it checks if the coordination is a hit or miss, checks if the coordination is already a hit, checks if a specific size of the ship is sunken and checks if the coordination caused a ship to be sunken. Since they are somehow related to each other and referencing one to another, they needed to be in one place. For simplicity, they could be gathered in a separate class, but I thought it'll cause more confusion if I do so because they are not reused in other places.

5. Significant Contracts (Correctness)

The most significant contracts occur in BOARD module. By the specification of battleship system, it is important to display error on the screen so that users can correct them and try again. This is why complex error checking is mostly done under ‘do end’ codes to display whatever error message they get. In BOARD, basic correctness is done as ‘require’ and ‘ensure’ to be sure that the code is running on the right track.

mark_on_board feature checks, if it’s argument coordination, has the right value. It checks if the value is positive, checks if the coordinates are under the board size and checks if the actual game is already started.

mark_fire and mark_bomb both follow the same procedure. They check if the value is positive, coordination is less than board size and game started. Then after the proper execution, it checks if shot or bomb count is incremented because no matter whether it was a hit or a miss, the bullet will be used anyway. It’s making sure if the used bullet is recorded properly on GAMEDATA.

paste_on_board is for pasting saved board into the current board. Undo and redo uses this to keep track of board status. It checks if targeting board is not empty because we never want to paste an empty board. Then it ensures that the target board and current board have the same items.

check_coord_is_hit and check_hit_caused_sink both checks if coordinates are reasonable as well as the number of ships are greater than 1. Since both features must go through an array of ships, this protection is needed.

check_win and check_lose both have preconditions of checking each other so that only one of them occurs at a time.

.

6. Summary of Testing Procedures

Test file	**Description**	**Passed**
at001.txt (instructor)	Test for basic fire and bomb commands.	✓
at002.txt (instructor)	Test for error cases of fire and bomb commands	✓
at101.txt (instructor)	Test for basic undo, redo and give_up commands	✓
at102.txt (instructor)	Test for combination of undo, redo, give_up with their own error cases.	✓
at01.txt	Test for error cases that might be created from custom_setup_test. Checks if arguments are not following the calculation as described in battleship.ui.txt	✓
at02.txt	Test for relationship between give_up command and game starting commands. Tests when some values are added to shots or score then give_up. Tests if give_up after undo.	✓
at03.txt	Test for relationship between error messages and undo, redo commands.	✓
at04.txt	Test for reinit ships generator. Check for conditions of big size of board templates	✓

7. Appendix (Contract view of all classes)

(Only classes that you created; do not include user input command classes, only model classes)

```
note
        description: "Display game board"
        author: "Taehoon Kim"
        date: "$Date$"
        revision: "$Revision$"

class interface
        BOARD

create
        make

feature -- call other functions
        history: HISTORY
        gamedata: GAMEDATA
        message: MESSAGE

feature -- get, set game starte variables
        get_started: BOOLEAN
        set_started
        set_not_started
        get_gameover: BOOLEAN
        set_gameover
                require
                        get_started

        set_not_gameover
        set_debugmode
        update_statenum (val: INTEGER_32)
                require
                        val >= 0

        get_numberofcommand: INTEGER_32

feature -- for undo, paste AFTER processed board to current board
        paste_on_board (imple: ARRAY2 [CHARACTER_8])
                require
                        not imple.is_empty
                ensure
                        implementation.same_items (imple)

feature -- query
-- Go through all ships any check any of them is hit
-- if sunk, return 'ship size'  <--- important!!
-- if just hit, return 0.
-- Because just hit and sink have different messages (used in ETF_FIRE)
```

```
        check_coord_is_hit (coord: COORD): INTEGER_32
                require
                        coord.x > 0 and coord.y > 0
                        not check_invalid_coord (coord)
                        gamedata.get_generated_ships.count > 0
                ensure
                        return_positive: Result >= 0

        check_ship_already_hit (coord: COORD): BOOLEAN
                require
                        coord.x > 0 and coord.y > 0
                        not check_invalid_coord (coord)

        check_ship_sunk (ship: TUPLE [size: INTEGER_32; row: INTEGER_32; col: INTEGER_32; dir:
BOOLEAN]): BOOLEAN
        check_hit_caused_sink (coord: COORD): BOOLEAN
                require
                        coord.x > 0 and coord.y > 0
                        not check_invalid_coord (coord)
                        gamedata.get_generated_ships.count > 0

        check_win: BOOLEAN
                require
                        not check_lose

        check_lose: BOOLEAN
                require
                        not check_win

feature -- query for error
        check_invalid_coord (coord: COORD): BOOLEAN
        check_already_fired (coord: COORD): BOOLEAN
        check_if_it_was_hit (coord: COORD): BOOLEAN
                require
                        coord.x > 0 and coord.y > 0
                        not check_invalid_coord (coord)

        check_fire_happened: BOOLEAN

feature -- query for display
        display_value_on_board (coord: COORD): CHARACTER_8
                require
                        coord.x > 0 and coord.y > 0
                        not check_invalid_coord (coord)

end -- class BOARD
```

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

class interface
        COORD

create
        make

feature
        x: INTEGER_32
        y: INTEGER_32
        valid (a_x, a_y: INTEGER_32): BOOLEAN
        debug_output: STRING_8
        out: STRING_8

end -- class COORD



note
        description: "[
        Just collection of game data. For example, default values of limits with
        different difficulties of the game.
        GAMEDATA is like a global that will be created when BOARD is make. Like HISTORY.
        Some Data like current_game and total score is placed in ETF_MODEL
        Ship generation occurs here but actual display will be done by 'implementation' in BOARD
        ]"
        author: ""
        date: "$Date$"
        revision: "$Revision$"

class interface
        GAMEDATA

create
        make

feature -- default values and messages
        row_chars: ARRAY [STRING_8]
        Game_mode_debug_test: STRING_8 = "debug_test"
        Game_mode_new_game: STRING_8 = "new_game"
        Game_mode_custom_setup: STRING_8 = "custom_setup"
        Game_mode_custom_setup_test: STRING_8 = "custom_setup_test"
        Easy_level_str: STRING_8 = "easy"
        Easy_level_int: INTEGER_32 = 13
        Easy_board_size: INTEGER_32 = 4
```

```
Easy_fire_limit: INTEGER_32 = 8
Easy_bomb_limit: INTEGER_32 = 2
Easy_score_limit: INTEGER_32 = 3
Easy_ships_limit: INTEGER_32 = 2
Medium_level_str: STRING_8 = "medium"
Medium_level_int: INTEGER_32 = 14
Medium_board_size: INTEGER_32 = 6
Medium_fire_limit: INTEGER_32 = 16
Medium_bomb_limit: INTEGER_32 = 3
Medium_score_limit: INTEGER_32 = 6
Medium_ships_limit: INTEGER_32 = 3
Hard_level_str: STRING_8 = "hard"
Hard_level_int: INTEGER_32 = 15
Hard_board_size: INTEGER_32 = 8
Hard_fire_limit: INTEGER_32 = 24
Hard_bomb_limit: INTEGER_32 = 5
Hard_score_limit: INTEGER_32 = 15
Hard_ships_limit: INTEGER_32 = 5
Advanced_level_str: STRING_8 = "advanced"
Advanced_level_int: INTEGER_32 = 16
Advanced_board_size: INTEGER_32 = 12
Advanced_fire_limit: INTEGER_32 = 40
Advanced_bomb_limit: INTEGER_32 = 7
Advanced_score_limit: INTEGER_32 = 28
Advanced_ships_limit: INTEGER_32 = 7
Msg_start_new: STRING_8 = "Start a new game"
Msg_fire_away: STRING_8 = "Fire Away!"
Msg_keep_fire: STRING_8 = "Keep Firing!"
Msg_hit: STRING_8 = "Hit!"
Msg_miss: STRING_8 = "Miss!"
Msg_win: STRING_8 = "You Win!"
Msg_game_over: STRING_8 = "Game Over!"
Err_ok: STRING_8 = "OK"
Err_game_already_started: STRING_8 = "Game already started"
Err_game_not_started: STRING_8 = "Game not started"
Err_invalid_coord: STRING_8 = "Invalid coordinate"
Err_already_fired_coord: STRING_8 = "Already fired there"
Err_no_shots: STRING_8 = "No shots remaining"
Err_no_bomb: STRING_8 = "No bombs remaining"
Err_adjacent_coord: STRING_8 = "Bomb coordinates must be adjacent"
Err_nothing_to_undo: STRING_8 = "Nothing to undo"
Err_nothing_to_redo: STRING_8 = "Nothing to redo"
Err_not_enough_ships: STRING_8 = "Not enough ships"
Err_too_many_ships: STRING_8 = "Too many ships"
Err_not_enough_shots: STRING_8 = "Not enough shots"
Err_too_many_shots: STRING_8 = "Too many shots"
Err_not_enough_bombs: STRING_8 = "Not enough bombs"
Err_too_many_bombs: STRING_8 = "Too many bombs"
Err_gave_up: STRING_8 = "You gave up!"
msg_ship_sunk (shipsize: INTEGER_32): STRING_8
msg_ships_sunk (shipsize1: INTEGER_32; shipsize2: INTEGER_32): STRING_8
```

```
feature -- update values
-- these updates are from 'MODEL' value to 'BOARD.GAMEDATA'
--                      see ETF_NEW_GAME.
            update_current_game (val: INTEGER_32)
            update_current_total_score (val: INTEGER_32)
            update_current_total_score_limit (val: INTEGER_32)
            add_score (shipsize: INTEGER_32)
            add_shot
            add_bomb
            add_ship
            sub_score (shipsize: INTEGER_32)
            sub_shot
            sub_bomb
            sub_ship
            update_shots (shots: INTEGER_32)
            update_bombs (bombs: INTEGER_32)
            update_ships (ships: INTEGER_32)
            update_score (score: INTEGER_32)
            update_total_score (score: INTEGER_32)

feature --query
-- Get series. Prevents client to modify gamedata values
            get_generated_ships: ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]]
            get_current_game: INTEGER_32
            get_current_level_int: INTEGER_32
            get_current_board_size: INTEGER_32
            get_current_fire: INTEGER_32
            get_current_bomb: INTEGER_32
            get_current_score: INTEGER_32
            get_current_total_score: INTEGER_32
            get_current_ships: INTEGER_32
            get_current_fire_limit: INTEGER_32
            get_current_bomb_limit: INTEGER_32
            get_current_score_limit: INTEGER_32
            get_current_total_score_limit: INTEGER_32
            get_current_ships_limit: INTEGER_32
            get_level_int (level_str: INTEGER_64): INTEGER_32
            get_board_size (level: INTEGER_32): INTEGER_32
            get_score_limit (level: INTEGER_32): INTEGER_32
            get_game_mode (custommode: BOOLEAN; debugmode: BOOLEAN): STRING_8
            get_ship_limit (level: INTEGER_32): INTEGER_32

feature --command
            set_game_default_value (level: INTEGER_32)
            set_game_default_value_custom (dimension: INTEGER_32; ships: INTEGER_32; max_shots:
INTEGER_32; num_bombs: INTEGER_32)
            set_generated_ships (gs: ARRAYED_LIST [TUPLE [size: INTEGER_32; row: INTEGER_32; col:
INTEGER_32; dir: BOOLEAN]])
            test_ships_generated
                        -- test random generation of ships

end -- class GAMEDATA
```

```
note
	description: "Summary description for {HISTORY}."
	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?

feature -- comands
	extend_history (a_op: OPERATION)
		ensure
			history [history.count] = a_op

	remove_right
			--remove all elements
			-- to the right of the current cursor in history
	remove_all
	forth
		require
			not after

	back
		require
			not before

	display_all
			-- only for testing. See all contents
	display_cursor_statenum

end -- class HISTORY
```

```
note
	description: "[
	-- Message has a form of
	-- number of command(integer), status(OK) or error, command status(could be more than 1)
	-- ex) state 1 OK -> Fire Away!
	-- ex) state 11 Game already started -> Keep Firing!
	-- ex) state 9 OK -> 4x1 and 3x1 ships sunk! Keep Firing!
	-- ex) state 12 OK -> 2x1 ship sunk! Keep Firing!
	]"
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	MESSAGE

create
	make

feature
	make (level: INTEGER_32; debug_mode: BOOLEAN)

feature
	set_msg_error (a_message: STRING_8)
	set_msg_error_reference (a_message: STRING_8)
	set_msg_command (a_message: STRING_8)
	get_msg_numofcmd (numofcmd: INTEGER_32): STRING_8
	get_msg_error: STRING_8
	get_msg_error_reference: STRING_8
			-- only for undo, redo
	get_msg_command: STRING_8
		require
			get_msg_command.count > 1

	clear_msg_command
		ensure
			Current.get_msg_command.is_empty

	clear_msg_error_reference

end -- class MESSAGE
```

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

deferred class interface
	OPERATION

feature -- deferred query
	get_msg_error: STRING_8
	get_msg_command: STRING_8
	get_statenum: INTEGER_32
	get_op_name: STRING_8
	get_implementation: ARRAY2 [CHARACTER_8]

feature -- deferred commands
	execute
	undo
	redo

end -- class OPERATION
```