EECS3311-W2020 — Project Report

Submitted electronically by:

Team members	Name	Prism Login	Student ID
Member – 1 :	Jaiveer Singh	jaiveer	214967087
Member – 2 :	Muhammad Arifur Rahman	arif36ca	215858855
*Submitted under Prism account:	jaiveer		

Contents

1. Requirements for Project "SimOdyssey"

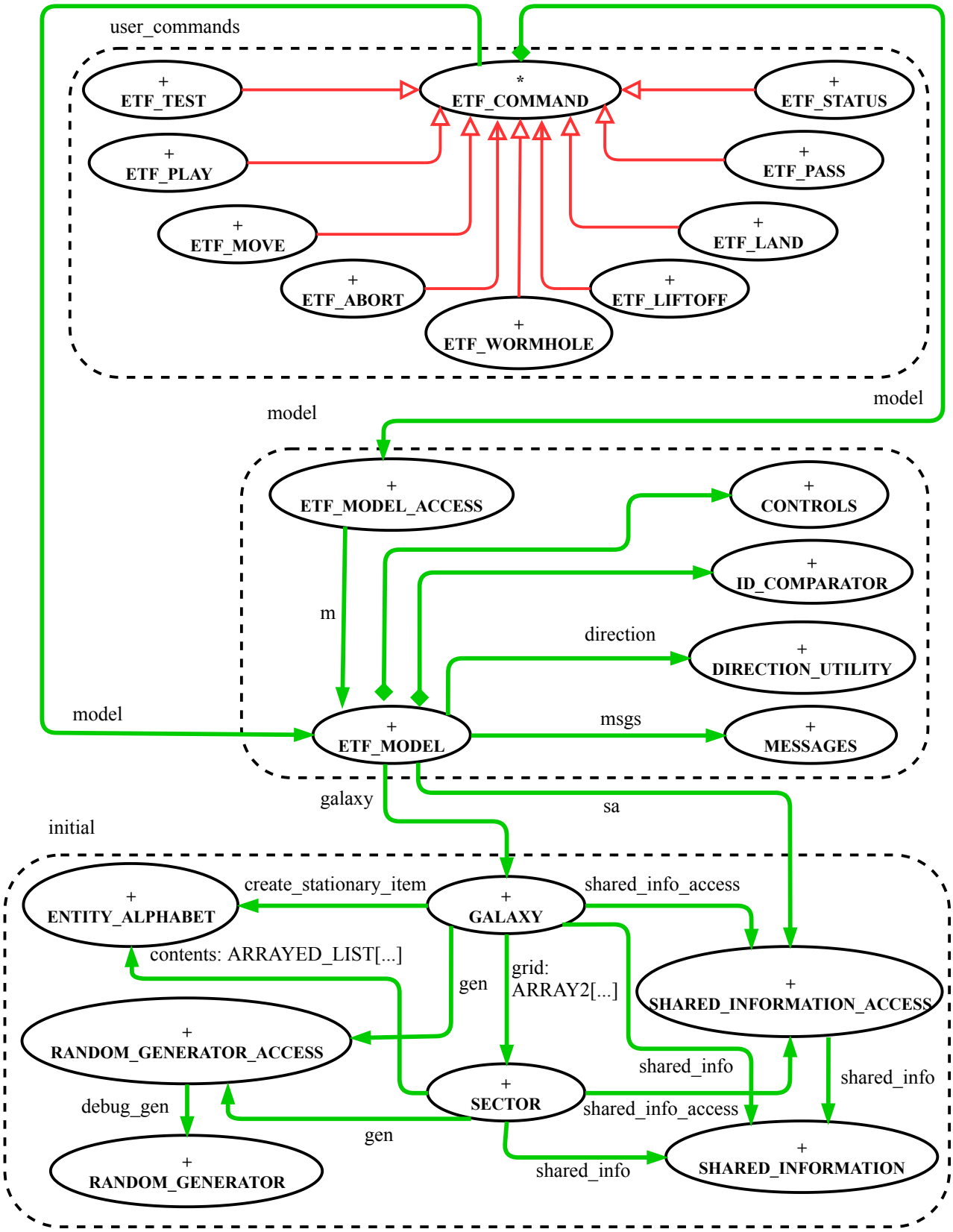
The system to be modeled and implemented is a simplified simulation of a galaxy. A two-dimensional grid of sectors represents the galaxy. The size of the grid is 5 by 5. Each sector in the grid is identified by its coordinates in terms of the row number and the column number. Entities are randomly allocated to sectors in the grid, except for the explorer and blackhole (the explorer is in the coordinate (1, 1) initially and the blackhole is fixed in the coordinate (3, 3)). For the purpose of moving the explorer, if the explorer is in any sector, it can travel to any of the 8 adjacent sectors normally. These are found in the north, north-east, east, south-east, south, south-west, west, and north-west positions directly adjacent to the given sector. The grid wraps along its boundaries meaning if we go north from a sector in the first row, we will move into the fifth row at the bottom of the grid.

Movable entities are Explorer (E), Asteroid (A), Benign (B), Planet (P), Malevolent (M) and Janitaur (J). They can move through the GALAXY and interact with other entities. Stationary entities are Wormhole (W), Blackhole (O), Blue Giant (*) and Yellow Dwarf (Y). They stay in one place and interact with the movable entities. The explorer is a movable entity controlled by the user. After the generation of the board, commands can be issued to control the behavior of the explorer. Some of the commands will constitute as a turn (such as moving the explorer) which will subsequently cause some of the other movable entities to make their move. Other commands (such as checking the status) will not modify the board. The system shall output the current abstract state of the game and a table representation of the galaxy.

The explorer's mission is to see if such starts have any planets orbiting them. If a planet is discovered, the explorer can land on the planet and conduct experiments to determine if life is supportable. The game is won when a planet capable of supporting life is discovered. The game is continued until either the explorer's life runs out, the explorer's fuel runs out, a planet with life is found, or the game is aborted. A new game can be started when the game is over.

Appropriate command-specific messages should be displayed after each command, invalid commands should give error messages and death messages should be shown if an entity is dead after executing the command. They are described in 'simodyssey2-messages.txt' file. Also, refer to the 'simodyssey2.definitions.txt' file to find more about commands and their conditions.

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



In the architecture of the design, we tried to keep it as generic as possible. The singleton pattern is used, where each command used by the classes of user_commands cluster rely on model attribute from ETF_COMMAND. Then the model is used for accessing different modules. It gives the flexibility to the software to accommodate any additional changes, or any additional functionality without making any changes to the design.

Class ETF_MODEL is where the basic game environment is set up. To prevent this to be a superman class, we have separated roles by creating additional classes such as CONTROLS and MESSAGES. ETF_MODEL class keeps track of basic game status such as whether the game is started, whether the move is valid, whether the Explorer is landed on planet, whether the planet supports life or whether the Explorer is dead. This class holds all the commands of model operations (such as play, test, move etc.). We decided to keep all these commands in this class to make other classes much simpler and easier to reuse the same execution code for other classes. Finally, this class displays all the required output such as printing out appropriate error messages, descriptions, game status and the board.

The class CONTROLS is where the game is controlled throughout the execution of different commands during playing. This class fills up the galaxy (board) with appropriate entities, controls the movement, behavior, or reproduction of all moveable entities, checks whether a moveable entity is alive. This class also provides all the necessary information to print out descriptions and sectors as strings. This class uses the class RANDOM_GENERATOR to generate random number which determines the move of a movable entity between sectors.

The class MESSAGES is created to set all the relevant messages according to the execution of the commands in each level of playing. All the messages (regular, error or death messages) are accessed by the ETF_MODEL which helps to output desired output strings.

The class DIRECTION_UTILITY converts an integer encoding to a direction which is used to move the movable entities in that direction. The ETF_MODEL class uses this module in the ‘move’ feature to ensure movement of movable entities in the correct directions.

3. Table of modules – responsibilities and secrets

1	ETF_MODEL	**Responsibility:** Set up the board, perform model operations and output appropriate strings of the board states.	**Alternate:** None
	Concrete	**Secret:** None	

2	CONTROLS	**Responsibility:** Control the filling of sectors, movement, or behavior of different entities of the galaxy. Check if the explorer is alive and print sectors and descriptions.	**Alternate:** None
	Concrete	**Secret:** None	

3	DIRECTION_UTILITY	**Responsibility:** Set up and calculate the direction to move the movable entities.	**Alternate:** None
	Concrete	**Secret:** None	

4	MESSAGES	**Responsibility:** Set up regular, error or death messages and output appropriate messages.	**Alternate:** None
	Concrete	**Secret:** None	

5	ID_COMPARATOR	**Responsibility:** Compare ids between entities.	**Alternate:** None
	Abstract	**Secret:** None	

6	GALAXY	**Responsibility:** Set up grids with appropriate entities and output grids in string form.	**Alternate:** None
	Concrete	**Secret:** None	

7	SECTOR	**Responsibility:** Set up quadrants of each sectors with appropriate entities. Get information if a quadrant has a particular entity.	**Alternate:** None
	Concrete	**Secret:** Implemented via ENTITY_ALPHABET.	

7.1	ENTITY_ALPHABET	**Responsibility:** Set up properties of different entities and get the types of entities.	**Alternate:** None
	Concrete	**Secret:** None	

8	SHARED_INFORMATION	**Responsibility:** Set up of thresholds of different entities.	**Alternate:** None
	Abstract	**Secret:** None	

9	RANDOM_GENERATOR	**Responsibility:** Generate random numbers.	**Alternate:** None
	Abstract	**Secret:** None	

4. Expanded description of design decisions

The most important module of the simodyssey project is CONTROLS. The reason the CONTROLS module is the most significant is because all operations performed by the simodyssey are somewhat linked to the CONTROLS module.

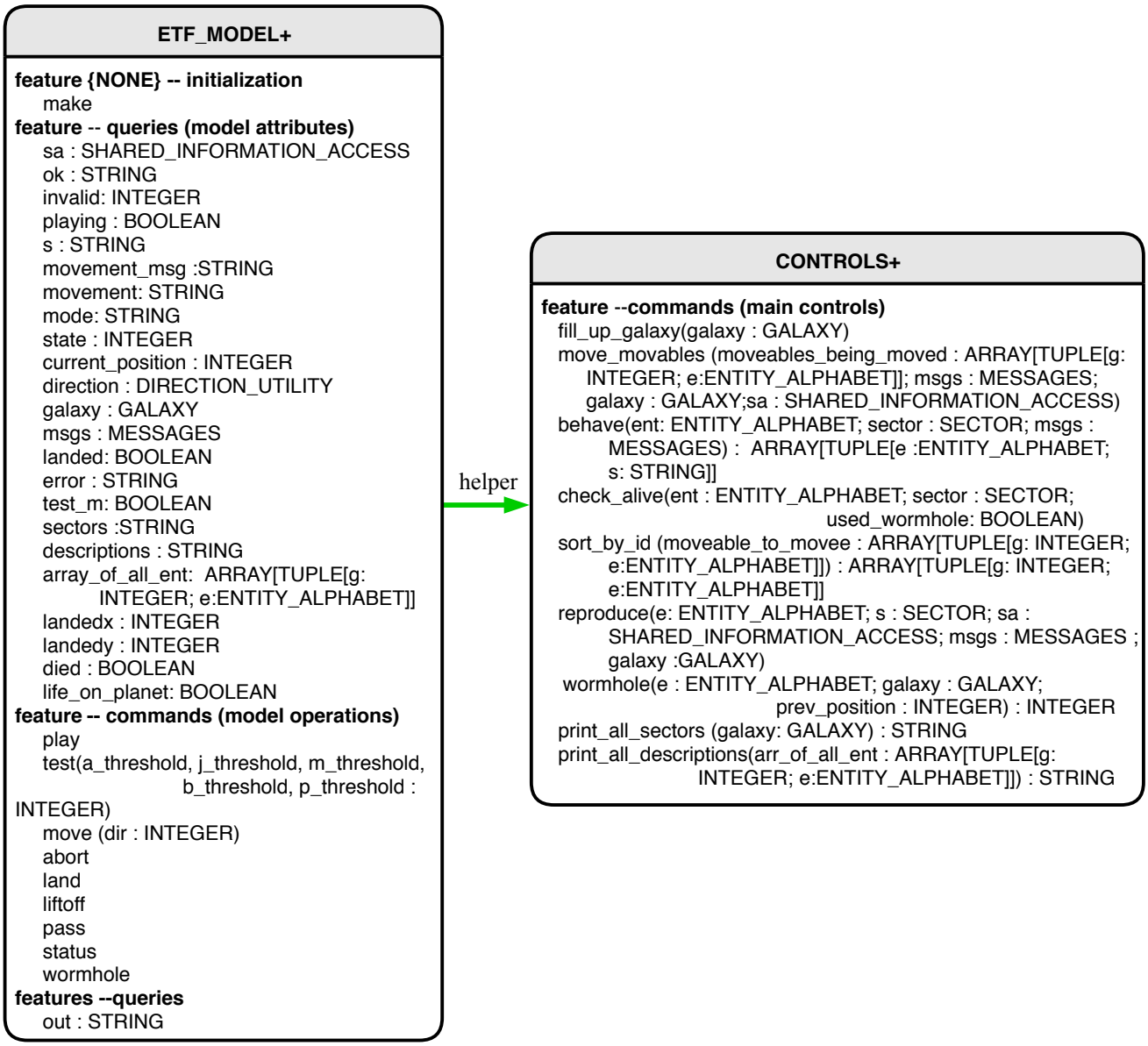
CONTROLS is responsible for moving movable entities to their next correct position, it does so by using another module called RANDOM_GENERATOR which generates a random number between a desired range of numbers. Furthermore, this random number is only generated when a moveable entity must be moved from its current position. There are many different moveable entities and these moveable entities can move in many ways. The CONTROLS module is the class that is responsible for moving them to their next correct position in the most suitable way possible. Whether it be through regular movements (i.e. moving the entity to an adjacent quadrant in a random manner) or with the use of a wormhole (which places the entity in a random sector in the galaxy).

The first feature of the CONTROLS module is **fill_up_galaxy**, this feature is intended to be used every time a new game is started. It is intended to fill the galaxy (i.e. all its sectors) with a filler entity so that all sectors are full. By filling all sectors up this feature allows for the movement of entities to be easier by allowing entities to replace only filler entities (this is done by using a feature in the SECTOR module which finds the **next_available_position** that returns the position of the first filler entity in the quadrant). This is important because the random number generator tells this feature which sector to move the entity to but not the position in the sector to which it should be moved and that is made possible in part because of this feature.

The feature **move_moveables** is responsible for moving moveable entities to their next position if and only if they must be moved (i.e. if their turns left = 0 and if the entity is not attached to a star). It also uses the other features in this module to correctly manage the behavior of each entity and to manage if it is still alive after a movement is made. It also uses the **sort_by_id** feature which helps sort the moveable entities by their id. So that the moveable entities are always moved in the correct order (i.e. giving priority to the movement of entities that have a lower id than those with a higher one).

The **print_all_sector** and **print_all_descriptions** features are used by the **move_moveable** only when the mode of playing the game is test mode as it can used after every movement to display the correct state of the game to the user.

Finally, the **wormhole** feature is also used when the correct moveable entities interacts with a wormhole (malevolent or benign entities) or when the explorer uses a wormhole. This is an important feature as it is one of the few ways in which an entity can move. There is also a **reproduce** feature which is used by only malevolent/benign/janitaur entities. This feature creates another of the same kind of entity with a new id and places it in the same sector as the entity which reproduced, but only if it's the turn of the entity to reproduce and if there is a position available in the quadrant for the new entity to occupy.



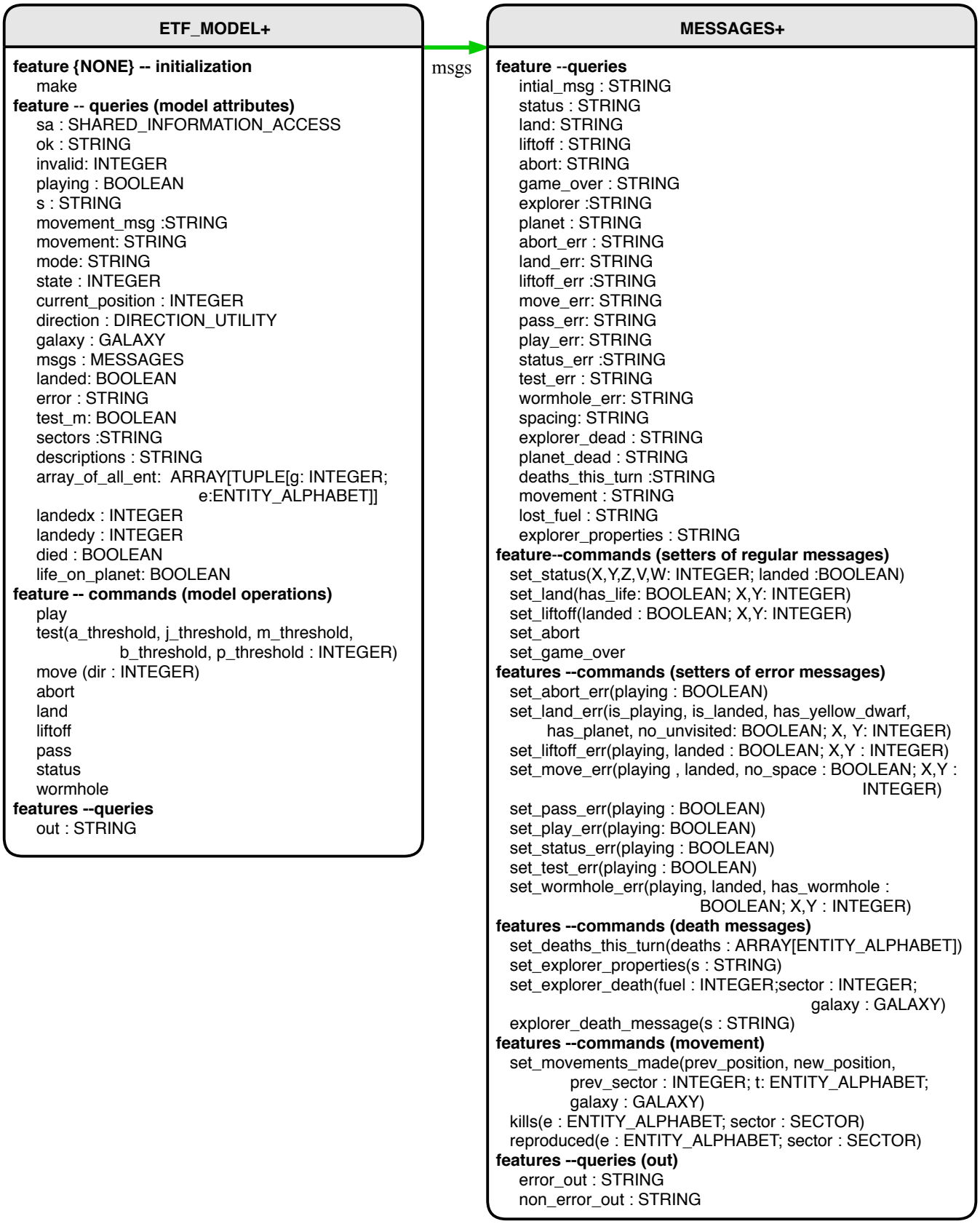
As you can see the only module which uses the CONTROLS module is ETF_MODEL. That is because all the required information from other modules of the project are fed to the CONTROLS module through the ETF_MODEL. This is to ensure that these modules are not created multiple times which could potentially cause logical errors to occur in the code. Also, so that essential information to perform the tasks of all the features of the CONTROLS module are only used only when they are needed. We could have created instances of the other modules in CONTROLS, but we believed it would be better to create them in the ETF_MODEL and feed them to the features whenever needed. So that the implementations are simpler (i.e. do not have to create classes every time they are needed, we can just reuse the same instance and modify it as changes are made in the game). We also believed that this design is more maintainable and reliable as there are less sources of error if any changes are made to the other features being used.

5. Significant Contracts (Correctness)

One of the most important modules in terms of correctness is the MESSAGES module. The messages module is important for the project as it helps with error checking in the ETF_MODEL. It does so by setting a STRING within the module with an error message if an error has occurred and leaving the STRING empty if no error has occurred. This can ensure the correctness of the features in the ETF_MODEL as before each feature performs any actions within, it first feeds certain information into a feature in the MESSAGES module and this module checks if the current state of the game is an error state (which makes the error string non-empty) or one that is valid for the game to proceed(keeps error string as empty).

Then in the ETF_MODEL after correct feature from the MESSAGES module is executed it checks if the error string in the MESSAGES module is empty or not. If it is empty the feature can proceed (meaning the state of the game is sufficient to ensure that the feature will have the correct outcome) and if not, the feature doesn't do what it is intended to do, and the game outputs the error messages generated by the MESSAGES module.

This module can be very significant in ensuring that certain situations exist before a feature can correctly do what it is meant to do. The MESSAGES module not only ensures the correct state for the actions performed in the features used in the ETF_MODEL it also ensures that the correct messages are outputting when different situations occur while playing the simodyssey game.


ETF_MODEL+
feature {NONE} -- initialization
make
feature -- queries (model attributes)
sa : SHARED_INFORMATION_ACCESS
ok : STRING
invalid: INTEGER
playing : BOOLEAN
s : STRING
movement_msg :STRING
movement: STRING
mode: STRING
state : INTEGER
current_position : INTEGER
direction : DIRECTION_UTILITY
galaxy : GALAXY
msgs : MESSAGES
landed: BOOLEAN
error : STRING
test_m: BOOLEAN
sectors :STRING
descriptions : STRING
array_of_all_ent: ARRAY[TUPLE[g: INTEGER;
e:ENTITY_ALPHABET]]
landedx : INTEGER
landedy : INTEGER
died : BOOLEAN
life_on_planet: BOOLEAN
feature -- commands (model operations)
play
test(a_threshold, j_threshold, m_threshold,
b_threshold, p_threshold : INTEGER)
move (dir : INTEGER)
abort
land
liftoff
pass
status
wormhole
features --queries
out : STRING
msgs
MESSAGES+
feature --queries
intial_msg : STRING
status : STRING
land: STRING
liftoff : STRING
abort: STRING
game_over : STRING
explorer :STRING
planet : STRING
abort_err : STRING
land_err: STRING
liftoff_err :STRING
move_err: STRING
pass_err: STRING
play_err: STRING
status_err :STRING
test_err : STRING
wormhole_err: STRING
spacing: STRING
explorer_dead : STRING
planet_dead : STRING
deaths_this_turn :STRING
movement : STRING
lost_fuel : STRING
explorer_properties : STRING
feature--commands (setters of regular messages)
set_status(X,Y,Z,V,W: INTEGER; landed :BOOLEAN)
set_land(has_life: BOOLEAN; X,Y: INTEGER)
set_liftoff(landed : BOOLEAN; X,Y: INTEGER)
set_abort
set_game_over
features --commands (setters of error messages)
set_abort_err(playing : BOOLEAN)
set_land_err(is_playing, is_landed, has_yellow_dwarf,
has_planet, no_unvisited: BOOLEAN; X, Y: INTEGER)
set_liftoff_err(playing, landed : BOOLEAN; X,Y : INTEGER)
set_move_err(playing , landed, no_space : BOOLEAN; X,Y :
INTEGER)
set_pass_err(playing : BOOLEAN)
set_play_err(playing: BOOLEAN)
set_status_err(playing : BOOLEAN)
set_test_err(playing : BOOLEAN)
set_wormhole_err(playing, landed, has_wormhole :
BOOLEAN; X,Y : INTEGER)
features --commands (death messages)
set_deaths_this_turn(deaths : ARRAY[ENTITY_ALPHABET])
set_explorer_properties(s : STRING)
set_explorer_death(fuel : INTEGER;sector : INTEGER;
galaxy : GALAXY)
explorer_death_message(s : STRING)
features --commands (movement)
set_movements_made(prev_position, new_position,
prev_sector : INTEGER; t: ENTITY_ALPHABET;
galaxy : GALAXY)
kills(e : ENTITY_ALPHABET; sector : SECTOR)
reproduced(e : ENTITY_ALPHABET; sector : SECTOR)
features --queries (out)
error_out : STRING
non_error_out : STRING

6. Summary of Testing Procedures

Test file	Description	Passed
	(instructor's acceptance tests)	
at001.txt	Tests winning condition in test mode.	✓
at002.txt	Tests winning condition in play mode.	✓
at003.txt	Tests losing condition in test mode (lose by out of life).	✓
	(student's acceptance tests)	
at001.txt	Tests basic error checking before starting 'play' or 'test'.	✓
at002.txt	Tests losing condition in test mode (Explorer got devoured by blackhole).	✓
at003.txt	Tests error in moving the Explorer as all quadrants of a sector are full.	✓
at004.txt	Tests error in attempting 'play' or 'test' command while the game is already running.	✓
at005.txt	Tests error in 'land' when no yellow dwarf or planets present at the current sector.	✓
at006.txt	Tests error in 'wormhole' when no wormhole present at the current sector.	✓
at007.txt	Tests error in 'liftoff' when the Explorer is not landed on a planet and when the game is over.	✓
at008.txt	Tests losing condition in play mode (lose by out of fuel).	✓
at009.txt	Tests losing condition in test mode (lose by out of fuel).	✓
at010.txt	Tests losing condition in test mode (Explorer got destroyed by asteroid).	✓
: :	: :	: :
at073.txt	Tests the explorer's interaction with janitaur and benign when they are in the same sector and check if asteroids can destroy explorers while it is landed	✓
at075.txt	Tests case where the explorer gets destroyed by an asteroid in test mode and tests the winning case of explorer in play mode in a new game	✓

Following is a screenshot of the Espec unit tests that we ran in Eiffel Studio:

Test Run:04/06/2020 3:10:39.033 PM

ROOT

Note: * indicates a violation test case

PASSED (10 out of 10)

Case Type	Passed	Total
Violation	0	0
Boolean	10	10
All Cases	10	10

State	Contract Violation	Test Name
Test1	TEST_GALAXY	
PASSED	NONE	t1: Tests basic error checking before starting 'play' or 'test'.
PASSED	NONE	t2: Tests losing condition in test mode (Explorer got devoured by blackhole).
PASSED	NONE	t3: Tests error in moving the Explorer as all quadrants of a particular sector is full.
PASSED	NONE	t4: Tests error in attempting 'play' or 'test' command while the game is already running.
PASSED	NONE	t5: Tests error in 'land' when no yellow dwarf or planets present at the current sector.
PASSED	NONE	t6: Tests error in 'wormhole' when no wormhole present at the current sector.
PASSED	NONE	t7: Tests error in 'liftoff' when the Explorer is not landed on a planet and when the game is over.
PASSED	NONE	t8: Tests losing condition in play mode (lose by out of fuel).
PASSED	NONE	t9: Tests losing condition in test mode (lose by out of fuel).
PASSED	NONE	t10: Tests losing condition in test mode (Explorer got destroyed by asteroid).

Following is a screenshot of the regression tests passing:

user@localhost:regression-testing

File Edit View Search Terminal Help

```
Success: log/student/at045.actual.txt and log/student/at045.expected.txt are identical.
Success: log/student/at046.actual.txt and log/student/at046.expected.txt are identical.
Success: log/student/at047.actual.txt and log/student/at047.expected.txt are identical.
Success: log/student/at048.actual.txt and log/student/at048.expected.txt are identical.
Success: log/student/at049.actual.txt and log/student/at049.expected.txt are identical.
Success: log/student/at050.actual.txt and log/student/at050.expected.txt are identical.
Success: log/student/at051.actual.txt and log/student/at051.expected.txt are identical.
Success: log/student/at052.actual.txt and log/student/at052.expected.txt are identical.
Success: log/student/at053.actual.txt and log/student/at053.expected.txt are identical.
Success: log/student/at054.actual.txt and log/student/at054.expected.txt are identical.
Success: log/student/at055.actual.txt and log/student/at055.expected.txt are identical.
Success: log/student/at056.actual.txt and log/student/at056.expected.txt are identical.
Success: log/student/at057.actual.txt and log/student/at057.expected.txt are identical.
Success: log/student/at058.actual.txt and log/student/at058.expected.txt are identical.
Success: log/student/at059.actual.txt and log/student/at059.expected.txt are identical.
Success: log/student/at060.actual.txt and log/student/at060.expected.txt are identical.
Success: log/student/at061.actual.txt and log/student/at061.expected.txt are identical.
Success: log/student/at062.actual.txt and log/student/at062.expected.txt are identical.
Success: log/student/at063.actual.txt and log/student/at063.expected.txt are identical.
Success: log/student/at064.actual.txt and log/student/at064.expected.txt are identical.
Success: log/student/at065.actual.txt and log/student/at065.expected.txt are identical.
Success: log/student/at066.actual.txt and log/student/at066.expected.txt are identical.
Success: log/student/at067.actual.txt and log/student/at067.expected.txt are identical.
Success: log/student/at068.actual.txt and log/student/at068.expected.txt are identical.
Success: log/student/at069.actual.txt and log/student/at069.expected.txt are identical.
Success: log/student/at070.actual.txt and log/student/at070.expected.txt are identical.
Success: log/student/at071.actual.txt and log/student/at071.expected.txt are identical.
Success: log/student/at072.actual.txt and log/student/at072.expected.txt are identical.
Success: log/student/at073.actual.txt and log/student/at073.expected.txt are identical.
Success: log/student/at074.actual.txt and log/student/at074.expected.txt are identical.
Success: log/student/at075.actual.txt and log/student/at075.expected.txt are identical.
==========================
Test Results: 75/75 passed.
```

7. Appendix (Contract view of all classes, i.e. their *specification)*

Class 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 attributes

	sa: SHARED_INFORMATION_ACCESS
	ok: STRING_8
	invalid: INTEGER_32
	playing: BOOLEAN
	s: STRING_8
	movement_msg: STRING_8
	movement: STRING_8
	mode: STRING_8
	state: INTEGER_32
	current_position: INTEGER_32
	direction: DIRECTION_UTILITY
	galaxy: GALAXY
	msgs: MESSAGES
	landed: BOOLEAN
	error: STRING_8
	test_m: BOOLEAN
	sectors: STRING_8
	descriptions: STRING_8
	array_of_all_ent: ARRAY [TUPLE [g: INTEGER_32; e: ENTITY_ALPHABET]]
	landedx: INTEGER_32
	landedy: INTEGER_32
	died: BOOLEAN
	life_on_planet: BOOLEAN

feature -- model operations

	play
	move (dir: INTEGER_32)
	abort
	land
	liftoff
	pass
	status
	test (a_threshold, j_threshold, m_threshold, b_threshold, p_threshold: INTEGER_32)
	wormhole
	reset
			-- Reset model state.

feature -- queries

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

end -- class ETF_MODEL
```

Class MESSAGES:

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

class interface
	MESSAGES

create
	make

feature --constructor
	make

feature --queries

	intial_msg: STRING_8
	status: STRING_8
	land: STRING_8
	liftoff: STRING_8
	abort: STRING_8
	game_over: STRING_8
	explorer: STRING_8
	planet: STRING_8
	abort_err: STRING_8
	land_err: STRING_8
	liftoff_err: STRING_8
	move_err: STRING_8
	pass_err: STRING_8
	play_err: STRING_8
	status_err: STRING_8
	test_err: STRING_8
	wormhole_err: STRING_8
	spacing: STRING_8
	explorer_dead: STRING_8
	planet_dead: STRING_8
	deaths_this_turn: STRING_8
	movement: STRING_8
	lost_fuel: STRING_8
	explorer_properties: STRING_8

feature --setters of regular messages

	set_status (x, y, z, v, w: INTEGER_32; landed: BOOLEAN)
	set_land (has_life: BOOLEAN; x, y: INTEGER_32)
	set_liftoff (landed: BOOLEAN; x, y: INTEGER_32)
	set_abort
	set_game_over

feature --setters of error messages

	set_abort_err (playing: BOOLEAN)
	set_land_err (is_playing, is_landed, has_yellow_dwarf, has_planet, no_unvisited: BOOLEAN;
	                                                                    x, y: INTEGER_32)
	                --sector x, y
	set_liftoff_err (playing, landed: BOOLEAN; x, y: INTEGER_32)
	                --sector x, y
	set_move_err (playing, landed, no_space: BOOLEAN; x, y: INTEGER_32)
	set_pass_err (playing: BOOLEAN)
	set_play_err (playing: BOOLEAN)
	set_status_err (playing: BOOLEAN)
	set_test_err (playing: BOOLEAN)
	set_wormhole_err (playing, landed, has_wormhole: BOOLEAN; x, y: INTEGER_32)

feature --death messages

	set_deaths_this_turn (deaths: ARRAY [TUPLE [e: ENTITY_ALPHABET; s: STRING_8]])
```

```
        set_explorer_properties (s: STRING_8)
        set_explorer_death (fuel: INTEGER_32; sector: INTEGER_32; galaxy: GALAXY)
        explorer_death_message (s: STRING_8)

feature --movement

        set_movements_made (prev_position, new_position, prev_sector: INTEGER_32; t:
                                                    ENTITY_ALPHABET; galaxy: GALAXY)

        kills (e: ENTITY_ALPHABET; sector: SECTOR)
        reproduced (e: ENTITY_ALPHABET; sector: SECTOR)

feature --out

        error_out: STRING_8
        non_error_out: STRING_8

end -- class MESSAGES
```

Class DIRECTION_UTILITY:

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

expanded class interface
        DIRECTION_UTILITY

create
        default_create

feature -- Queries

        N: INTEGER_32
                        -- Tuple modifier for North
                        -- move up one row (1)
        E: INTEGER_32
                        -- Tuple modifier for East
        S: INTEGER_32
                        -- Tuple modifier for South
        W: INTEGER_32
                        -- Tuple modifier for West
        Ne: INTEGER_32
                        -- Tuple modifier for North East
                        -- move up one row (1)
        Nw: INTEGER_32
                        -- Tuple modifier for North West
        Se: INTEGER_32
                        -- Tuple modifier for South East
        Sw: INTEGER_32
                        -- Tuple modifier for South West
        Dir_arr: ARRAY [INTEGER_32]
                        -- Array of each of the cardinal direction modifiers
        cal_new (curr: INTEGER_32; rows: INTEGER_32; col: INTEGER_32; move: INTEGER_32):
                                                                        INTEGER_32
        num_dir (int: INTEGER_32): INTEGER_32
                        -- Convert an integer encoding to a direction.

end -- class DIRECTION_UTILITY
```

Class CONTROLS:

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

expanded class interface
	CONTROLS

create
	default_create

feature -- main controls

	fill_up_galaxy (galaxy: GALAXY)
	move_movables (moveables_being_moved: ARRAY [TUPLE [g: INTEGER_32; e: ENTITY_ALPHABET]];
	                msgs: MESSAGES; galaxy: GALAXY; sa: SHARED_INFORMATION_ACCESS)
	behave (ent: ENTITY_ALPHABET; sector: SECTOR; msgs: MESSAGES): ARRAY [TUPLE [e:
	                                                    ENTITY_ALPHABET; s: STRING_8]]
	check_alive (ent: ENTITY_ALPHABET; sector: SECTOR; used_wormhole: BOOLEAN)
	                --only use after moving a planet and explorer
	sort_by_id (moveable_to_movee: ARRAY [TUPLE [g: INTEGER_32; e: ENTITY_ALPHABET]]): ARRAY
	                                    [TUPLE [g: INTEGER_32; e: ENTITY_ALPHABET]]
	reproduce (e: ENTITY_ALPHABET; s: SECTOR; sa: SHARED_INFORMATION_ACCESS; msgs: MESSAGES;
	                                                                    galaxy: GALAXY)
	                --some entities may reproduce as turns pass
	wormhole (e: ENTITY_ALPHABET; galaxy: GALAXY; prev_position: INTEGER_32): INTEGER_32
	print_all_sectors (galaxy: GALAXY): STRING_8
	                -- tells you what's in each sector
	print_all_descriptions (arr_of_all_ent: ARRAY [TUPLE [g: INTEGER_32; e:
	                                                    ENTITY_ALPHABET]]): STRING_8
	                --give arr of all entities in graph returns print out of all descriptions

end -- class CONTROLS
```

Class ID_COMPARATOR:

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

class interface
	ID_COMPARATOR

create
	default_create

feature

	attached_less_than (e1, e2: attached TUPLE [g: INTEGER_32; e: ENTITY_ALPHABET]): BOOLEAN
	                -- effect e1 < e2

end -- class ID_COMPARATOR
```

Class ETF_MODEL_ACCESS:

```
note
	description: "Singleton access to the default business model."
	author: "Jackie Wang"
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	ETF_MODEL_ACCESS

create
	default_create

feature

	M: ETF_MODEL

invariant
		M = M

end -- class ETF_MODEL_ACCESS
```

Class GALAXY:

```
note
	description: "Galaxy represents a game board in simodyssey."
	author: "Kevin B"
	date: "$Date$"
	revision: "$Revision$"

class interface
	GALAXY

create
	make,
	make_empty

feature -- attributes

	grid: ARRAY2 [SECTOR]
			-- the board
	gen: RANDOM_GENERATOR_ACCESS
	shared_info_access: SHARED_INFORMATION_ACCESS
	shared_info: SHARED_INFORMATION

feature --constructor

	make
		-- creates a dummy of galaxy grid
feature --commands

	make_empty
	set_stationary_items
		-- distribute stationary items amongst the sectors in the grid.
		-- There can be only one stationary item in a sector
	create_stationary_item: ENTITY_ALPHABET
		-- this feature randomly creates one of the possible types of stationary actors

feature -- query

	out: STRING_8
			--Returns grid in string form

end -- class GALAXY
```

Class SECTOR:

```
note
	description: "Represents a sector in the galaxy."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	SECTOR

create
	make,
	make_dummy

feature -- attributes

	shared_info_access: SHARED_INFORMATION_ACCESS
	shared_info: SHARED_INFORMATION
	gen: RANDOM_GENERATOR_ACCESS
	contents: ARRAYED_LIST [ENTITY_ALPHABET]
			--holds 4 quadrants
	row: INTEGER_32
	column: INTEGER_32

feature -- constructor

	make (row_input: INTEGER_32; column_input: INTEGER_32; a_explorer: ENTITY_ALPHABET)
			--initialization
		require
			valid_row: (row_input >= 1) and (row_input <= shared_info.Number_rows)
			valid_column: (column_input >= 1) and (column_input <=
                                                shared_info.Number_columns)

feature -- commands

	make_dummy
			--initialization without creating entities in quadrants
	populate
	-- this feature creates 1 to max_capacity-1 components to be initially stored in the
	-- sector. The component may be a planet or nothing at all.

feature -- Queries

	print_sector: STRING_8
			-- Printable version of location's coordinates with different formatting
	is_full: BOOLEAN
			-- Is the location currently full?
	has_stationary: BOOLEAN
			-- returns whether the location contains any stationary item
	has_planet: BOOLEAN
	has_wormhole: BOOLEAN
	has_yellow_dwarf: BOOLEAN
	has_blue_gaint: BOOLEAN
	has_star: BOOLEAN
	has_blackhole: BOOLEAN
	has_explorer: BOOLEAN
	has_benign: BOOLEAN
	next_available_quad: INTEGER_32
			-- return the left-most next available quadrant
			-- require: sector not full

end -- class SECTOR
```

Class ENTITY_ALPHABET:

```
note
	description: "[
		Alphabet allowed to appear on the galaxy board.
	]"
	author: "Kevin Banh"
	date: "April 30, 2019"
	revision: "1"

class interface
	ENTITY_ALPHABET

create
	make,
	make_empty

feature -- Constructor

	make (a_char: CHARACTER_8)

feature -- Attributes

	item: CHARACTER_8
	turns_left: INTEGER_32
	id: INTEGER_32
	sector_pos: INTEGER_32
	is_attached: BOOLEAN
	supports_life: BOOLEAN
	visited: BOOLEAN
	luminosity: INTEGER_32
	fuel: INTEGER_32
	life: INTEGER_32
	dies: BOOLEAN
	landed: BOOLEAN
	actions_left_until_reproduction: INTEGER_32
	max_fuel: INTEGER_32
	load: INTEGER_32

feature -- Query

	make_empty
	out: STRING_8
			-- Return string representation of alphabet.
	is_equal (other: ENTITY_ALPHABET): BOOLEAN
			-- Is other attached to an object considered
			-- equal to current object?
	is_stationary: BOOLEAN
			-- Return if current item is stationary.
	is_moveable: BOOLEAN
	is_wormhole: BOOLEAN
	is_yellow_dwarf: BOOLEAN
	is_blue_gaint: BOOLEAN
	is_planet: BOOLEAN
	is_explorer: BOOLEAN
	is_star: BOOLEAN
	is_blackhole: BOOLEAN
	is_benign: BOOLEAN
	is_malevolent: BOOLEAN
	is_janitaur: BOOLEAN
	is_asteroid: BOOLEAN
	is_filler: BOOLEAN
	set_actions_left_until_reproduction (i: INTEGER_32)
	set_turns_left (i: INTEGER_32)
	set_id (i: INTEGER_32)
	set_sector_pos (g: GALAXY)
	set_luminosity (i: INTEGER_32)
	set_attached (b: BOOLEAN)
	set_supports_life (b: BOOLEAN)
	set_visited (b: BOOLEAN)
```

```
        set_landed (b: BOOLEAN)
        set_max_fuel (f: INTEGER_32)
        dead
        decrease_fuel
        decrease_life
        decrease_actions_left_until_reproduction
        increase_fuel (i: INTEGER_32)
        increase_load
        set_load (i: INTEGER_32)

invariant
        allowable_symbols: item = 'E' or item = 'P' or item = 'A' or item = 'M' or item = 'J' or
                item = 'O' or item = 'W' or item = 'Y' or item = '*' or item = 'B' or item = '-'

end -- class ENTITY_ALPHABET
```

Class RANDOM_GENERATOR:

```
note
        description: "[
                The RANDOM_GENERATOR class is used to generate
                random numbers, either using the same seed
                (deterministically).
        ]"
        author: "Kevin Banh"
        date: "April 30, 2019"
        revision: "1"

class interface
        RANDOM_GENERATOR

create {RANDOM_GENERATOR_ACCESS}
        make_debug

feature -- queries

        num: INTEGER_32
                        -- Returns the current number in a sequence of random numbers

feature -- commands

        forth
                        -- Move to next number in a sequence of random numbers

end -- class RANDOM_GENERATOR
```

Class SHARED_INFORMATION:

```
note
	description: "[
		Common variables such as thresholds of movable entities
		and constants such as number of stationary items for generation of the board.
	]"
	author: ""
	date: "$Date$"
	revision: "$Revision$"

class interface
	SHARED_INFORMATION

create {SHARED_INFORMATION_ACCESS}
	make

feature

	Number_rows: INTEGER_32 = 5
			-- The number of rows in the grid
	Number_columns: INTEGER_32 = 5
			-- The number of columns in the  grid
	Number_of_stationary_items: INTEGER_32 = 10
			-- The number of stationary_items in the grid
	Janitaur_max_fuel: INTEGER_32 = 5
	Janitaur_reproduction: INTEGER_32 = 2
	Janitaur_max_load: INTEGER_32 = 2
	Benign_max_fuel: INTEGER_32 = 3
	Benign_reproduction: INTEGER_32 = 1
	Malevolent_max_fuel: INTEGER_32 = 3
	Malevolent_reproduction: INTEGER_32 = 1
	next_moveable_id: INTEGER_32
	asteroid_threshold: INTEGER_32
			-- used to determine the chance of an asteroid being put in a location
	janitaur_threshold: INTEGER_32
			-- used to determine the chance of a janitaur being put in a location
	malevolent_threshold: INTEGER_32
			-- used to determine the chance of a malevolent being put in a location
	benign_threshold: INTEGER_32
			-- used to determine the chance of a benign being put in a location
	planet_threshold: INTEGER_32
			-- used to determine the chance of a planet being put in a location
	Max_capacity: INTEGER_32 = 4
			-- max number of objects that can be stored in a location
	Max_fuel: INTEGER_32 = 3

feature --commands

	test (a_threshold: INTEGER_32; j_threshold: INTEGER_32; m_threshold: INTEGER_32;
				b_threshold: INTEGER_32; p_threshold: INTEGER_32)
			--sets threshold values
		require
			valid_threshold: 0 < a_threshold and a_threshold <= j_threshold and
				j_threshold <= m_threshold and m_threshold <= b_threshold and
				b_threshold <= p_threshold and p_threshold <= 101

	set_malevolent_threshold (threshold: INTEGER_32)
	set_janitaur_threshold (threshold: INTEGER_32)
	set_asteroid_threshold (threshold: INTEGER_32)
	set_planet_threshold (threshold: INTEGER_32)
	set_benign_threshold (threshold: INTEGER_32)
	intialize_next_id

end -- class SHARED_INFORMATION
```

Class RANDOM_GENERATOR_ACCESS:

```
note
	description: "[
		Singleton for accessing RANDOM_GENERATOR.
	]"
	author: "Kevin Banh"
	date: "April 30, 2019"
	revision: "1"

expanded class interface
	RANDOM_GENERATOR_ACCESS

create
	default_create

feature -- Query

	Debug_gen: RANDOM_GENERATOR
			-- deterministic generator for debug mode

	rchoose (low: INTEGER_32; high: INTEGER_32): INTEGER_32
			--generates a number from low to high inclusive
		require
			valid_num: low >= 0 and high > 0
			valid_range: low < high

invariant
		Debug_gen = Debug_gen

end -- class RANDOM_GENERATOR_ACCESS
```

Class SHARED_INFORMATION_ACCESS:

```
note
	description: "Singleton access for shared information."
	author: ""
	date: "$Date$"
	revision: "$Revision$"

expanded class interface
	SHARED_INFORMATION_ACCESS

create
	default_create

feature -- Query

	Shared_info: SHARED_INFORMATION

invariant
		Shared_info = Shared_info

end -- class SHARED_INFORMATION_ACCESS
```