



EECS3311 - W20 - SimOdyssey2 Report

Submitted electronically by:

Team members	Name	Prism Login	Signature
Member 1:	Chethana Wickramasinghe	chewick	
Member 2:	Jamie Dishy	jdishy	
*Submitted under Prism accounts: jdishy, chewick			

Contents

1. Requirements for SimOdyssey2 Project	2
2. BON Class Diagram Overview (Architecture of the Design)	3
3. Table of Modules — Responsibilities and Information Hiding	5
4. Expanded Description of Design Decisions	7
5. Significant Contracts (Correctness)	9
6. Summary of Testing Procedures	11
7. Appendix (Contract View of All Classes)	13

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 SimOdyssey2 Project

The following section reviews the requirements provided by the customer to achieve their needs:

The subject is to design and construct a simulator to train a space explorer to search different sectors of the galaxy. This simulation is a two-dimensional 5 by 5 grid of sectors. Each sector in a grid is identified by its coordinates in terms of the row number and the column number. Each sector contains four quadrants and may contain at most four entities, with one entity per quadrant. There are 10 entities involved in SimOdyssey2 that are divided between movable entities and stationary entities. Movable entities include Explorer, Asteroid, Benign, Planet, Malevolent and Janitaur. Stationary entities include Wormhole, Blackhole, Blue Giant and Yellow Dwarf. Taking these entities into account, the Explorer must move throughout the simulated galaxy with the purpose of finding a planet at the same sector as a Yellow Dwarf that supports life.

The SimOdyssey2 simulation may be started in two different modes: test mode or play mode. Test mode provides the user with grid details including entity descriptions¹, dead entities, attacked entities, cloned entities, and further allows the user to influence the concentration of entity types on the grid based on threshold. In play mode, there is less information and the initial density of entities cannot be influenced. In both cases, the Explorer always begins at row 1, column 1, while the black hole always exists at row 3, column 3. All other entities are randomly placed in positions on the grid. Throughout this simulation, you must take into account entities that harm the Explorer, entities that harm other entities, entities that may clone themselves, as well as entities that move other entities to random sectors on the grid.

The explorer may move in one of eight directions, associated to the eight adjacent sectors. These are found in the north, north-east, east, south-east, south, south-west, west, north-west positions directly adjacent to the given sector. Note that the grid wraps along its boundaries, meaning that moving north at row one takes the explorer to row 5 at the bottom of the grid. For each move the Explorer takes, other movable entities will subsequently move according to their turn value. If a movable entity moves, its associated parameters must be updated. The state of the Explorer may change to “is dead” if its life runs out, runs out of fuel, enters a black hole, or is hit by an Asteroid.

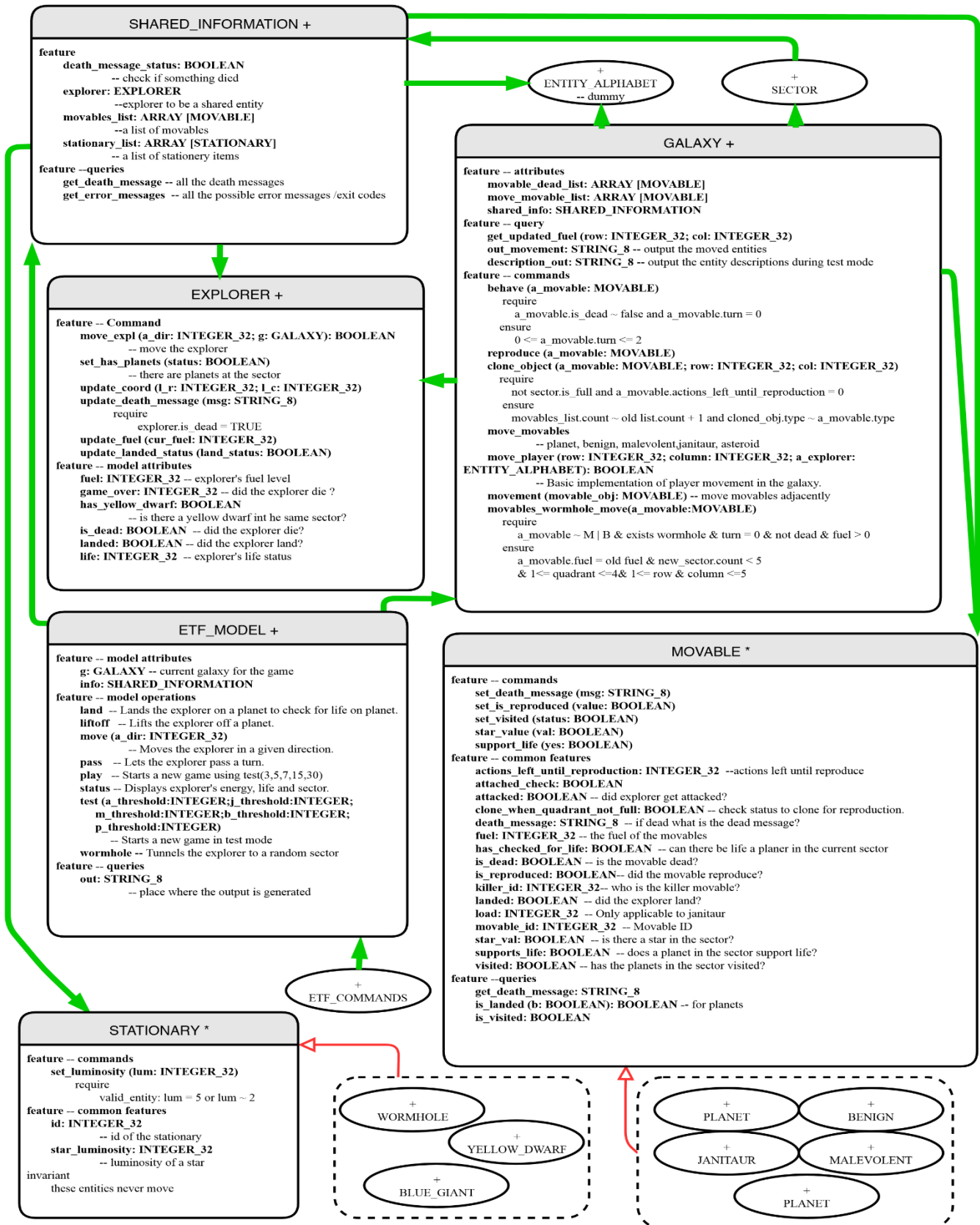
In the case that user input is invalid, particular error messages are displayed in the simulation. Such invalid scenarios include starting a game during an existing game, landing an Explorer that is not at a planet, lifting an Explorer off from a planet that has not landed, inputting a threshold that is not in non-descending order, and moving an Explorer to a full sector.

The acceptance tests in section 6 describe some of the input-output behaviour at the console for this project.

¹ Entity descriptions may include ID, fuel, turn value, life, load, actions_left_until_reproduction, visited, attached, supports life.

2. BON Class Diagram Overview (Architecture of the Design)

A detailed GALAXY with all contracts is in [Appendix B](#)



Overall design and the main design decisions

Simplicity:

Overall there is a cluster of commands, a Galaxy where an Explorer, some Movable and Stationary Objects are placed inside the created galaxy during the first valid Play or Test mode.

Choosing the right abstractions:

Created abstraction class called STATIONARY and MOVABLE. STATIONARY help locate all the stationary objects been created in the galaxy and give ID in the appropriate order. MOVABLE help to create all the movable objects, give IDs accordingly and keep track of each movable's actions. Decided to not include explorer as part of the movable to have independent control over the player.

Information hiding:

Command and query features of GALAXY DEFERRED MODULE can only be access by ETF_MODEL.

Command features of MOVABLE ABSTRACT MODULE can only be accessed by GALAXY, SECTOR, and SHARED_INFORMATION.

Command features of STATIONARY ABSTRACT MODULE can only be accessed by GALAXY and SHARED_INFORMATION.

From above examples it will be clear how information hiding is used to have limited access to some modules but not the rest of the other modules. This way it is guaranteed only authorized modules can make functional changes.

Reliability (decreasing the likelihood of bugs):

One of the special approaches taken to ensure reliability is to have a filtering mechanism for the 'out' feature in ETF_MODEL. Where this 'out' feature first filters all errors and next filters out according to the mode(test/play), then filters according to each command selected by the use. Another example is to have all death messages sorted and categorized in shared information so that they can be accessed easily and guaranty that the message specific to each condition to be correct. Furthermore, in GALAXY and MOVABLE it seems like it has a lot of features, but it is because all the actions are categorized, so that is will be easy to make quick changes in minimum number of locations and to make debugging purposes easy. This will also be a great example to show where Re-usability is taken into consideration. In other words, having more specific features make it easy to use them for different purposes and keep the functionality simple as possible.

Extendibility (minimizing adaptation effort when the problem varies):

For example, death message is an extension of the description out of each dead movable object. This way the death messages are not constructed from scratch but simply an add-on to the description out of a movable.

3. Table of Modules — Responsibilities and Information Hiding

Note: Modules that are significant for their design implementation are listed in this section. For further review of other modules, please see the [Appendix A](#)

1	GALAXY	Responsibility: Implements the simulation display, object movement and behaviour	Alternative: Only implement the simulation display and designate features for object movement and behaviour to a separate class
	Concrete	Secret: <ul style="list-style-type: none"> - Accesses particular SECTORs using data structure ARRAY2 - Stores objects that moved in an array list for simulation output - Stores dead objects in an array list and sorts them for simulation display - Replaces old object positions with a dash symbol ‘-’ 	
2	MOVABLE	Responsibility: Creates both deferred and effective features that may be used by inheriting objects that move in the simulation	Alternative: Since not all inheriting classes implement the deferred features in MOVABLE, create an interface. The interface will implement the deferred features used by a subset of inheriting classes from MOVABLE.
	Abstract	Secret: none	

3	STATIONARY	Responsibility: Creates deferred features that may be used by inheriting objects that are stationary in the simulation	Alternative: none
	Abstract	Secret: none	

4	EXPLORER	Responsibility: Implements features for EXPLORER object	Alternative: EXPLORER inherits from MOVABLE abstract class
	Concrete	Secret: none	

5	SHARED_INFORMATION	Responsibility: For single references of EXPLORER, MOVABLE objects and STATIONARY objects. Contains error messages and death messages for display.	Alternative: Designate a separate class with Singleton structure to implement error and death messages for the simulation display
	Concrete	Secret: <ul style="list-style-type: none"> - Error messages handled and organized via switch statements 	

6	ETF_MODEL	Responsibility: Calls particular features in a specific order based on user input	Alternative: none
	Concrete	Secret: <ul style="list-style-type: none"> - Implements flags to influence the appropriate display based on user input (ie <i>play_check</i>, <i>abort_flag</i>, <i>cmd_name</i>...etc.) 	

4. Expanded Description of Design Decisions

The most important module in SimOdyssey2 is GALAXY with its implemented feature commands *move_movable*[MOVABLE], *wormhole_move_movable*[MOVABLE], *reproduce*[MOVABLE] and *behave*[MOVABLE]. This module, along with its feature commands, handles the core functionality of each movable entity created on the grid by moving them, reproducing them, and destroying them if applicable. Doing so involves updating object attributes such as fuel level, life level, as well as turn values. The following paragraphs will delve into how GALAXY works and will describe the design decisions that are implemented for optimal simplicity, maintainability and reusability, as movable entities are handled and tracked. In particular, these design decisions involve applying Singleton design, as well as implementing an inheritance structure. Additionally, this section will address trade-offs for the design decisions and why this implementation was chosen.

GALAXY Functionality

To begin, GALAXY iterates through each movable entity on the grid, that is not dead, and may update the entity's sector along with its parameters impacted by the movement. While *movement*[MOVABLE] and *movable_wormhole*[MOVABLE] move the object onto a particular sector and quadrant of the grid, *update_fuel*[MOVABLE] updates the fuel of movable entities, *reproduce*[MOVABLE] is called to clone the object if applicable, and *behave*[MOVABLE] updates the object's turn value and may destroy other objects in the process. These features are concise and handle various possible cases that may arise, such as destroying objects that land in the same sector as the black hole, increasing the fuel value if a wormhole places an object in the same sector as a star, as well as handling situations where the expected sector to move to is full. As the features of GALAXY play a fundamental role in creating a working simulation, they must be designed clearly for maintainability and reusability in the future. To do this, Singleton design as well as inheritance structures are used, which shape the entire overall code structure of this project.

Singleton Design

Singleton design throughout SimOdyssey2 allows for convenient usage and tracking of objects that have single references to them. For example, with each Explorer action, the turn value of each movable entity must be updated. This requirement is conveniently handled by ensuring a single reference to each movable object, from their creation to their death state. To implement Singleton design, all movable entities are stored in an array list within SHARED_INFORMATION, called *movable_list*[MOVABLE], which is populated when objects are created in SECTOR. This decision allows for reusable and extendible code because regardless of the number of objects on the grid - from 10 to 100 - all movable objects are conveniently stored in one single reference place for usage. This is particularly useful during *test* mode, where users may influence the concentration of different movable objects. From updating their turn values, to moving an object to a new sector, to determining whether or not fuel should be updated, these factors are easily managed based on the single reference principle.

Inheritance Structure

That being said, using Singleton design is possible particularly due to the inheritance structure of MOVABLE class. MOVABLE is a deferred abstract class that initializes features used by PLANET, JANITAUR, ASTEROID, BENIGN and MALEVOLENT. Features initialized for these objects include fuel levels, determining death state, actions_until_reproduction, as well as turn value. As such, when each

movable entity is first generated in `SECTOR.populate`, they reference features listed in `MOVABLE`, which are initialized depending on their particular inheriting class. For example, a newly created `JANITUR` object initializes `actions_until_next_reproduction` to 2, while that of `BENIGN` is 1. Since each `MOVABLE` object has associated features assigned when first created, these objects may conveniently be stored in the array list for single reference usage. This allows for simple access of object features and avoids needing to create unnecessary data structures like hashmaps created for each object. That stated, inheritance highly minimizes the need to create data structures to store information for each object and allows for easy extendability if features must be added for each movable object.

Tradeoffs

As the use of Singleton design and inheritance are crucial for convenience and maintainability, they have been implemented in particular ways that result in tradeoffs. Beginning with inheritance, the current implementation deals with the abstract class `MOVABLE` with various inheriting concrete classes. The tradeoff associated with this design decision is that not all inheriting classes use the features from `MOVABLE`, rendering some features as unnecessary and unused. For example, `ASTEROID` and `PLANET` inherit from `MOVABLE`, however they do not require features that deal with *fuel* or *actions_left_until_reproduction*. An alternative approach might be to create abstract class `MOVABLE` with deferred features that are used by all inheriting classes, and create an additional interface for deferred features used by particular inheriting classes. For example, an interface can be used to create deferred features for *fuel* and *actions_left_until_reproduction*, since not all classes that inherit from `MOVABLE` will implement these. This way, `PLANET` and `ASTEROID` will not inherit from the interface, while the other concrete classes that must implement these features will inherit from the interface.

Taking this tradeoff into account, the current structural design was chosen for the simplicity of inheriting from one class, rather than adding complexity with multiple concrete classes inheriting from a deferred class and potentially multiple interfaces. Because the current approach trades a few unused features for a simple inheritance structure, `GALAXY` only needs to deal with `MOVABLE` objects. Most importantly, the current design structure continues to support optimal reusability, maintainability and extendability. Since these factors are not jeopardized by the current design decision, the developers proceeded with this approach.

Moreover, the core functionality of `SymOdyssey2` is implemented in `GALAXY`, which is what makes this module most significant. The tradeoff for this design decision is that `GALAXY` may appear convoluted and overly full, and can be shortened by implementing some features in another class. This may include implementing the features for movement in a different class. That being said, the current approach was chosen for convenience, as all of the features in `GALAXY` are closely related and directly influence one another.

5. Significant Contracts (Correctness)

The following section lists the most significant contracts within module GALAXY. Note the relationship between **routine and subroutine preconditions and postconditions**. For example, the postcondition of a routine may reference the precondition of a subroutine; if this subroutine precondition is met, then the subroutine's post-condition is implied.

reproduce (a_movable: MOVABLE)

-- updates actions_left_until_reproduction and calls clone_object if applicable
 -- significant contract with *clone_object*

require

a_movable.entity_alphabet /~ 'P' and a_movable.entity_alphabet /~ 'A'

ensure

if not sector.is_full and actions_left_until_reproduction = 0 →

-- **postcondition of clone_object:**

shared_info.movables_list.count ~ old shared_info.movables_list.count + 1 and shared_info.movables_list[count].type ~ a_movable.type and shared_info.movables_list[count].actions_left_until_reproduction <= 2 and shared_info.movables_list[count].turn <= 2
 a_movable.actions_left_until_reproduction <= 2

clone_object (a_movable: MOVABLE; row: INTEGER_32; col: INTEGER_32)

-- clones object *a_movable*

require

not sector.is_full and a_movable.actions_left_until_reproduction = 0

ensure

shared_info.movables_list.count ~ old shared_info.movables_list.count + 1 and shared_info.movables_list[count].type ~ a_movable.type and shared_info.movables_list[count].actions_left_until_reproduction <= 2 and shared_info.movables_list[count].turn <= 2

move_movables

-- only moves a planet or benign or malevolent or janitaur or asteroid
 -- calls either *movement* or *movable_wormhole_move*
 -- significant contracts with *movement* and *movable_wormhole_move*

require

shared_info.movables_list.count > 0 and
 \forall shared_info.movables_list \nexists ('E' and '*' and 'Y')

ensure

0 < a_movable.row < 6 and 0 < a_movable.column < 6 and 0 < a_movable.quadrant < 5

across shared_info.movables_list as movable_object all

-- **precondition of movable_wormhole_move**

if movable_object is MALEVOLENT or BENIGN and \exists wormhole at [movable_object.row, movable_object.column] →

-- **postcondition of movable_wormhole_move**

a_movable.fuel = old a_movable.deep_twin.fuel and

a_movable.new_sector.count < 5

a_movable.quadrant between 1-4

a_movable.row and a_movable.column are between 1 and 5

else

-- **postcondition of movement**

$a_movable \nexists$ at $a_movable.old_sector$
 $a_movable$ is in its new sector on a quadrant between 1-4
 $a_movable.row$ and $a_movable.column$ are between 1 and 5

movable_wormhole_move ($a_movable$: MOVABLE)

-- moves an object of type MOVABLE through a wormhole

require

$a_movable$ is MALEVOLENT or BENIGN and \exists wormhole at [$a_movable.row$,
 $a_movable.column$]
 $a_movable.turn = 0$ and
 $a_movable.is_dead = FALSE$ and
 $a_movable.fuel > 0$

ensure

$a_movable.fuel = old\ a_movable.deep_twin.fuel$ and
 $a_movable.new_sector.count < 5$
 $a_movable.quadrant$ between 1-4
 $a_movable.row$ and $a_movable.column$ are between 1 and 5

movement ($movable_obj$: MOVABLE)

-- moves an object of type MOVABLE to adjacent sector if not full

require

$movable_obj.turn = 0$ and
 $movable_obj.is_dead = FALSE$ and
 $movable_obj.fuel > 0$

ensure

$movable_obj \nexists$ at $movable_obj.old_sector$
 $movable_obj$ is in its new sector on a quadrant between 1-4
 $movable_obj.row$ and $movable_obj.column$ are between 1 and 5

behave ($a_movable$: MOVABLE)

-- handles attacks and destroys objects if applicable

require

$a_movable.is_dead \sim false$ and
 $a_movable.turn = 0$

ensure

$0 \leq a_movable.turn \leq 2$

6. Summary of Testing Procedures

TEST FILE	DESCRIPTION	PASSED
at001.txt	-- Tests winning condition in test mode.	PASSED
at002.txt	--Tests winning condition in play mode.	PASSED
at003.txt	--Tests losing condition in test mode (lose by out of life).	PASSED
at004.txt	--Tested for different test thresholds and moving explorer outside the grid.	PASSED
at005.txt	--Test land for scenarios where there is no planet and no yellow dwarf separately.	PASSED
at006.txt	--Test a scenario where explorer land successfully.	PASSED
at007.txt	--Testing for reproduction and destroying.	PASSED
at008.txt	--Testing for running out of fuel and going out of bounds.	PASSED
at009.txt	--Testing for explorer getting devoured by blackhole and game ends.	PASSED
at10.txt	--Wormhole sends a movable to a sector.	PASSED
at11.txt	--Test for error messages and command status.	PASSED
at12.txt	--Test wormhole command on explorer when there is a wormhole in the same sector as the explorer.	PASSED
at13.txt	--Test for test command with not non-decreasing thresholds and test for explorer getting destroyed by an asteroid.	PASSED
at14.txt	--Test for the maximum thresholds test (97,98,99,100,101).	PASSED

7. Appendix (Contract View of All Classes)

Appendix A

1	YELLOW_DWARF	Responsibility: Implements inherited features for stationary YELLOW_DWARF objects	Alternative: none
	Concrete	Secret: none	

2	BLUE_GIANT	Responsibility: Stationary object with a unique entity '*' and a unique luminosity value	Alternative: none
	Concrete	Secret: none	

3	WORMHOLE	Responsibility: Implements inherited features for stationary WORMHOLE objects	Alternative: none
	Concrete	Secret: none	

4	MALEVOLENT	Responsibility: Implements inherited features for MALEVOLENT objects	Alternative: none
	Concrete	Secret: none	

5	ASTEROID	Responsibility: Implements inherited features for ASTEROID objects	Alternative: none
---	----------	---	--------------------------

	Concrete	Secret: none	
--	----------	---------------------	--

6	BENIGN	Responsibility: Implements inherited features for BENIGN objects	Alternative: none
	Concrete	Secret: none	

7	PLANET	Responsibility: Implements inherited features for PLANET objects	Alternative: none
	Concrete	Secret: none	

8	JANITUR	Responsibility: Implements inherited features for JANITUR objects	Alternative: none
	Concrete	Secret: <ul style="list-style-type: none"> - Stores asteroids in its load using a counter of type INTEGER 	

9	SECTOR	Responsibility: Creates stationary and movable objects and allocates them throughout sectors of the grid	Alternative: none
	Concrete	Secret: none	

Appendix B

GALAXY +

```

feature -- attributes
    movable_dead_list: ARRAY [MOVABLE]
    move_movable_list: ARRAY [MOVABLE]
    shared_info: SHARED_INFORMATION

feature -- query
    get_updated_fuel (row: INTEGER_32; col: INTEGER_32)
    out_movement: STRING_8 -- output the moved entities
    description_out: STRING_8 -- output the entity descriptions during test mode

feature -- commands
    behave (a_movable: MOVABLE)
        -- handles attacks and destroys objects if applicable
        require
            a_movable.is_dead ~ false and
            a_movable.turn = 0
        ensure
            0 <= a_movable.turn <= 2

    reproduce (a_movable: MOVABLE)
        -- updates actions_left_until_reproduction and calls clone_object if applicable
        -- significant contract with clone_object
        require
            a_movable.entity_alphabet /~ 'P' and a_movable.entity_alphabet /~ 'A'
        ensure
            if not sector.is_full and actions_left_until_reproduction = 0 →
                -- postcondition of clone_object:
                shared_info.movables_list.count ~ old shared_info.movables_list.count + 1 and
                shared_info.movables_list[count].type ~ a_movable.type and
                shared_info.movables_list[count].actions_left_until_reproduction <= 2 and shared_info.movables_list[count].turn <= 2
                a_movable.actions_left_until_reproduction <= 2

    clone_object (a_movable: MOVABLE; row: INTEGER_32; col: INTEGER_32)
        require
            not sector.is_full and a_movable.actions_left_until_reproduction = 0
        ensure
            movables_list.count ~ old list.count + 1 and cloned_obj.type ~ a_movable.type

    move_movables
        -- only moves a planet or benign or malevolent or janitaur or asteroid
        -- calls either movement or movable_wormhole_move
        -- significant contracts with movement and movable_wormhole_move
        require
            shared_info.movables_list.count > 0 and
            all shared_info.movables_list # ('E' and '*' and 'Y')
        ensure
            0 < a_movable.row < 6 and 0 < a_movable.column < 6 and 0 < a_movable.quadrant < 5
            across shared_info.movables_list as movable_object all
                -- precondition of movable_wormhole_move
                if movable_object is MALEVOLENT or BENIGN and wormhole at [movable_object.row, movable_object.column] →
                    -- postcondition of movable_wormhole_move
                    a_movable.fuel = old a_movable.deep_twin.fuel and a_movable.new_sector.count < 5
                    a_movable.quadrant between 1-4
                        a_movable.row and a_movable.column are between 1 and 5
                    else
                        -- postcondition of movement
                        a_movable # at a_movable.old_sector
                        a_movable is in its new sector on a quadrant between 1-4
                        a_movable.row and a_movable.column are between 1 and 5

    move_player (row: INTEGER_32; column: INTEGER_32; a_explorer: ENTITY_ALPHABET): BOOLEAN
        -- Basic implementation of player movement in the galaxy.
        ensure
            0 < new_row < 6 and 0 < new_column < 6 and 0 < new_quadrant < 5

    movement (movable_obj: MOVABLE) -- move movables adjacently
        -- moves an object of type MOVABLE to the adjacent sector if not full
        require
            movable_obj.turn = 0 and
            movable_obj.is_dead = FALSE and
            movable_obj.fuel > 0
        ensure
            movable_obj /at movable_obj.old_sector
            movable_obj is in its new sector on a quadrant between 1-4
            movable_obj.row and movable_obj.column are between 1 and 5

    movables_wormhole_move(a_movable:MOVABLE)
        -- moves an object of type MOVABLE through a wormhole
        require
            a_movable is MALEVOLENT or BENIGN and wormhole at [a_movable.row, a_movable.column]
            a_movable.turn = 0 and
            a_movable.is_dead = FALSE and
            a_movable.fuel > 0
        ensure
            a_movable.fuel = old a_movable.deep_twin.fuel and a_movable.new_sector.count < 5
            a_movable.quadrant between 1-4
            a_movable.row and a_movable.column are between 1 and 5.

```