EECS3311-W20 Software Design
SimOdyssey Lab4

By retrieving this Lab and using this document you affirm that you are registered in EECS3311-W20 at York University this term. Registered EECS3311 students may download the Lab for their private use, but may not communicate it to anyone else, even other students in the class. This document and any of your work may not appear on the internet, but must be **private**. Students may do this Lab on their own, or with one other member of the course. A team contains a maximum of two students (no exceptions). Each team must submit their own work thus complying with York academic integrity guidelines. See course wiki for details of academic integrity principles.

Table of Contents

1. Lab4 goals and Design Document

Refer to the course wiki for details on the electronic submission of this lab, including the code and a report.

Lab4 is *Phase1* of the course Project which requires you design a galactic game. In the Project (which is *Phase2*), there will be additional features and entities. It is thus important that Phase1 be well-designed to ensure that adding behavior will not destroy the cohesiveness of the design, so that the design is maintainable after deployment.

Design is the construction of abstractions of data and computation and the organization of these abstractions into a working software application (*Introduction to Software Design with Java*, M. Robillard, 2017)

In design, the skill you wish to develop is the ability to distill a complex problem into its simplest components, and to organize the components into a cohesive and maintainable product.

Keep in mind that in this Lab and even more in the Project, you will need to produce a Design Document. For more on this document, see the footnote.[1]

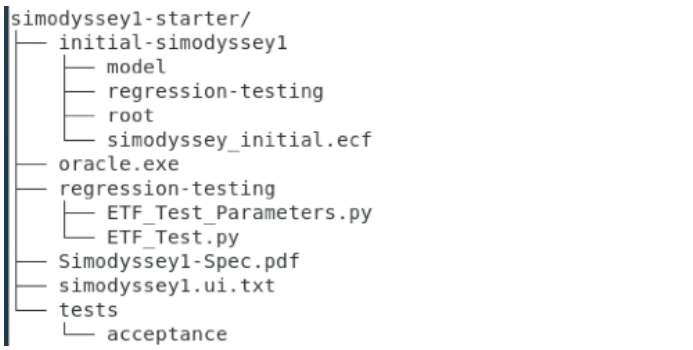
2. What you are provided with

For this lab you are provided with many resources including a specification of the Lab, a UI grammar (with which you will generate an ETF project), some acceptance tests, an oracle (in place of your customer) for writing and validating your own tests, and some starter code for board generation. See Figure 1.

In Appendix A, you will find an **algorithm** (rather than code) that specifies abstractly the generation and movement of entities in the game (which can also be inferred from the oracle). Ultimately, we will use the oracle to assess the correctness of your design.

Note: If you merely attempt to implement the algorithm directly, without thinking about design (information hiding, modularity, classes, their features, client-supplier/inheritance relationships, architecture, etc.), then your code might work but your design is likely to be poor with code smells (such as Superman classes that do too much). In your design documentation (e.g., BON class diagrams), such direct implementation from the algorithms will not result in a passing grade.

[1] http://seldoc.eecs.yorku.ca/doku.php/eiffel/sdd/start



Simodyssey1-Spec.pdf
-- specification document

simoddyssey1.ui.txt
-- grammar of user commands at User Interface

initial-simodyssey/
-- starter code to generate an initial galaxy

oracle.exe
-- this is your customer who will confirm
-- that your own acceptance tests are valid

tests/
-- we provide you with 3 acceptance tests
-- you must write many more for design correctness

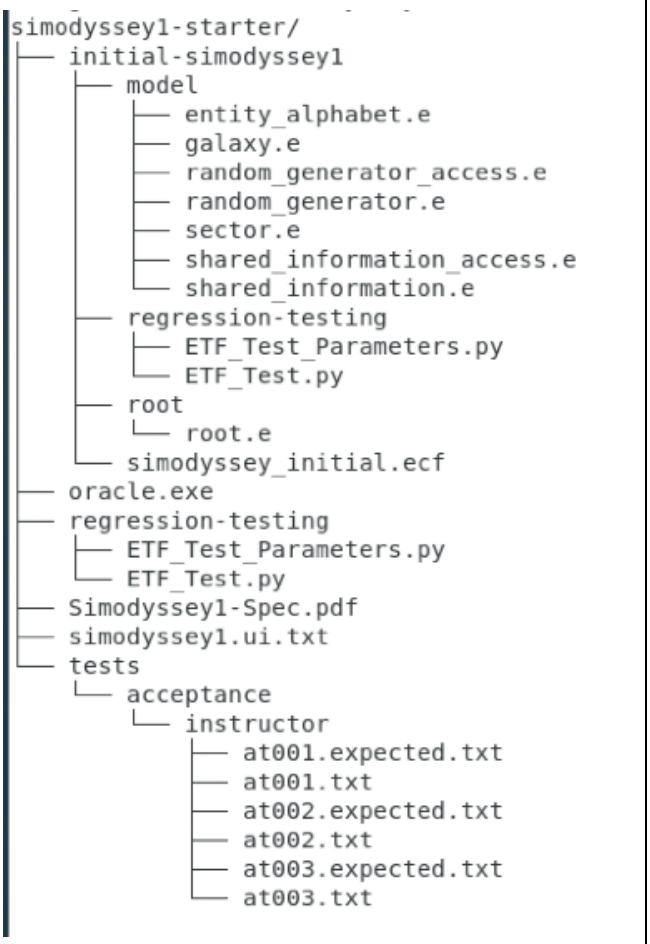


Figure 1 Specification

Suggested workflow:

- Read carefully this specification document to understand both the background of the game (before the Appendix) and abstract algorithms of how the game should be implemented.
- While you are reading this document, you are advised to play with the *oracle.exe* to obtain a strong grasp of the assignment.
- We assume that you are familiar with designing systems with ETF.
- In the event there is conflicting information between the oracle and documentation, prioritize the oracle and report the bug.
- Review the starter code which provides a way to generate the board. See section How to Use the Starter Project for a guide on the starter code.
- Incrementally code a solution by the basic acceptance tests given to you, then by your own acceptance tests (whose expected outputs should be confirmed by the oracle). Ensure that you perform regular regression testing.

3. SimOdyssey: The story so far…

The nations of Earth are in desperate need of a galaxy exploration simulator to prepare a new generation for deep space exploration. The combined effects of the hole in the ozone layer, global warming, nuclear catastrophes and the fear that ABBA might still reform have inspired action. At long last, our planet has finally mounted a concerted effort not merely to find out if "anyone's out there", but whether or not we can join them. As such, they require a simulator to train space explorers to search different sectors of our galaxy containing stars of the same type as our own sun. These stars, known as "Yellow Dwarfs", are believed to hold the best hope of discovering planets that support life - at least "as we know it". The explorer's mission is to see if such stars 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 (rock and atmospheric sampling for oxygen, carbon or water, existence of Pizza Huts, and so on). The game is won and the simulation ends when a planet capable of supporting life is discovered.

Having been recently trained in the art of software design, the joint nations of Earth tasked YOU with the design and construction of the simulator.

4. What is SimOdyssey?

1,1	1,2	1,3	1,4	1,5
2,1	2,2	2,3	2,4	2,5
3,1	3,2	3,3	3,4	3,5
4,1	4,2	4,3	4,4	4,5
5,1	5,2	5,3	5,4	5,5

Figure 2 How the Galaxy fits together

The system to be modeled and implemented is a simplified simulation of a galaxy. A two-dimensional grid of sectors represents the galaxy. In the current game you are being asked to program, the size of the grid is 5 by 5 so the grid will look like Figure 2. Each sector in the grid is identified by its coordinates in terms of the row number and the column number.

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 For example, looking at sector (3,3),

we can see that (2,4) is the north-east neighbour.

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. For example, the north neighbour of sector (1,3) is sector (5,3). Each sector contains four quadrants where each quadrant may contain an entity. An example of the distribution of the entities, when the game is first started, could be as follows:

1,1 **Explorer**	1,2 **Blue Giant**	1,3 **Planet**	1,4	1,5 **Planet** **Planet**
2,1	2,2	2,3 **Wormhole**	2,4	2,5 **Planet**
3,1 **Planet** **Yellow Dwarf**	3,2	3,3 **Black Hole**	3,4 **Planet** **Wormhole**	3,5
4,1 **Wormhole**	4,2 **Planet** **Blue Giant**	4,3 **Planet**	4,4 **Planet** **Blue Giant**	4,5
5,1 **Planet** **Wormhole**	5,2	5,3 **Blue Giant**	5,4	5,5 **Planet** **Wormhole**

A sample simulation (execution) is shown in Figure 3, where each entity has its corresponding character:

```
Stationary
--------
O: Blackhole
Y: Yellow Dwarf
*: Blue Giant
W: Wormhole

Movable:
--------
E: Explorer
P: Planet
```

Figure 3 Sample run

```
   state:0.0, ok
  Welcome! Try test(30)
->play
  state:1.0, mode:play, ok
  Movement:none
    (1:1)  (1:2)  (1:3)  (1:4)  (1:5)
    E---   *---   P---   ----   PP--
    (2:1)  (2:2)  (2:3)  (2:4)  (2:5)
    ----   ----   W---   ----   P---
    (3:1)  (3:2)  (3:3)  (3:4)  (3:5)
    PY--   ----   O---   PW--   ----
    (4:1)  (4:2)  (4:3)  (4:4)  (4:5)
    W---   P*--   P---   P*--   ----
    (5:1)  (5:2)  (5:3)  (5:4)  (5:5)
    PW--   ----   *---   ----   PW--
->move(N)
  state:2.0, mode:play, ok
  Movement:
    [0,E]:[1,1,1]->[5,1,3]
    [3,P]:[1,5,2]->[5,4,1]
    [8,P]:[4,3,1]->[3,2,1]
    (1:1)  (1:2)  (1:3)  (1:4)  (1:5)
    ----   *---   P---   ----   P---
    (2:1)  (2:2)  (2:3)  (2:4)  (2:5)
    ----   ----   W---   ----   P---
    (3:1)  (3:2)  (3:3)  (3:4)  (3:5)
    PY--   P---   O---   PW--   ----
    (4:1)  (4:2)  (4:3)  (4:4)  (4:5)
    W---   P*--   ----   P*--   ----
    (5:1)  (5:2)  (5:3)  (5:4)  (5:5)
    PWE-   ----   *---   P---   PW--
->wormhole
  state:3.0, mode:play, ok
  Movement:
    [0,E]:[5,1,3]->[2,2,1]
    [8,P]:[3,2,1]->[4,3,1]
    [10,P]:[5,1,1]->[1,5,2]
    (1:1)  (1:2)  (1:3)  (1:4)  (1:5)
    ----   *---   P---   ----   PP--
    (2:1)  (2:2)  (2:3)  (2:4)  (2:5)
    ----   E---   W---   ----   P---
    (3:1)  (3:2)  (3:3)  (3:4)  (3:5)
    PY--   ----   O---   PW--   ----
    (4:1)  (4:2)  (4:3)  (4:4)  (4:5)
    W---   P*--   P---   P*--   ----
    (5:1)  (5:2)  (5:3)  (5:4)  (5:5)
    -W--   ----   *---   P---   PW--
->move(SW)
  state:4.0, mode:play, ok
  Movement:
    [0,E]:[2,2,1]->[3,1,3]
    [1,P]:[1,3,1]->[2,4,1]
    [2,P]:[1,5,1]->[2,5,2]
    [4,P]:[2,5,1]->[2,4,2]
    [6,P]:[3,4,1]->[2,3,2]
    [10,P]:[1,5,2]->[2,4,3]
    [11,P]:[5,5,1]->[5,4,2]
    (1:1)  (1:2)  (1:3)  (1:4)  (1:5)
    ----   *---   ----   ----   ----
    (2:1)  (2:2)  (2:3)  (2:4)  (2:5)
    ----   ----   WP--   PPP-   -P--
    (3:1)  (3:2)  (3:3)  (3:4)  (3:5)
    PYE-   ----   O---   -W--   ----
    (4:1)  (4:2)  (4:3)  (4:4)  (4:5)
    W---   P*--   P---   P*--   ----
    (5:1)  (5:2)  (5:3)  (5:4)  (5:5)
    -W--   ----   *---   PP--   -W--
```

```
->land
  state:5.0, mode:play, ok
  Tranquility base here - we've got a life!
```

5. Entities (Summary)

Space (in the simulations) is inhabited by a variety of entities that have different behaviours. There are two main types of entities. They are:

Movable entities - which move throughout the galaxy interacting with other entities.

- Planet.
- The explorer - which is a unique movable entity controlled by the user.

Stationary entities - which stay in one place throughout the game.

- Wormhole, blackhole, blue giant and yellow dwarf (where blue giant and yellow dwarf are considered to also be a star object).

We repeat below the code for each entity:

```
Stationary
--------
O: Blackhole
Y: Yellow Dwarf
*: Blue Giant
W: Wormhole

Movable:
--------
E: Explorer
P: Planet
```

6. High-Level Summary of the Game

The SimOdyssey game can be started in two different modes:

- In **test** mode, more information is provided regarding the board, and the amount of planets created at the beginning can be influenced (e.g., test(90) has a higher probability of generating more planets compared with test(5)).
- In **play** mode, there is less information and the initial density of entities cannot be influenced.

Both modes use a deterministic "random" number generator: i.e., the generated sequence of numbers is always the same, but changes from play to play based on the same random seed. After starting a new game, a 5-by-5 board is generated. Other than (1,1) which has the explorer and (3,3) which contains a black hole, the other entities are generated and placed at "random" positions on the board. Each sector has 4 quadrants and thus at most 4 entities (one per quadrant). So, for example, a sector may have `PWE-` meaning a Planet, a Wormhole and an Explorer.

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 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.

7. Generation of the Board

This section contains a detailed version of how the board is generated. A more precise abstract algorithm for generating the board is included in Appendix A.

For the purpose of this game, the board size is 5 by 5. **It is imperative to use the random numbers generated in the exact manner as this document describes it, or your output will not match the oracle used to test your code**. Sample starter code is provided that does this and contains the random number generator you must use.

An example of the board generation is provided in the starter code that may be used in your final solution, although it will have to be refactored for good design.

Note that the starter code has a line in `{SECTOR}.populate`

```
turn := gen.rchoose(0, 2)
```

where the value of `turn` should be used for setting the number of turns left for that entity to behave (when creating a new movable entity).

Initially, each of the sectors of the board (galaxy) is created, starting from the leftmost to the rightmost column row-by-row (where the smallest rows are generated first). In other words, the sectors will be generated in the order of (1,1), (1,2) … (1,5), (2,1), (2,2) … (2,5), (3,1) and etc. The order is important as there is randomness involved. Each sector can contain up to 4 entities.

When a sector is generated, a number from 1 .. 3 is first randomly generated, denoting the maximum number of movable entities generated in that sector (excluding the explorer).

Note that for sector (3,3), no random number generation is required. A black hole is created at that sector.

For sector (1,1), an explorer is placed at the first quadrant before any generation is done.

The number generator then generates a number from 1 to 100 (called a *threshold*). If in *play* mode, if the number is 1 to 29, a planet is generated.

If in *test* mode, the specification of a numbers which determine the threshold, is required:

```
test (planet_threshold: THRESHOLD)
```

(as defined in the UI grammar file). The user command `play` is equivalent to `test(30)`. Thus if n is the random number generated, then if

$$0 < n < planet\_threshold$$

then a planet is generated; otherwise nothing is generated. The threshold thus controls the number of planets generated on the board. The higher the threshold, the more likely more planets are generated as shown in Figure 4. This allows test mode to check the game for different concentrations of planets. The number for specifying the threshold is bounded between 1 to 101.

Figure 4 Sample generation of the board via threshold

```
->test(30)...
    (1:1)   (1:2)   (1:3)   (1:4)   (1:5)
    E---    *---    P---    ----    PP--
    (2:1)   (2:2)   (2:3)   (2:4)   (2:5)
    ----    ----    W---    ----    P---
    (3:1)   (3:2)   (3:3)   (3:4)   (3:5)
    PY--    ----    O---    PW--    ----
    (4:1)   (4:2)   (4:3)   (4:4)   (4:5)
    W---    P*--    P---    P*--    ----
    (5:1)   (5:2)   (5:3)   (5:4)   (5:5)
    PW--    ----    *---    ----    PW--
->abort...
->test(1)...
    (1:1)   (1:2)   (1:3)   (1:4)   (1:5)
    EY--    ----    ----    W---    Y---
    (2:1)   (2:2)   (2:3)   (2:4)   (2:5)
    ----    *---    ----    ----    ----
    (3:1)   (3:2)   (3:3)   (3:4)   (3:5)
    W---    ----    O---    ----    ----
    (4:1)   (4:2)   (4:3)   (4:4)   (4:5)
    ----    *---    ----    ----    Y---
    (5:1)   (5:2)   (5:3)   (5:4)   (5:5)
    ----    Y---    ----    W---    W---
->abort...
->test(100)...
    (1:1)   (1:2)   (1:3)   (1:4)   (1:5)
    EPPP    PPY-    PPW-    PP--    PPW-
    (2:1)   (2:2)   (2:3)   (2:4)   (2:5)
    P---    P---    PP--    PP--    P---
    (3:1)   (3:2)   (3:3)   (3:4)   (3:5)
    PW--    PPP-    O---    PW--    PY--
    (4:1)   (4:2)   (4:3)   (4:4)   (4:5)
    PP*-    PW--    P---    P---    PPP-
    (5:1)   (5:2)   (5:3)   (5:4)   (5:5)
    PP--    P---    P*--    PP--    PPY-
```

As documented in the grammar, the command `test` in the above table has an argument that allows the software developer to create different initial distributions of movable entities on the board for testing the application (see Figure 5).

Figure 5 UI grammar for test parameter

`test(planet_threshold: THRESHOLD)`	**type** THRESHOLD = 1..101

For example, `test(1)` creates an initial distribution of entities on the board in which there are no movable entities, only stationary ones, other than the explorer at sector (1,1). `test(100)` gives a high density amount of planets while `test(30)` gives a modest density amount of planets.

The stationary entities are then created. 10 stationary entities will be created excluding the blackhole. The creation process works by generating a row number and then a column number. If that sector does not contain a stationary entity and the sector is not full, then a stationary entity will be created. To determine which stationary object is being created, a random number is generated from between 1 and 3: 1 is a yellow dwarf, 2 is a blue giant and 3 is a wormhole. If the sector contains a stationary entity, new row numbers and column numbers will be generated. The process continues until 10 stationary entities have been created.

An important note is that when each entity is created, a unique id is assigned to it. The id is very important in terms of breaking ties when several movable entities are scheduled to move in the same turn and to keep track of attributes of each entity on the board. The explorer has an id of 0 while the blackhole has an id of -1. Planets are assigned the id of 1 and increasing based on the order of creation, whereas stationary entities are assigned the id of -2 and decreasing based on the order of creation.

8. How to Use the Starter Project

The main goal of the starter code is to provide a way for creating new game boards, including the placement of the entities. **We again stress that this code should be refactored for good design**.

You must use the provided classes `RANDOM_GENERATOR` and `RANDOM_GENERATOR_ACCESS` in your submission so that your output will match the output of the oracle. These classes implement the **singleton pattern** to prevent issues arising from different instances of the generator. You do not have to use other provided classes. If you compile and run the starter code, you will see a printout of the game board with various entities placed in different sectors. Here are the classes in the starter project:

- In the `ROOT` class, we first modify the threshold value. A `GALAXY` object is created (which uses threshold values) and then printed.
    - {`GALAXY`}.`make` invokes the random number generator, meaning that you should only invoke the `make` command when there is a need (i.e., *test* and *play*).
    - Therefore, when you consider adapting the starter code to your ETF project, you should not invoke {`GALAXY`}.`make` in {`ETF_MODEL`}.`make` and {`ETF_MODEL`}.`reset`, because those two commands may not be triggered by *test* or *play*. Again, we do not require you to use class `GALAXY` in your design; it is only classes `RANDOM_GENERATOR` and `RANDOM_GENERATOR_ACCESS` that are required.
- `GALAXY` owns all of the `SECTOR` objects, and is responsible for the creation of them, as well as placing the stationary entities (i.e. Yellow Giants, Blue Dwarf, etc.) into a random SECTOR. Key features include:
    - *make* which first create a grid of `SECTOR` populated with movable entities, the explorer and blackhole. (Please take note of a line in the `SECTOR` populate feature, where the generator is called to initialize the next turn of the entity. It is not used for anything here, but you need it when you actually create the entity rather than the `ENTITY_ALPHABET`.) At the end, 10 stationary entities are added to random sectors.
    - *out* which provide an ASCII output of the abstract state.
- `ENTITY_ALPHABET` provides a way to abstract of each of the entities in the game for the output in the starter code, leaving you to figure out the proper implementation themselves.

- `SHARED_INFORMATION` is a singleton that keeps track of the values that the starter code uses to generate the board (threshold value, board size). This is useful because it upholds the single-choice principle of software design, since the program has a single source to point to and look for certain information, e.g. how many rows should there be on the board.
- `RANDOM_GENERATOR_ACCESS` has a feature called *rchoose* which lets you generate a number in a specified range. More on this feature in the appendix.

9. Game Being Played

After the board has been created, many commands become "available" for the player to use. Commands can be split into two types: valid command uses and invalid command uses. The valid command usage can be further split into two types: commands when validly used, constitute a **turn**, and commands that don't.

Commands not constituting a turn like *status* do not modify the board state, with the exception of *play* and *test* (which set the initial state of the board). In general, valid commands that constitute a turn, as well as valid play and valid test commands, cause the output to give information about the board and the game, whereas the rest of the commands, including invalid uses, cause the output to give information about the game (but not the board). See section *Abstract State: Main Messages* (and the subsequent sections) for more information about the board and game information to display.

For commands that causes a turn to go when validly used (*move*, *wormhole*, *land*, *liftoff*, *pass*), see section *What Happens in a Turn*.

The explorer initially starts at (1,1), has a *life* of 3 and *fuel* of 3 (the maximum amount). Fuel decrease by 1 each time the explorer uses the *move* command successfully. Fuel is gained when in a sector with a star based on the star's luminosity intensity. Life is lost based on interactions with other entities and cannot be gained.

The game is over when

- a planet with life is found,
- the explorer runs out of fuel or life (by blackhole)
- or the user issues an *abort* command. A new game (with a different initial state) can then be started.

10. What Happens in a Turn?

The commands *move*, *wormhole*, *land*, *liftoff* and *pass* are commands that will cause a turn to occur when valid. The commands are considered valid when a game is in play (and not over) and:

- for *move*, the new sector to travel to is adjacent and not full
- for *wormhole*, a wormhole exists at the current sector of the explorer
- for *land*, a planet is attached to a yellow dwarf at the current sector of the explorer that has yet to be landed on (and the explorer is currently not landed on a planet)
- for *liftoff*, the explorer to be landed on a planet attached to a yellow dwarf, but contains no life.

The order of a turn consists of the following:

- Explorer acts; then explorer consumes 1 unit of fuel when it uses a "move" command, whereas the rest of the commands (that causes a turn to occur when valid) do not consume any fuel.
- Fuel is added to the explorer if it is in a sector with a star (where the fuel gain is based on the star's luminosity).
- If explorer *fuel* ≤ 0 or the explorer is in sector (3,3), the explorer dies (removed from the board), but the turn continues (i.e. other entities act) before the game ends.
- The other movable entities then act (i.e. planets).
    - Other movable entities act based on id and a variable *turns_left*.
    - In the order from the lowest id, a movable entity's *turns_left* is decreased by 1. When it hits 0, that movable entity is enabled to act for that turn. See section *Entities (Detailed)* below to see how each type of entities interacts with each other.
        - In general, a planet when acting can be split into three phases:
            - The planet moves to a neighbouring sector (see *movement* in Appendix A).
            - The planet is checked if it is alive after moving (see *check* in Appendix A).
            - After movement to a neighbouring sector, if the planet is still alive, then there are further checks and actions: 1) if there is at least one star in the same sector, then the planet is considered *attached* to stars in that sector; 2) if one of the stars is a yellow dwarf, then the planet may be marked as supporting life (see *behave* in Appendix A). The behave phase will not occur if the entity dies during the check phase.
    - After the entity's action, if they are still alive, a number is again generated between 0..2, again signifying the next turn.

For output, when the commands are invalid, a message indicating that is output (and a turn will not occur). When the command is valid, it depends on if the command causes a turn (and if it is *play* or *test*) or if the command does not cause a turn. For the latter, only a message is output. For the former, a message is potentially outputted along with entity movements and the board. If the game is in test mode, additional information such as mapping of sectors to entities, entities deaths and entities stats.

11. Entities (Detailed)

For a summary of constant and variable attributes of each entity of the game, see **Table 5** and **Table 6**.

SUMMARY OF ENTITIES AND FIELDS

Table 5: Variable fields in entities.

Icon: CHAR	fuel: INTEGER	life: INTEGER	load : INTEGER	actions_ left_until_ reproduction: INTEGER	attached?: BOOLEAN	support_ life?: BOOLEAN	visited?: BOOLEAN	landed?: BOOLEAN	turns_left: INTEGER	death_message: STRING
O										
W										
*										
Y										
E	✓	✓						✓		✓
P					✓	✓	✓		✓	✓

Table 6: Constant fields in entities.

Icon: CHAR	ID: INTEGER	Luminosity: INTEGER	Max_Load: INTEGER	Max_Fuel: INTEGER	Reproduction_ Interval: INTEGER
O	-1				
W	neg				
*	neg	2			
Y	neg	3			
E	0			3	
P	pos				

EXPLORER

The explorer travels the galaxy searching for a planet capable of supporting life. An explorer has the following behavior:

• It is a movable entity. • The explorer gets to choose which action to perform when activation occurs. Please refer to **Grammar** section. • Has maximum fuel of 3 which decreases by 1 each time it moves. It can also take a wormhole, pass, land or liftoff which does not consume any fuel. Fuel is recharged by moving into a sector with a star where it gains fuel equivalent to the star's luminosity value. • Has a life value of three, which is reduced to zero when running out of fuel or entering a region with a black hole). • A life or fuel value of zero ends the game. • Represented by the character 'E'.	

PLANET

Planet is a movable entity. Planets initially have no sign of life associated with them and move to sectors looking for stars. If a star is found in a sector the planet will *remain in "orbit" in that sector*. If the star is a yellow dwarf the planet has a 50 percent chance of developing a life supporting capability. A planet has the following behaviour:

• It is a movable entity • Is destroyed if a blackhole consumes it • Is represented by the character 'P'. **MOVEMENT** • It first moves to a randomly selected neighbouring sector (unless it is already in a sector with a star initially, then it does not move) **CHECK** • If in sector (3,3), it dies **BEHAVE** • If the sector contains a star, the planet remains in that sector (and becomes attached). • If a planet shares a sector with a yellow dwarf, the planet has a 50% chance to support life • If planet is not attached, then reset *turns_left* 0-2.	

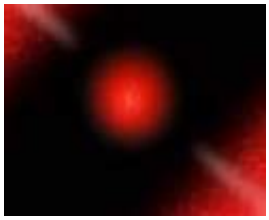
WORMHOLE

A wormhole is a stationary entity. The explorer, if in a sector where there is a wormhole, can use the wormhole to exit that sector and travel to a random sector. A wormhole has the following behaviour:

• It is a stationary entity. • It can take the explorer to a random sector. • It is represented by the character 'W'.	

BLACKHOLE

A super-dense entity that absorbs everything in its sector. *Needless to say, a sector that contains a blackhole contains nothing else.* A blackhole has the following behaviour:

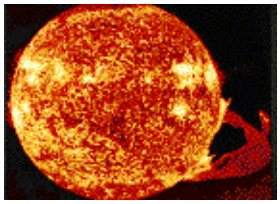
• It devours any entity in its sector. • Is represented by the character 'O'.	

STAR

Stars are special type of stationary entities. Each type of starts has a luminosity intensity, which determines how effectively they can re-power space vehicles. Possible stars include:

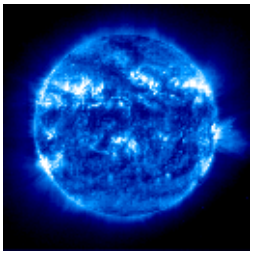
YELLOW_DWARF

A yellow dwarf is a star like our sun. It can have multiple "orbiting" planets, which can sustain life, but only after the planet has gone into the yellow dwarf's orbit. A yellow dwarf has the following behaviour:

• It is a star • It has a luminosity of two. • Is represented by the character 'Y'.	

BLUE_GIANT

A blue giant is a super luminous star– just the thing for recharging those flagging solar fuel cells. A blue giant can have planets in orbit. A blue giant has the following behaviour:

• It is a star • It has a luminosity of five. • Is represented by the character '*'.	

12. Grammar (Formal)

Based on the previous descriptions of the game and entities, the following grammar of types and events will be used to generate a starter ETF project for you to implement:

```
system simodyssey1
-- SimOdyssey Project specifications:

  -- The game is played in test mode using a random generator
  -- with a deterministically generated seed.
  -- (The normal game that is not being implemented in this project
  -- uses a random generator with a non-deterministically
  -- generated seed).

  -- The system consists of a two-dimensional grid of sectors.
  -- Entities are randomly allocated to sectors in the grid,
  -- except for the explorer and blackhole

  -- Movable entities can move through the GALAXY and interact
  -- with other entities.
  -- Stationary entities stay in one place and
  -- interact with the movable entities.
  -- The explorer is a movable entity controlled by the user.

  -- Upon each valid command that constitute a turn
  -- the system shall output the current abstract state of the game
  -- (and also a table representation of the galaxy).

  -- Invalid commands give error messages.

  -- Queries shall display results only, not the system state.

  -- IMPORTANT: Read the provided documentation for more details of system behavior.

-----------------------------------------------------------------------
-- SimOdyssey System Types
-----------------------------------------------------------------------

type DIRECTION = {N, NE, E, SE, S, SW, W, NW}
                  --movement directions

type THRESHOLD = 1..101
                  --values for specifying thresholds in test mode
```

```
-------------------------------------------------------------------
-- SimOdyssey User Commands
-------------------------------------------------------------------
test(p_threshold:THRESHOLD)
   -- Starts a new game in test mode provided game
   -- has not been started yet or is over.
   -- Test mode uses a deterministic random generator and displays
   -- the abstract state of the game.
   -- Allows the setting of threshold values to populate the board initially
   -- (between 1 and 101 non-decreasing), e.g
    -- p_threshold: 40, i.e. generate Planets for 1..39
    -- a random number of 40 to 100 generates no moveable entities
        -- If the random number generator generates a number from
    -- 1 to 100 and the number is in the interval from 1 (inclusive)
        -- to the p_threshold (exclusive) a planet is created. From
        -- p_threshold (inclusive) to 101 (exclusive) generates nothing.
   -- Note that this command will not cause a turn to pass/occur.

play
   -- Starts a new game using test(30)
   -- provided a game has not been started yet or is over.
   -- Play mode displays only the board and key messages as outputs
   -- and not the complete abstract state.

abort
   -- Ends the game prematurely. Only valid when game is
   -- in progress.

move (dir: DIRECTION)
    -- Moves the explorer in a given direction.
    -- A game has to be in progress and the sector
    -- to travel to is not full.
    -- Note that this command will cause a turn to pass/occur.
    -- After the explorer moves, other moveable entities whose clock
    -- time (rest) is zero also act in id order, i.e. 1, 2, ..
    -- In test mode: displays entity actions, abstract state,
    -- then board.

land
    -- Lands the explorer on a planet to check for life on planet.
    -- A game has to be in progress, the explorer is not already
    -- landed and there must be a planet with a yellow dwarf in the
    -- current sector where that planet has not been landed on yet.
    -- If there are multiple planets in this sector, land on the one
    -- that has not been landed on yet with the lowest id.
    -- Note that this command will cause a turn to pass/occur.
    -- After the explorer land, other moveable entities whose clock
    -- time (rest) is zero also act in id order, i.e. 1, 2, ..
    -- In test mode: displays entity actions, abstract state,
    -- then board.

liftoff
    -- Lifts the explorer off a planet.
    -- A game has to be in progress and the explorer is landed
    -- on a planet which also has a yellow dwarf in the same
```

```
    -- sector that cannot support life.
    -- The explorer remains in its quadrant, but can now move.
    -- Note that this command will cause a turn to pass/occur.
    -- After the explorer liftoff, other moveable entities whose clock
    -- time (rest) is zero also act in id order, i.e. 1, 2, ..
    -- In test mode: displays entity actions, abstract state,
    -- then board.

pass
    -- Lets the explorer pass a turn.
    -- Note that this command will cause a turn to pass/occur
    -- and other entities can affect the explorer.
    -- After the explorer pass, other moveable entities whose clock
    -- time (rest) is zero also act in id order, i.e. 1, 2, ..
    -- In test mode: displays entity actions, abstract state,
    -- then board.

wormhole
    -- Tunnels the explorer to a random sector (first open quadrant).
    -- A game has to be in progress and there must be
    -- a wormhole in the current sector.
    -- Note that this command will cause a turn to pass/occur
    -- and other entities can affect the explorer.
    -- After the explorer wormholes, other moveable entities whose
    -- clock time (rest) is zero also act in id order, i.e. 1, 2, ..
    -- In test mode: displays entity actions, abstract state,
    -- then board.

-------------------------------------------------------------------------
-- SimOdyssey Queries
-------------------------------------------------------------------------

status
     -- Displays explorer's energy, life and sector.
     -- Note that this command does not cause a turn to pass/occur
```

13. Abstract State: Main Messages

This section defines what the (abstract) state of a SimOdyssey game. Users interact with your program by entering commands (e.g., move(SE), status, abort). As long as an input command conforms to the syntax (specified in the file simodyssey.definitions.txt), the state changes. Different states will have different output. For example, consider the following abstract state resulted from executing the command move(SE):

```
...
->move(SE)
  state:3.0, mode:test, ok
  Explorer got devoured by blackhole (id: -1) at Sector:3:3
  The game has ended. You can start a new game.
  Movement:
    [0,E]:[2,2,3]->[3,3,2]
    [1,P]:[1,1,2]->[2,1,1]
    [4,P]:[5,2,2]->[1,2,4]
    [10,P]:[2,4,1]->[1,3,3]
    [11,P]:[2,4,2]->[1,4,1]
    [14,P]:[3,1,2]->[2,5,3]
    [15,P]:[2,3,3]->[1,3,4]
    [16,P]:[5,5,2]->[4,4,1]
    [23,P]:[5,4,1]
  Sectors:
    [1,1]->-,-,-,-
    [1,2]->[2,P],[3,P],[-3,*],[4,P]
    [1,3]->[6,P],[19,P],[10,P],[15,P]
    [1,4]->[11,P],-,-,-
    [1,5]->-,-,-,-
    [2,1]->[1,P],-,-,-
    [2,2]->[7,P],[-9,*],-,[5,P]
    [2,3]->[8,P],[9,P],-,-
    [2,4]->-,-,[-10,W],-
    [2,5]->[12,P],[13,P],[14,P],-
    [3,1]->-,-,-,-
    [3,2]->[-6,*],-,-,-
    [3,3]->[-1,O],-,-,-
    [3,4]->-,-,-,-
    [3,5]->[-11,W],-,-,-
    [4,1]->-,-,-,-
    [4,2]->[17,P],[-2,*],-,-
    [4,3]->-,-,-,-
    [4,4]->[16,P],-,-,-
    [4,5]->[-7,W],-,-,-
    [5,1]->[-5,Y],-,-,-
    [5,2]->-,-,-,-
    [5,3]->[20,P],[21,P],[22,P],[-8,Y]
    [5,4]->[23,P],[18,P],-,-
    [5,5]->[-4,W],-,-,-
  Descriptions:
    [-11,W]->
    [-10,W]->
    [-9,*]->Luminosity:5
    [-8,Y]->Luminosity:2
    [-7,W]->
    [-6,*]->Luminosity:5
```

```
  [-5,Y]->Luminosity:2
  [-4,W]->
  [-3,*]->Luminosity:5
  [-2,*]->Luminosity:5
  [-1,O]->
  [1,P]->attached?:F, support_life?:F, visited?:F, turns_left:2
  [2,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [3,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [4,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [5,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [6,P]->attached?:F, support_life?:F, visited?:F, turns_left:1
  [7,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [8,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [9,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [10,P]->attached?:F, support_life?:F, visited?:F, turns_left:2
  [11,P]->attached?:F, support_life?:F, visited?:F, turns_left:2
  [12,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [13,P]->attached?:F, support_life?:F, visited?:F, turns_left:1
  [14,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [15,P]->attached?:F, support_life?:F, visited?:F, turns_left:1
  [16,P]->attached?:F, support_life?:F, visited?:F, turns_left:1
  [17,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [18,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [19,P]->attached?:F, support_life?:F, visited?:F, turns_left:1
  [20,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [21,P]->attached?:T, support_life?:F, visited?:F, turns_left:N/A
  [22,P]->attached?:F, support_life?:F, visited?:F, turns_left:0
  [23,P]->attached?:F, support_life?:F, visited?:F, turns_left:2
Deaths This Turn:
  [0,E]->fuel:2/3, life:0/3, landed?:F,
    Explorer got devoured by blackhole (id: -1) at Sector:3:3
  (1:1)  (1:2)  (1:3)  (1:4)  (1:5)
  ----   PP*P   PPPP   P---   ----
  (2:1)  (2:2)  (2:3)  (2:4)  (2:5)
  P---   P*-P   PP--   --W-   PPP-
  (3:1)  (3:2)  (3:3)  (3:4)  (3:5)
  ----   *---   O---   ----   W---
  (4:1)  (4:2)  (4:3)  (4:4)  (4:5)
  ----   P*--   ----   P---   W---
  (5:1)  (5:2)  (5:3)  (5:4)  (5:5)
  Y---   ----   PPPY   PP--   W---
Explorer got devoured by blackhole (id: -1) at Sector:3:3
The game has ended. You can start a new game.
```

The above example illustrates different parts of an abstract state of SimOdyssey:

- The first line `state:3.0, mode:test, ok` will always be displayed.
  - The state has two number separated by a "." and starts at 0.0.
    - The first number is incremented (by one) after executing a valid command that constitutes a turn (e.g., move, land), or after a valid play or test command. These commands also set the second number to 0.
    - Otherwise, if it is not a command that constitutes a turn (e.g., abort, status), or it is an invalid command, then the second number is incremented by 1.
  - The first line also contains, if in a game, what mode it is in ("test" or "play").
  - There is also indication ("ok" or "error") of validity of the command.
- The next line `Explorer got devoured by blackhole (id: -1) at Sector:3:3` is an example of an optional message. This message depends on the command executed:
  - Invalid command use will cause an error message to be displayed (see section *Abstract State: Error Messages*).
  - Valid status command will provide information on the explorer including whether it is landed or not. Message will also be displayed for abort, landing, liftoff, and winning. See section *Abstract State: Command-Specific Messages*.
  - Messages will also be displayed in the event of an explorer's death; an explorer can die by running out of fuel, or life (blackhole). See section *Abstract State: Death Messages* for more information
- The next line "Movement" appears in both play and test mode when a valid turn has occurred (and when a valid play and test command has occurred).
  - It shows, in order of activation on that turn, the movement of entities including the explorer. The format of the coordinates is in the form of row, column and quadrant. Note the case where the entity was not able to move due to the sector being full.
- Following the "Movement", only appearing in test mode when a valid turn has occurred (and when a valid play and test command has occurred), are the sections called "Sectors", "Descriptions" and "Deaths This Turn".
  - The section "Sectors" shows all the sectors with all the entities inside each sector.
  - In the "Descriptions" section, organized from lowest to highest id, the state of all alive entities (at the end of the turn). A next turn of 0 means it will activate on the next turn.
  - Following that, the "Deaths This Turn" section displays the state of all entities that died that turn in the order of they died in. Note that different entities can die by various means as specified in the Entities (Detailed) section and each method of dying has a different message. See section *Abstract State: Death Messages* for more information.
- The next piece of information appears in both play and test mode when a valid turn has occurred (and when a valid play and test command has occurred) which is the board, in an ASCII representation.
- Only in test mode, following the board is a potential death message from the explorer same as the one in the optional second line to notify the event of a loss. This is also where the game over message would reside.

14. Abstract State: Command-Specific Messages

Below are different messages that may occur when using different commands and they are valid:

Initial Message:
1.
“Welcome! Try test(30)”

status:
1. (Where explorer is not landed.)
“Explorer status report:Travelling at cruise speed at [X,Y,Z]
Life units left:V, Fuel units left:W”
2. (Where explorer is landed.)
“Explorer status report:Stationary on planet surface at [3,1,3]
Life units left:3, Fuel units left:3”

X, Y, Z are the row, column and quadrant respectively of where the explorer is and V and W are the current life and fuel of the explorer.

land:
1. (Where life is found on the planet)
“Tranquility base here - we've got a life!”
2. (Where no life is found on the planet)
“Explorer found no life as we know it at Sector:X:Y” where X and Y are the row and column denoting the sector the explorer landed on a not visited planet attached to a yellow dwarf.

liftoff
1.
“Explorer has lifted off from planet at Sector:X:Y” where X and Y are the row and column denoting the sector the explorer’s location.

abort:
1.
“Mission aborted. Try test(3,5,7,15,30)**”**
Game is over:
1.
“The game has ended. You can start a new game.”

15. Abstract State: Death Messages

Below are different death messages each entity may have:

EXPLORER:
1. (Out of fuel.) “Explorer got lost in space - out of fuel at Sector:X:Y” where X and Y are the row and column where the explorer ran out of fuel.
2. (Death due to blackhole.) “Explorer got devoured by blackhole (id: -1) at Sector:3:3”

PLANET:
1. (Death due to blackhole.) “Planet got devoured by blackhole (id: -1) at Sector:3:3”

16. Abstract State: Error Messages

Below are possible error messages for the second line. Note that valid command message (like landing) are not indicated here. They are listed in order of priority. X and Y can be 1..5

ABORT
1. "Negative on that request:no mission in progress."

LAND
1. "Negative on that request:no mission in progress."
2. "Negative on that request:already landed on a planet at Sector:X:Y"
3. "Negative on that request:no yellow dwarf at Sector:X:Y"
4. "Negative on that request:no planets at Sector:X:Y"
5. "Negative on that request:no unvisited attached planet at Sector:X:Y"

LIFTOFF
1. "Negative on that request:no mission in progress."
2. "Negative on that request:you are not on a planet at Sector:X:Y"

MOVE
1. "Negative on that request:no mission in progress."
2. "Negative on that request:you are currently landed at Sector:X:Y"
3. "Cannot transfer to new location as it is full."

PASS
1. "Negative on that request:no mission in progress."

PLAY
1. "To start a new mission, please abort the current one first."

STATUS
1. "Negative on that request:no mission in progress."

TEST
1. "To start a new mission, please abort the current one first."

WORMHOLE
1. "Negative on that request:no mission in progress."
2. "Negative on that request:you are currently landed at Sector:X:Y"
3. "Explorer couldn't find wormhole at Sector:X:Y"

17. Appendix A. Algorithm/Randomness

1. DEFINITIONS
2. TYPES AND FUNCTIONS
3. BOARD CREATION
4. MAIN CONTROL

1. DEFINITIONS
There are two categories of entities: stationary and movable. We represent each entity with an icon using a single character (e.g., 'O' for Blackhole).

Stationary

O: Blackhole
Y: Yellow Dwarf
*: Blue Giant
W: Wormhole

Movable:

E: Explorer

P: Planet

```
2. TYPES, VARIABLES AND FUNCTIONS
---------------------
-- The game board contains 25 sectors (left to right, top to bottom):
type SECTOR = {(1, 1), (1, 2), ..., (1, 5), (2, 1), ..., (5, 5)}

-- Each sector has four quadrants (left to right):
type QUADRANT = 1..4

fun next_available_quad: FUN[SECTOR, QUADRANT]
    -- return the left-most next available quadrant
    -- require: input sector not full
fun has_stationary: FUN[SECTOR, BOOLEAN]
    -- has_stationary[s], where s is a sector,
    -- returns true if s has a stationary object
    -- false if not
fun is_full: FUN[SECTOR, BOOLEAN]
    -- each sector maps to a boolean
    -- true if each of the 4 quadrants has an entity
fun return_m_ent_low_high: FUN[SECTOR, SEQ[ENTITY]]
    -- return_m_ent_low_high[s], where s is a sector,
    -- returns a sequence of movables that are in s.
    -- Also, the returned sequence must be sorted from lowest to highest id

var next_stationary_id: INTEGER = -1
    -- id to be assigned to the next generated stationary
    -- initial value is -1 and is decremented when necessary

var next_movable_id: INTEGER = 0
    -- id to be assigned to the next generated movable
    -- initial value is 0 and is incremented when necessary

-- Used to generate the board.
-- Refer to {SECTOR}.populate in the starter code.
var planet_threshold: 1..101

type ACTION = {pass, move, wormhole, land, liftoff}
    -- possible action of explorer

fun is_ENT_TYPE(ent:ENTITY):BOOLEAN
    --Replace ENT_TYPE with any entity type
    --Returns true if ent is of type ENT_TYPE
    --e.g., is_planet(ent) returns true if ent is a planet.

rchoose(low:INTEGER, high:INTEGER): INTEGER
        -- random choose (see class RANDOM_GENERATOR_ACCESS in starter code)
    require
        low < high
        low >= 0 and high > 0
    do
        Result:= random integer from the closed interval [low, high]
    end
```

```
3. BOARD CREATION
--------------------
placement
    -- initial distribution of entities on the board (see SECTOR class in starter)
    do
        create the Explorer in sector (1,1) with its id := 0 in its first quadrant
        next_movable_id := next_movable_id + 1
        create the Blackhole in sector (3,3) with its id := -1 in its first quadrant
        next_stationary_id := next_stationary_id - 1
        -- Stage 1: place movables
        across SECTOR (except Blackhole sector (3, 3)) as sector:
            n := rchoose(1,3) -- maximum number of movable entities at current sector
            across 1..n:
                value := rchoose(1,100)
                if value < planet_threshold then
                    create a planet with id := next_movable_id in
                        next_available_quad[sector]
                    turns_left := rchoose(0,2) -- for this planet
                    next_movable_id := next_movable_id + 1
                else
                    -- value exceeds all entity thresholds
                    -- No entity is created.
                end
            end
        end
        -- Stage 2: place 10 stationary entities
        from
            loop_counter := 1
        until
            loop_counter > 10
        loop
            row := rchoose(1,5)
            col := rchoose(1,5)
            -- Put a new stationary to sector (row, col) if it does not already
            -- contains a stationary and it is not full.

            if (not has_stationary[(row,col)]) and (not is_full[(row,col)]) then
                n := rchoose(1,3)
                if n = 1 then
                    create a yellow dwarf with id := next_stationary_id in
                        next_available_quad[(row,col)]

                elseif n = 2 then
                    create a blue giant with id := next_stationary_id in
                        next_available_quad[(row,col)]

                elseif n = 3 then
                    create a wormhole with id := next_stationary_id in
                        next_available_quad[(row,col)]

                end
                next_stationary_id := next_stationary_id - 1
                loop_counter := loop_counter + 1
            end
        end
    end
```

4.MAIN CONTROL

After any valid explorer **action** (pass, move, wormhole, land or liftoff), a turn will occur. Being valid means that there are no error messages when the command (e.g., pass) is being used. For each planet, if their turns_left is 0, that entity will act; otherwise their turns_left decreases by 1. The order in which each planet acts is from the lowest to highest id. Dead entities are removed from the board and will not act. When an explorer dies, its life is set to 0.

```
turn(action: ACTION)
        -- The explorer performs `action`, if valid, then other movables act.
    do
        if action causes an error (see Abstract State: Error Messages) then
            -- A turn does not occur because of an invalid `action`
        else
            act(action) --explorer performs an action (based on user input)
            check(explorer) -- ensures explorer is alive after moving
            -- After the explorer acts, the game may end either because
            -- 1) the explorer dies; or 2) it lands on a planet that supports life.
            -- For 1) the across loop below continues; For 2) skip the across loop.
            across movable entities (except explorer) by ids in ascending order as
                entity:
                    -- When an entity dies, remove it from board
                    -- If it is not an explorer, also remove entity from the movable
                    -- entities being iterated over.
                if entity.turns_left = 0 then
                    --special case for planet
                    if is_planet(entity) and ∃star at entity.sector then
                        entity.attached? := true
                        if ∃ yellow dwarf at entity.sector then
                            num := rchoose(1,2) -- num = 2 means life
                            if num = 2 then
                                entity.support_life? := true
                            end
                        end
                    else
                        movement(entity)
                        check(entity)
                        if entity did not die then
                            behave(entity)
                            entity.turns_left := rchoose(0,2)
                        end
                    end
                else
                    entity.turns_left := entity.turns_left - 1
                end
            end
        end
    end
```

```
act(action:ACTION)
    --explorer performs an action (based on user input)
    do
        if action = pass then
            -- do nothing
        elseif action = move(dir) then
            -- let new_sector be the sector pointed to by `dir`
            place explorer in next_available_quad[new_sector]
        elseif action = wormhole then
            wormhole(explorer)
        elseif action = land then
            explorer.landed? := true --see Table 5
            if planet being landed on supports life then
                -- game ends
                -- movables will not act
            end
        elseif action = liftoff then
            explorer.landed? := false
        end
    end

wormhole(ent:ENTITY)
    -- Explorer may use a wormhole to move.
    do
        if is_explorer(ent) then
            from
                added := false
            until
                added
            loop
                temp_row :=  rchoose(1,5)
                temp_column := rchoose(1,5)
                if (not is_full[(temp_row,temp_column)]) then
                    place ent in next_available_quad[(temp_row,temp_column)]
                    --where if wormhole to same location, quadrant of entity
                    --using wormhole counts as an available quadrant
                    added := true
                end
            end
        end
    end

movement(ent:ENTITY)
    --movement by movable entities to neighbouring sector
    do
        if is_planet(ent) then

            --1 means N, 2 means NE, 3 means E, 4 means SE, ... 8 means NW
            direction := rchoose(1,8)
            if not is_full(ent destination sector) then
                place ent in next_available_quad[(new sector)]
            end
        end
    end
```

```
check(ent:ENTITY)
    -- ensures entity is alive after moving
    do
        if is_explorer(ent) and ent moved from one sector to another sector without
            using a wormhole for movement successfully then
            ent.fuel := fuel -1
        end

        if is_explorer(ent) and ∃ star at ent.sector then
            ent.fuel := min(ent.fuel + star.luminosity, ent.max_fuel)
        end

        if is_explorer(ent) and ent.fuel = 0 then

            ent dies
        end

        if ∃ blackhole at ent.sector then
            ent dies
        end
    end

behave(ent:ENTITY)
        --planet behaviour
    do
        if is_planet(ent) then
            if ∃ star at ent.sector then
                ent.attached? := true
                if ∃ yellow dwarf at ent.sector then
                    num := rchoose(1,2)
                    if num = 2 then
                        ent.support_life? := true
                    end
                end
            else
                ent.turns_left := rchoose(0, 2)
            end
        end
    end
```

18. Appendix B: Additional Debug Help - toggle_RNG_out

The oracle also supports an additional command called “toggle_RNG_out”. **You DO NOT have to implement this command.** We will not test it, we promise. The purpose of this command is to help you debug RNG (Random Number Generator, which is given to you in the starter project) usage. The description of this command in the grammar is:

```
--toggle_RNG_out
    -- Toggles the RNG out flag.
    -- When enabled, if any commands result in the board being output,
    -- then additional information about the RNG usage is appended
    -- at the very end.
    -- There will be a list with entities in the form of (W->X:[Y,Z]).
    -- If the RNG is not used that turn, then "none" will
    -- be appended for that section.
    -- The order of the tuples (left-to-right then up-to-down)
    -- determine the order of usage of the RNG.
    -- W represents the entity using the RNG (where it can be
    -- in the form of any movable entity's alphabet or "G"
    -- denoting the galaxy).
    -- The RNG generates a number from Y to Z and X is the
    -- actual value it generated.
```

This command, when used, does not count as a turn. What it does though is that it toggles a flag which we will call RNG_out. The output after using this command will indicate whether this flag is set to true or false. It can be used anytime. When the flag is set to true, whenever a command is used that causes the board to be displayed, at the very end, additional information about the usages of the RNG that turn will also be displayed. (Nothing different happens if the flag is set to false.) The form of the additional information is in (W→X:[Y,Z]). W denotes which entity used the RNG by their alphabet representation. W can be all movable entities or “G” which stand for galaxy and is only used when setting up the board initially. If an explorer uses a wormhole, “W” would be “E” when finding the coordinates for the wormhole. If a planet’s turn is resetting, “W” would be “P”. Y and Z denotes the range the RNG is being asked to generate a number for; from Y to Z inclusive. X is the actual number generated.

19. Appendix C. Amendments

Clarifications and revision on the specification will be made available here.