Team member: Jason Hui prism login: jason75

Requirements

Our code simulates a game called SimOdyssey, in which the player or explorer has the goal of finding life within the galaxy. The player will only be able to control the movement of the explorer and during each turn, there are different valid and invalid actions that the player can execute. For example, a valid action would be a player can always pass a turn or show the status if the player is in a game. Invalid action would be moving when the player is not in a game or land when there is no planet in the same sector.

There are 4 quadrants within each sector of the galaxy. And within those 4 quadrants of the sector, many different entities can spawn and occupy the space. Entities can be generalized to two main types- movable and stationary. Movable includes entities, such as the explorer, asteroid, benign, malevolent, and janitaur. As mentioned, all movable except the explorer will not be able to be controlled by the player, meaning they will act automatically. However, this does not imply that all other movable entities' behavior is completely random, there are priorities within their behavior. For example, an asteroid will prioritize taking a wormhole for movement if a wormhole is in the same sector. Different movable entities will also behave differently meaning they interact differently with different entities. Without the presence of a benign, a malevolent can attack the explorer if they are in the same sector. However, if a benign is present, then the benign will destroy the malevolent instead. Also some movable entities such as benign, janitaur, and malevolent can reproduce automatically while in game.

The other type of main entity is stationary which includes wormhole, blackhole, and stars such as blue giants and yellow dwarf. Each stationary entity has property that makes them unique, the wormhole can travel a moving entity to a random sector, a blackhole will devour any object in the same sector, and stars have different luminosity.

Finally, to represent there are two ways to start a game, either by using tests or play. Test mode will display more information about the state of the game such as the statistics of each entity in the game.

Team member: Jason Hui prism login: jason75

Bon Diagram Overview

The bon diagram shows two main clusters, the simodyssey2 cluster and the galaxy cluster. The galaxy cluster contains every class necessary to represent the game board, which is a galaxy. Within the galaxy cluster, there is a subcluster called entities, which includes every important class that is required to represent every different entity in the game. Inheritance is used here to prevent superman class and thus follow the cohesion principle, since each entity subclass only stores information relevant to them. This allows the reuse of code in other classes. For example, if the galaxy needs to update a certain state or display a state of a certain entity, they can reuse code defined in their entity class.

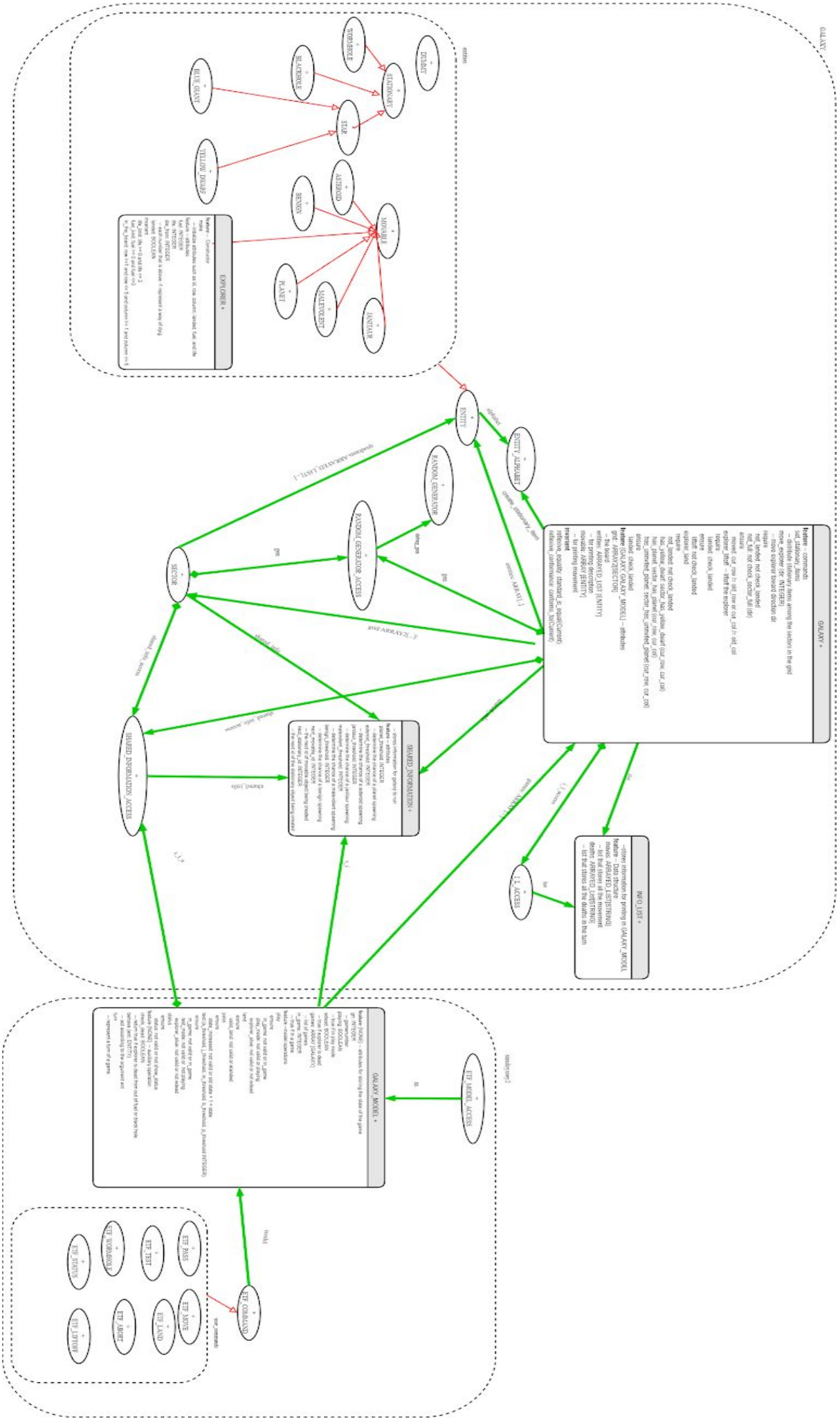
The galaxy class and sector class represent the game board that holds all the entities and have the operations to update the locations of the entities. Shared_information and Information_list are two separate classes because they hold information for different purposes. The former holds information for Galaxy class to operate while the latter holds information for Galaxy_model class to print information out. They are separate because I want both classes to be cohesive and not store any relevant information.

In the simodyssey2 cluster, the Galaxy_Model class represents the game itself rather than just the game board. This is what the user will mostly be concerned about. Since operations that are called by the user are all defined in this class. As you can see from the bon diagram, there are some auxiliary features that are only available within Galaxy_Model class because those features are there to help model operations within that class. Such as the turn feature, which is required for pass, move, wormhole, and other operations, and not required for other classes. So it is hidden to other classes. There is also an ETF_Command sub cluster, which holds all the classes that the user can operate with.

The only downside to this design is that my design does not follow the single choice principle, so whenever we have to extend the code, such as adding a command, we have to first add a class into the ETF_Command sub cluster, then we have to define it in the GALAXY_Model and we also have to create subsequent functions required in the galaxy to update the game board if necessary. So my code has low extendibility, but my code is highly cohesive, since no class is storing information they don't need.

** my bon diagram is too huge to be fitted into one page in word so I put another copy of the same bon diagram in the docs folder

Team member: Jason Hui prism login: jason75



Team member: Jason Hui prism login: jason75

Table of Module

		abstract/concrete	Responsibility	Secret	Alternative
1	ARRAYED_LIST [GALAXY]	concrete	Holds a sequence of games (galaxy)	None	LINKED_LIST
1.1	GALAXY	concrete	Represents the game board	The generation of galaxy is only allowed when it is added into an arrayed list in GALAXY_MODEL	None
2	ARRAY[ENTITY]	concrete	Sequence of entities currently in the game	Only available in the galaxy class to be used for sorting so it can be stored in an array list	None
2.1	ARRAYED_LIST [ENTITY]	concrete	Same as 2	None	LINKED_LIST
2.1.1	ENTITY	abstract	Holds all the basic attributes for an entity	None	None
3	ARAY[ENTITY]	concrete	Holds all the movable entities that currently exist	None	None
4	INFORMATION_ LIST	concrete	Holds all information required to print in test mode such as movements and deaths	Singleton access by information_acces s	None

Team member: Jason Hui prism login: jason75

Expanded Description of design decisions

Galaxy_Model class represents the game and the interface used by the user to operate the game, so it has ARRAYED_LIST[GALAXY}, which is an array list of galaxies, each representing a different game board. The creation of the galaxy is only allowed in Galaxy_Model class because users should not be able to create a game or galaxy in other classes by calling play or test. Using an array to access the list of games is very fast because accessing elements by index takes O(1) time in an array.

In the Galaxy class, there are two data structures that store entities that currently exist in the game. These lists are sorted by ids, so when features that operate the model are called, it can interact with entities with the lowest ids first. This is especially important for visiting planets or destroying entities. The galaxy also has an attribute called ilist which is an Informatino_List object. This holds all the information required for printing information in test mode such as movements and deaths. The list is cleared after getting printed every turn to make sure no information from the previous turn is printed. Using an array_list and other structures won't affect the performance too much since a lot of model operations require traversing through the whole movable array. Which is O(n), n ebing the number of movable entities currently on the board.

Sector has an array called quadrants of max size 4, this represents the 4 spaces available that can hold an entity. The data structure used in sector is also quite efficient since, quadrants will only have a max size of 4 so using an array is pretty suitable

My design and architecture is not the most extendible since if we want to add a new class of entity or new action. We will have to add it in multiple places such as GALAXY and GALAXY_Model. One way to eliminate this is to define features such as reproduce, or move within the each entity subclass. So instead of repeating using a control structure such as if attached {ASTEROID}... then elseif attached {BENIGN} … in multiple places, we can just use ent.reproduce or ent.behave. This is similar to the state design pattern where we use the state as an object that calls the method instead of using the state as a parameter of the method. However, if we define behave, reproduce, and other methods in each entity subclass, I notice that we will violate cohesion since the entity has to access galaxy to reproduce, and the entity subclass is not supposed to have any information other than qualities of the entity. So I did not use the state design principle to eliminate the violation of the single choice principle

Team member: Jason Hui prism login: jason75

Contracts

The module that has the most contracts is the galaxy cluster. The entity subclass has a lot of contracts as invariants that make sure all the entities created are correct. Invarints, such as correct_id makes sure movable entities will have positive id while stationary entities will have a negative id. Other invariants such as life_limit, fuel_limit, and turn_liimit makes sure they won't have an attribute that exceeds the limit or drops below the limit. Finally in_the_board makes sure their locations are within the board.

The more interesting contracts are written in the GALAXY class where most operations for the game are happening. The best example to explain is explorer_land which is a feature that lands the explorer in game. This has 4 preconditions, first, the explorer cannot be landed which is checked by using check_landed. Then it has to check if there is a yellow dwarf, a planet, and finally if there is a planet that is not visited. These all use auxiliary query to check if these conditions are met. And land operation will only execute if all of these conditions are met. After the execution of the feature, its precondition checks if the execution call was successful by checking if the explorer is landed. Liftoff is very similar to land except with less preconditions and the postcondition checks if the explorer has lifted off (not landed). Explorer_move is an operation with a different contract. It checks if the destination sector is full and it also checks if the explorer is moving. After the execution of the feature, the post condition checks if it has changed its position. Wormhole first checks if there is a wormhole present in the sector then the next precondition which is not is_explorer (ent) or not check_landed just means that if the entity is an explorer then it should not be landed. Notice that if the entity is not an explorer then the statement will be true.

The Sector has an invariant contract that limits the size of the quadrants to be less than or equal to 4. This makes sure that every sector can hold at most four entities. And the feature put_quadrants has a post condition that simply confirms that the argument entity is actually put into the quadrants.

In Galayx_Model, a lot of post conditions also make sure that the feature called was successfully executed similar to Galaxy class. For example, after a successful execution of play, it will make sure that if it was a valid call, the game will be in play mode, and in_game will also turn to true. Similarly, after calling a valid pass call, it will make sure that the state has increased. The land operation here is similar to the one in GALAXY, in which

Team member: Jason Hui prism login: jason75

Testing

test	Description	passed
at01.txt	Tests winning condition in test mode	yes
at02.txt	Tests winning condition in test mode	yes
at03.txt	Test losing by out of life by malevolent in test mode	yes
a04.txt	Testing explorer destroyed by asteroid in test mode	yes
at05.txt	Testing death by out of fuel in test mode	yes
at06.txt	Used to test some bugs regarding wormhole and move	yes
at07.txt	Testing death by blackhold	yes
at08.txt	Test spawning alot of entities and pass a lot of turns	yes
at09.txt	Testing death of other entities by blackhole	yes
at10.txt	Tests death by out of fuel in play mode	yes

Team member: Jason Hui prism login: jason75

APPENDIX

```
note
	description: "Model that represents the game SimOdyssey2"
	author: "Jason Hui"
	date: "$Date$"
	revision: "$Revision$"

class interface
	GALAXY_MODEL

create {ETF_MODEL_ACCESS}
	make

feature -- model operations

	default_update
			-- Perform update to the model state.

	abort
			-- abort the current game
		ensure
			aborted: not valid or not in_game

	land
			-- lands the player
		ensure
			explorer_landed: not valid or elanded

	liftoff
			-- lifts off the player

	move (dir: INTEGER_32)
			-- move player towards the direction dir

	pass
			-- pass a turn
		ensure
			state_increased: not valid or old state + 1 = state

	play
			-- create a game in play mode
		ensure
			play_mode: not valid or playing
			in_game: not valid or in_game
			explorer_alive: not valid or not edead

	status
			-- display status
		ensure
			status: not valid or show_status

	test (a_threshold: INTEGER_32; j_threshold: INTEGER_32; m_threshold: INTEGER_32;
b_threshold: INTEGER_32; p_threshold: INTEGER_32)
		ensure
			test_mode: not valid or not playing
			in_game: not valid or in_game
			explorer_alive: not valid or not edead

	wormhole

	reset
			-- Reset model state.

feature -- queries

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

	print_sectors: STRING_8
```

```
		print_movement: STRING_8

		print_descriptions: STRING_8

		print_death: STRING_8

		print_test: STRING_8

		print_rng: STRING_8

end -- class GALAXY_MODEL
note
		description: "Galaxy represents a game board in simodyssey."
		author: "JH"
		date: "$Date$"
		revision: "$Revision$"

class interface
		GALAXY

create {GALAXY_MODEL}
		make

feature -- attributes

		grid: ARRAY2 [SECTOR]
				-- the board

		gen: RANDOM_GENERATOR_ACCESS

		shared_info_access: SHARED_INFORMATION_ACCESS

		shared_info: SHARED_INFORMATION

		movable: ARRAY [ENTITY]
				--for movement

		entities: ARRAYED_LIST [ENTITY]
				-- for description

		ilist: INFO_LIST

		cur_row: INTEGER_32
				-- new location of player

		cur_col: INTEGER_32
				-- new location of player

feature --commands

		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

		move_explorer (dir: INTEGER_32)
				-- moves the exploerer toward direaction dir
			require
				not_landed: not check_landed
				not_full: not check_sector_full (dir)
			ensure
				moved: cur_row /= old_row or cur_col /= old_col

		wormhole (ent: ENTITY)
				-- Entity ent will use wormhole to teleport to a location
			require
				has_wormhole: sector_has_wormhole (ent.row, ent.column)
				explorer_not_landed: not is_explorer (ent) or not check_landed
```

Team member: Jason Hui prism login: jason75

```
		explorer_land: INTEGER_32
				-- land exploerer on the unvisited planet with the lowest id
			require
				not_landed: not check_landed
				has_yellow_planet: sector_has_yellow_dwarf (cur_row, cur_col)
				has_planet: sector_has_planet (cur_row, cur_col)
				has_unvisited_planet: sector_has_unvisited_planet (cur_row, cur_col)
			ensure
				landed: check_landed

		explorer_liftoff
				-- liftoff the explorer
			require
				landed: check_landed
			ensure
				liftoff: not check_landed

feature -- query

		is_explorer (ent: ENTITY): BOOLEAN
				--check ifent is of type explorer

		check_sector_full (dir: INTEGER_32): BOOLEAN
				--returns true if the target sector is full

		sector_full (row: INTEGER_32; col: INTEGER_32): BOOLEAN
				-- return true if grid[row, col].quadrant is full

		sector_has_planet (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_unvisited_planet (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_blackhole (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_wormhole (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_yellow_dwarf (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_star (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		sector_has_benign (row: INTEGER_32; col: INTEGER_32): BOOLEAN

		check_landed: BOOLEAN
				--returns true if explorer is landed

		out: STRING_8
				--Returns grid in string form

end -- class GALAXY

note
		description: "Represents a sector in the galaxy."
		author: "JH"
		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
```

Team member: Jason Hui prism login: jason75

```
                        --      contents: ARRAYED_LIST [ENTITY_ALPHABET] --holds 4 quadrants

        quadrants: ARRAYED_LIST [ENTITY]

        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 intially
stored in the
                        -- sector. The component may be a planet or nothing at all.

feature -- Queries

        is_dummy (e: ENTITY): BOOLEAN
                        -- check if entity e is a dummy

        print_sector: STRING_8
                        -- Printable version of location's coordinates with different
formatting

        prints: STRING_8

        is_full: BOOLEAN
                        -- Is the location currently full?

        has_star: BOOLEAN

        has_wormhole: BOOLEAN

        has_explorer: BOOLEAN

        has_yellow_dwarf: BOOLEAN

        quadrants_full: BOOLEAN
                        --true if full

        has_stationary: BOOLEAN
                        -- returns whether the location contains any stationary item

        last_entity_location: INTEGER_32
Invariant
        correct_quadrants_size: quadrants.count <= 4


end -- class SECTOR
```