EECS3311-W20 — Project Report

Submitted electronically by:

|  |  |  |  |
| --- | --- | --- | --- |
| **Team members** | **Name** | **Prism Login** | **Signature** |
| Member 1: | Joseph Emanuele | joeycs | J. E |
| Member 2: | Tamal Alexander | tamal | T. A |
| \*Submitted under Prism account: | | joeycs | J. E |

**Contents**

[1.](#_30j0zll) Requirements for simodyseey - tamal 2

[2.](#_1fob9te) BON class diagram overview (architecture of the design) - joey 7

[3.](#_3znysh7) Table of modules — responsibilities and information hiding - tamal 8

[4.](#_2et92p0) Expanded description of design decisions - joey 9

[5.](#_tyjcwt) Significant Contracts (Correctness) - joey 10

[6.](#_3dy6vkm) Summary of Testing Procedures - tamal 11

[7.](#_1t3h5sf) Appendix (Contract view of all classes) - tamal 12

# Requirements for Project - Simodyssey

Our clients supplied us with the following requirements for the game Simodyssey: The topic at hand is game development. This space exploration game will allow a player to choose between two playing modes. The first mode is referred to as “test”, this mode provides a more detailed outlook on the current state of the galaxy and as well as, the amount and type of entities generated. This entity creation can be influenced by the number that is passed when the test mode is created. The higher the number implies that the probability of generating more planets increases. The second mode is referred to as “play” , this mode will provide less information about the current state of the galaxy and the number of entities that are initially created cannot be influenced.

Within both modes, random numbers are generated in a sequential manner that is always the same and maintained, but changes often from “play” to “play” mode. After a game has started, a 5-by-5 galaxy is created. This would also place an explorer entity at quadrant (1,1) and a blackhole entity at (3,3). The other entities that are stationary and movable are randomly placed in the galaxy. Moreover, these entities are placed in sectors within the galaxy; each sector can contain a maximum of four quadrants. This means that each sector can only hold a maximum of four entities ( one entity per quadrant).

After the galaxy is created, the player would be able to issue commands to move the explorer through the galaxy. Some of these commands would be considered as a turn, which may then cause some of the movable entities to shift their position in the galaxy. However, other commands would not modify the entities within the galaxy but may just display the status of the explorer. The game can end in a few ways, such as; when the explorer runs out of fuel, lives, or when a planet with life is found or when the game is aborted. Once a game has ended a new game can be started.

The document in the appendix will provide further details on the user interface grammar. Also, the acceptance tests in the report will provide further details on the input and output status of the game operating in console mode.

# BON class diagram overview (architecture of the design)

Jjjjjj

The design of Sim Odyssey, at a high level of abstraction, consists of two main components: the model cluster and the entities cluster.

The model cluster contains classes pertinent to the game itself. These classes manipulate the state of the galaxy, the grid in which the game is facilitated, and produce output which provides useful visual feedback to the player regarding the state of the game. Specifically, ETF\_MODEL uses STRING attributes to update the game’s grid representation and messages while accepting a variety of user commands, GAME keeps track of the behaviour of entities and outputs the state of the galaxy, and the GALAXY and SECTOR classes organize entities into data structures in order to inform the GAME class about where entities should be displayed and how they should interact with each other.

The entities cluster contains a hierarchy of classes related by inheritance which classes in the model cluster instantiate as objects to compose the contents of the galaxy and its sectors. The purpose of this hierarchical organization of classes is threefold: firstly, the design takes advantage of code reuse by providing different types of entities, which can be grouped according to their required functionalities, with common features and attributes existing in deferred parent classes. Second, the design grants simplicity when implementing features in which it is required that only particular types of entities be manipulated. With the use of dynamic binding, it is possible to check if a context entity inherits from a particular parent class, providing access to features and attributes specific to this type and its children. This functionality allows for elegant implementations of features which can be applied to any child of the entity class, preventing the need for additional classes or features in the model cluster which are specifically implemented to operate on different types of entities. Finally, with the power of multiple inheritance, the hierarchy is expandable, allowing for the addition of entity subclasses inheriting from a combination of existing or new classes depending on the new entity’s desired behaviour.

To justify the design decisions discussed above, the following scenario will be considered: a programmer wants to create a new type of entity called SWARM, a fleet of insects who reproduce at a high rate and destroy any forms of life in their path. This new class could simply inherit from REPRODUCING\_ENTITY to make use of code which creates clones, CPU\_ENTITY to be considered in features which use random number generation to generate behaviour, and MOVABLE\_ENTITY to allow it access to features and attributes used for movement. Now, all the programmer must do is implement any deferred features such as set\_behaviour, inherited from CPU\_ENTITY, and their new entity should act as desired. The programmer can take advantage of existing logic structures in the model cluster to generate SWARM objects’ placement in sectors, movement throughout the galaxy, and interaction with other entities

# Table of modules — responsibilities and information hiding

Ask jackie about this

|  |  |  |  |
| --- | --- | --- | --- |
| 1.1 | ETF\_MODEL | **Responsibility**: Handles all the user commands for the game | **Alternative**: none |
| Concrete | **Secret** : Implemented by initializing each attribute to a default value of the current model |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.2 | GAME | **Responsibility**: see ETF\_MODEL | **Alternative**: none |
| Concrete | **Secret:** implemented via Arrays, entities that moved this turn and entities that died this turn |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.2.1 | GALAXY | **Responsibility**: see GAME | **Alternative**: none |
| Concrete | **Secret:** none |

|  |  |  |  |
| --- | --- | --- | --- |
| 1.2.1.1 | SECTOR | **Responsibility**: see GALAXY | **Alternative**: none |
| Concrete | **Secret:** none |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | SHARED\_INFORMATION\_ACCESS | **Responsibility**: provides a singleton access for shared information | **Alternative**: none |
| Abstract | **Secret:**  none |

|  |  |  |  |
| --- | --- | --- | --- |
| 2.1 | SHARED\_INFORMATION | **Responsibility**: contains attributes that are constants, which are used to generate stationary and movable entities | **Alternative**: none |
| Concrete | **Secret:** implemented via linked lists that stores the movable entities and rng usage |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | ENTITY\_COMPARATOR | **Responsibility**: compares the entities based on their id | **Alternative**: none |
| Concrete | **Secret**: none |

# Expanded description of design decisions

*Only for the most important module in your design.*

*What alternative designs were considered and rejected based on the criteria of reliability, simplicity, and maintainability?* The design is maintainable if it exhibits conceptual integrity that defines the key abstractions so that designers and programmers can reason about the system you describe and predict its behaviour. Software developers reading your SDD should be able to grasp your design without having to read thousands of lines of code. This will make you system extendible and re-usable.

* discuss entities cluster and inheritance hierarchy
* old design: reproducing entities inherit from effective cloneable class as well as movable entity, all other movable entities simply inherit from movable entity
* new design: movable entities inherit from a combination of reproducing entity, cpu entity, sentient entity depending on their needed functionality
* why? needed shared features/attributes from reproducing entities, class needed to be deferred as having a make feature did not make sense in the design
  + reproducing entities were already effective classes with make features

# Significant Contracts (Correctness)

(only for the module with the most significant contracts)

.

# Summary of Testing Procedures

The following table shows a variety of acceptance tests that were performed to meet the requirement specification of the client.

|  |  |  |
| --- | --- | --- |
| **Test file** | **Description** | **Passed** |
| *at001.txt* | when the explorer wins in play mode | ✓ |
| *at002.txt* | When the explorer wins in Test mode | ✓ |
| *at003.txt* | When the explorer loses by running out of fuel | ✓ |
| *at004.txt* | When explorer passes through a wormhole and then tries to land | ✓ |
| *at005.txt* | When the explorer gets destroyed by an asteroid | ✓ |
| *at006.txt* | When the explorer wins with multiple movement and passes | ✓ |
| *at007.txt* | Attempting to start another game when the user is already in a game | ✓ |
| *at008.txt* | Starting and aborting multiple games using the commands Test and abort | ✓ |
| *at009.txt* | Using the command pass x20 | ✓ |
| *at010.txt* | Moving the explorer before a game has started | ✓ |
| *at011.txt* | Checking the position of the explorer after passing the boundaries of the board | ✓ |
| *at012.txt* | Checking the position of the explorer after moving pass the boundaries of the board (movement West) | ✓ |
| *at013.txt* | Landing on a planet then lifting off | ✓ |
| *at014.txt* | When the Explorer get devoured by a blackhole | ✓ |
| *at015.txt* | When the Explorer lands in a sector with two planets and a yellow dwarf | ✓ |
| *at016.txt* | Checking the status of the Explorer | ✓ |
| *at017.txt* | When explorer and multiple astroids die | ✓ |
| *at018.txt* | when explorer tries to move to a sector that is full | ✓ |
| *at019.txt* | When planets and janitaur gets devoured by blackhole | ✓ |
| *at020.txt* | when asteroids get imploded by janitaur | ✓ |
| *at021.txt* | Checking if the position of the explorer remains consistent after multiple pass and move commands | ✓ |
| *at022.txt* | Scenario when the explorer moves in multiple directions, attempting to land on a planet | ✓ |
| *at023.txt* | Scenario when a game is recreated with a very low threshold in test mode | ✓ |
| *at024.txt* | Scenario when a game is created with a very high threshold in test mode | ✓ |
| *at025.txt* | Scenario when the user starts and ends a game multiple times while allowing multiple passes to occur | ✓ |
| *at026.txt* | Scenario when the explorer navigates through the galaxy creates with a low threshold | ✓ |
| *at027.txt* | Scenario when the asteroid threshold is set to a lower number compared to the other movable entities | ✓ |
| *at028.txt* | Scenario when the explorer tries to navigate through the galaxy with the aim of getting destroyed by an asteroid | ✓ |
| *at029.txt* | Scenario that checks the status multiple times after the initial play mode is executed | ✓ |
| *at030.txt* | Checking status command correctness in multiple scenarios | ✓ |
| *at031.txt* | Scenario when the benign, explorer, and malevolent are in the same sector | ✓ |
| *at032.txt* | Scenario when the asteroid, explorer, and janitaur are in the same sector | ✓ |

(b) Provide a screen shot of the *ESpec* unit tests that you ran. Ensure that the test comments are descriptive.

# Appendix (Contract view of classes)

1. The Game class ( GAME.e ) looks like this :

|  |
| --- |
| note  description: "Summary description for {GAME}."  author: ""  date: "$Date$"  revision: "$Revision$"    **class** interface  GAME    **create**  make    **feature** *-- attributes*    died\_this\_turn: ARRAY [MOVABLE\_ENTITY]    galaxy: GALAXY    game: INTEGER\_32    info: SHARED\_INFORMATION\_ACCESS    mode: STRING\_8    moved\_this\_turn: ARRAY [MOVABLE\_ENTITY]    **feature** *-- commands*    advance\_turn  *-- Allow all movable entities to move once*  *-- Entities' behaviours are determined*  *-- Entities perform secondary actions if applicable*  **require**  galaxy\_populated: **not** info.Shared\_info.entities.is\_empty  galaxy\_has\_movable: across  info.Shared\_info.entities **is** entity  some  attached {MOVABLE\_ENTITY} entity  **end**  **ensure**  dead\_entities\_removed: across  (**old** info.shared\_info.entities.twin) **is** entity  all  (attached {MOVABLE\_ENTITY} entity **as** m\_e **and** **then** m\_e.is\_dead) **implies** **not** info.shared\_info.entities.has (m\_e)  **end**    check\_entity (l\_entity: MOVABLE\_ENTITY)  *-- Check the state of the given movable entity*  *-- Check if the entity has used fuel, gained fuel, or died if applicable*  **ensure**  entity\_died: l\_entity.is\_dead **implies** died\_this\_turn.has (l\_entity) **and** **not** l\_entity.death\_message.is\_empty    end\_game  *-- End the current game, resetting the state of the galaxy*  **require**  in\_game  **ensure**  game\_over: mode.is\_empty    move\_entity (l\_entity: MOVABLE\_ENTITY; dir: INTEGER\_32): BOOLEAN  *-- Move given movable entity in given direction*  **require**  entity\_exists\_in\_galaxy: info.Shared\_info.entities.has (l\_entity)  entity\_alive: **not** l\_entity.is\_dead  **ensure**  successful\_move: **not** l\_entity.failed\_to\_move **implies** (l\_entity.sector.row /= l\_entity.prev\_sector\_row **or** l\_entity.sector.column /= l\_entity.prev\_sector\_col **or**l\_entity.pos /= l\_entity.prev\_sector\_pos)    new\_game (m: STRING\_8; a\_threshold, j\_threshold, m\_threshold, b\_threshold, p\_threshold: INTEGER\_32)  *-- Re-initialize game with given game mode and thresholds while keeping track of the current game number*  **require**  valid\_mode: m ~ "test" **or** m ~ "play"  **ensure**  game\_incremented: game = **old** game + 1  mode\_set: mode ~ m    warp\_entity (l\_entity: SENTIENT\_ENTITY)  *-- Move given entity to a random location in the galaxy*  **require**  entity\_exists\_in\_galaxy: info.Shared\_info.entities.has (l\_entity)  entity\_alive: **not** l\_entity.is\_dead  **ensure**  successful\_move: **not** l\_entity.failed\_to\_move **implies** (l\_entity.sector.row /= l\_entity.prev\_sector\_row **or** l\_entity.sector.column /= l\_entity.prev\_sector\_col **or**l\_entity.pos /= l\_entity.prev\_sector\_pos)    **feature** *-- queries*    check\_move (l\_entity: MOVABLE\_ENTITY; dir: INTEGER\_32): BOOLEAN  *-- Determines if moving the given entity in given direction is a valid action*    find\_entity (e: ENTITY): detachable ENTITY  *-- Attempts to return the given entity as it exists in the galaxy*  *-- Uses entity comparator to determine if an object equivalent entity exists in the galaxy*    get\_game: INTEGER\_32  *-- Returns the current game number*    get\_mode: STRING\_8  *-- Returns the current game mode*    in\_game: BOOLEAN  *-- Checks if a game is in progress*    out: STRING\_8  *-- New string containing terse printable representation*  *-- of current object*    **invariant**  valid\_game: game >= 0    **end** *-- class GAME* |

Figure 1: Shows the contract view of the GAME class

1. The Galaxy class ( GALAXY.e ) looks like this :

|  |
| --- |
| note  description: "Galaxy represents a game board in simodyssey."  author: "Kevin B"  date: "$Date$"  revision: "$Revision$"    **class** interface  GALAXY    **create**  make,  make\_dummy    **feature** *-- attributes*    gen: RANDOM\_GENERATOR\_ACCESS    grid: ARRAY2 [SECTOR]  *-- the board*    shared\_info: SHARED\_INFORMATION    shared\_info\_access: SHARED\_INFORMATION\_ACCESS    **feature** *-- query*    has\_free\_sector: BOOLEAN  *-- determine if the grid contains at least one sector which has at least one free location*  **ensure**  grid\_not\_full: Result **implies** shared\_info.entities.count < 100    out: STRING\_8  *--Returns grid in string form*    **feature** *--commands*    create\_stationary\_item (sector: SECTOR; id: INTEGER\_32): ENTITY  *-- this feature randomly creates one of the possible types of stationary actors*    put\_item (e: ENTITY; row: INTEGER\_32; col: INTEGER\_32)  *-- place given entity in sector located at [row, col] in the grid*  **ensure**  added\_to\_grid: shared\_info.entities.has (e)  added\_to\_sector: e.sector.contents.has (e)    remove\_item (e: ENTITY; row: INTEGER\_32; col: INTEGER\_32)  *-- remove given entity in sector located at [row, col] in the grid*  **ensure**  removed\_from\_grid: (**old** shared\_info.entities.deep\_twin).has (e) **implies** **not**  shared\_info.entities.has (e)  removed\_from\_sector: (**old** e.sector.contents.deep\_twin).has (e) **implies** **not**  e.sector.contents.has (e)    set\_stationary\_items  *-- distribute stationary items amongst the sectors in the grid.*  *-- There can be only one stationary item in a sector*    **feature** *--constructor*    make  *-- creates a dummy of galaxy grid*    make\_dummy    **end** *-- class GALAXY* |

Figure 2: Shows the contract view of the GAME class

1. The Sector class ( SECTOR.e) looks like this :

|  |
| --- |
| note  description: "Represents a sector in the galaxy."  author: ""  date: "$Date$"  revision: "$Revision$"    **class** interface  SECTOR    **create**  make,  make\_dummy    **feature** *-- Queries*    has\_stationary: BOOLEAN  *-- returns whether the location contains any stationary item*    is\_full: BOOLEAN  *-- Is the location currently full?*    out: STRING\_8  *-- returns string representation of this sector*    print\_sector: STRING\_8  *-- Printable version of location's coordinates with different formatting*    sorted\_contents: ARRAY [ENTITY]  *-- returns sorted array of entities existing in current sector*  *-- sorted by ID in ascending order*    **feature** *-- attributes*    chars\_out: STRING\_8  *-- string representation of entities existing in current sector*    column: INTEGER\_32    contents: ARRAYED\_LIST [ENTITY]  *--holds 4 quadrants*    gen: RANDOM\_GENERATOR\_ACCESS    row: INTEGER\_32    shared\_info: SHARED\_INFORMATION    shared\_info\_access: SHARED\_INFORMATION\_ACCESS    **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** *-- constructor*    make (row\_input: INTEGER\_32; column\_input: INTEGER\_32; a\_explorer: ENTITY)  *--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)    **end** *-- class SECTOR* |

Figure 3: Shows the contract view of the GAME class