Final Project - Into the Darkness Reflection

Ziwei Wu

November 27, 2017

Design

This is the final project of the course. The goal is to create a text-based game with a certain theme. This project provided the most freedom out of all projects in the course as there are no restrictions on what themes or story that you can use in designing the game.

The first thing I want to establish is what is the theme/story of my game. One of the requirements is to use a pure abstract class "Space", which represents the space in the game. The player should be able to move between spaces, interact with the spaces and eventually complete the goal for the game. A lot of themes can fit this model such as crime based, escape room based type of game. I decided to come up with something a little more unique - a spaced-based mission game.

A lot of influence of the story comes from hard sci-fi books such as "The Three Body Problems" by Liu Cixin. The brief summary of the story is the following. In 2030, the earth has been dominant by artificial intelligence, the human race is approaching distinction due to the control of the AI. A team of scientists embarks on a mission to save humanity by finding another planet as the new home, their motivation comes from a detection of a black hole near Saturn. The black hole apparently has a high chance of also being a wormhole that can passage to a distant region of the Universe. They hope that by traveling through it, they can reach the new home and save humanity.

In term of the game mechanic, the player plays the captain of the ship also known as "Frontier". The ship will navigate to Mars as its first stop, then onto Jupiter, Saturn and two of Saturn's satellite called Enceladus and Titan. Along the way, the player will explore each planet and collect a particular resource called antiparticles. When collected a sufficient number of particles, the player can create a material called anti-matter. Anti-matter ensures that safe passage of the ship through the black hole. Once the player successfully passed black hole, the mission is completed.

The game also needs a time constraint. Two constraints in this game are 1. the fuel of the ship 2. the crews of the ship. Each time the ship travels to and explores a planet, some fuel is consumed. If the fuel is ran out at any point of the game before reaching black hole, the game is over. However player can find fuel packs on most of the planets to refill the fuel. During exploration of each planet, there might be event that crews are lost. When all crews are lost, the game also ends. So the player must pay close attention to the status of the ship to ensure these two resources are not depleted.

ship

The "ship" class is the class that represents the frontier ship, it contains the following variables.

- name, string: name of the ship
- description, string: description of the ship
- crews, int: number of crews onboard
- fuel, int: current fuel level
- fuel_tank, int: capacity of fuel tank
- fuelPacks, int: current number of fuel packs onboard
- fuelPacks_capacity, int: maximum number of fuel packs the ship can hold
- antiParticles, int: current number of anti-particles onboard
- fuelPacks_capacity, int: maximum number of anti-particles the ship can hold
- antiMatter_created, bool: a flag to indicate if the anti-matter has been created
- antiParticles_storage, vector: a vector containers to store the particles
- fuelPacks_storage, vector: a vector container to store the fuel packs.

The class includes the set and get methods for each member variables. It also includes member functions to add anti-particles and fuel packs onboard, to use the packs or to create dark matter. Its key function is "display_status" which outputs ship's current information to the player, so the player can monitor the status of the ship closely.

Space

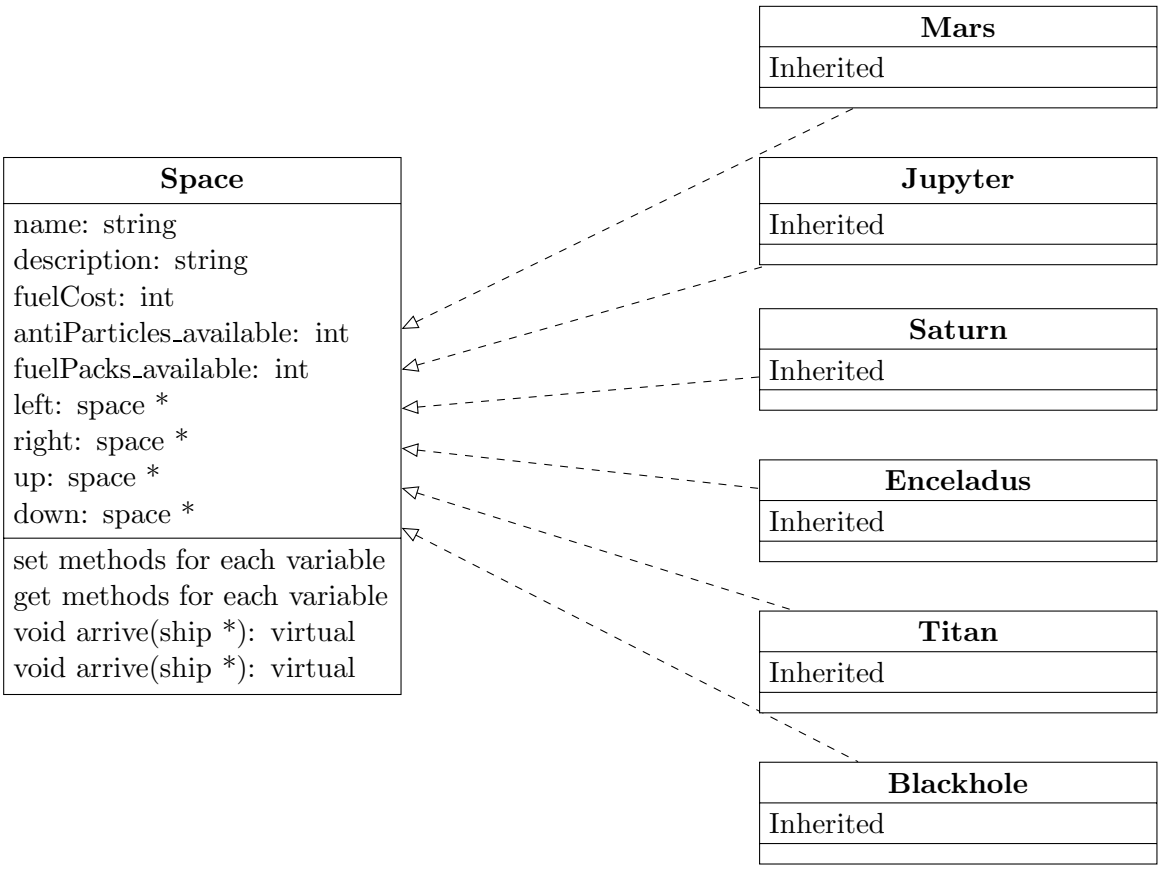
The core class of the game is the Space class, which will be the parent class for each planet. Space would contain the following member variables:

- name, string: the name of the planet.
- description, string: a description of the planet
- fuel cost, int: how many fuels it cost to travel to the planet or to explore the planet.
- antiParticles_available: The number of anti-particles available on the planet
- fuelPacks_available: The number of fuel packs available on the planet.
- left, pointer to Space class: points to the left of the planet
- right, pointer to Space class: points to the right of the planet
- up, pointer to Space class: points to the up of the planet
- down, pointer to Space class: points to the down of the planet

With the idea of encapsulation, the Space class contains set and get methods for all its class variables. It also contains two pure virtual functions.

- void arrive(ship *): This function takes a pointer to a ship object, it defines the arrival event of the ship to the planet. For example, fuel needs to be consumed for the ship to travel to the planet, then followed by a description of the planet.
- void explores(ship *): This function takes a pointer to a ship object, it defines the explore event in which the ship sends a probe to explore the planet. For example, the crew of probe can meet a character, then the probe can gather some anti-particles and fuel packs for the ship. But there might be a chance for the loss of probe, and some crews would get lost.

The derived class of Space including Mars, Jupyter, Saturn, Enceladus, Titan, and Blackhole. With each derived class must redefine the pure virtual function. Here is the class hierarchy diagram.



Each planet classes inherited from Space class. Each of them has a redefined arrive and explore functions. For example, when the ship explores Saturn. Due to its large planetary ring, the ship may discover additional resources inside the ring or encounters a fleet of comets and sustain damage to the fuel tank. In terms of member variables, each planet has different level of resource available depends on their size. For example, Jupiter has more resource than Mars. Finally, is a unique Space derive class. Its explore function checks if the player has met the mission completion requirement.

Linked Structure

The planets in the game are connected by pointers. Each planet has four pointers variables that points to left, right, up and down. The links between planets are in form of a double link in the left and right direction. For example, Jupiter's right points Saturn, and Saturn's left points Jupiter.

Satellites are connected to planets in a up and down fashion. For example, Enceladus's up points to Saturn and Saturn's down points to Enceladus. If a direction pointer is not pointing to anything, it's set to null. This type of linkage allows player to navigate freely between each planet or satellite given they have enough fuels and crews at the early stage of game.

Menu

The menu class initialize the core gameplay and display the menus. It includes the "start_menu", "planet_menu", and "draw_map". This part of the program is actually more time consuming than I thought. start_menu function outputs display the start menu of the game and display the background story of the game. draw_map function display a map of destinations and ship's current location. planet_menu function displays the options for exploring the current planet or move to the other planets. This function is more tricky than I thought because it must handle the situations when a direction points to no planet. My solution is to let the option display "no destination available", and if the user still chooses that option let the user know that the input is invalid. So the menu options are dynamic depends on player's current location.

Testing

The testing of the program is broken into two phases: 1. input validation 2. gameplay. Input validation is the first layer of defense to protect the program from taking any undesired inputs. Various inputs were tested to ensure that input validation is robust. The gameplay was tested for different combinations of parameters to check for proper gameplay, and whether there are any errors or memory leaks.

Table 1: Input Validation Testing

Testing Cases	Input	Expected Output	Observed Output
Start Menu (1 to 2)	0	Invalid Input	Invalid Input
Start Menu (1 to 2)	c	Invalid Input	Invalid Input
Start Menu (1 to 2)	3	Invalid Input	Invalid Input
Start Menu (1 to 2)	1000000	Invalid Input	Invalid Input
Start Menu (1 to 2)	1	valid Input	valid Input
planet Menu (1 to 8)	9	Invalid Input	Invalid Input
planet Menu (1 to 8)	0	Invalid Input	Invalid Input
planet Menu (1 to 8)	d	Invalid Input	Invalid Input
planet Menu (1 to 8)	-1	Invalid Input	Invalid Input
planet Menu (1 to 8)	1000000	Invalid Input	Invalid Input
planet Menu (1 to 8)	7	valid Input	valid Input
Add fuel (0 to current fuelPacks)	-1	invalid Input	Invalid Input
Add fuel (0 to current fuelPacks)	0	valid Input	valid Input
Add fuel (0 to current fuelPacks)	current fuelPacks	valid Input	valid Input
Add fuel (0 to current fuelPacks)	current fuelPacks+1	invalid Input	Invalid Input
Explore events (1 to 2)	0	Invalid Input	Invalid Input
Explore events (1 to 2)	a	Invalid Input	Invalid Input
Explore events (1 to 2)	3	Invalid Input	Invalid Input
Explore events (1 to 2)	1000000	Invalid Input	Invalid Input
Explore events (1 to 2)	-1	Invalid Input	Invalid Input
Explore events (1 to 2)	1	valid Input	valid Input

The input validation testing results showed that the input validation is robust, and invalid data types and values are prevented from entering the program. This ensures that only valid inputs are fed into the program.

Table 2: Gameplay Testing

Testing Parameters Outcomes	Game Play	Error or Infinitely Loop	Graveyard display	Memory Leak
Good ending	Correct	No Errors/Infinite loop	Correct	No Leak
Alternative ending	Correct	No Errors/Infinite loop	Correct	No Leak
Mission Failed(Crew <= 0)	Correct	No Errors/Infinite loop	Correct	No Leak
Mission Failed(Fuel <= 0)	Correct	No Errors/Infinite loop	Correct	No Leak
Mission Failed(Abort mission)	Correct	No Errors/Infinite loop	Correct	No Leak

Good ending - created anti-matter and passed through the black hole
Alternative ending - accept the offer by the alien on Titan
Various combinations of gameplay path were used to test the game. All tests were passed such that the game mechanics are correct, and there are no errors or memory leaks

Reflection

What desgin changes were made?

For this project, I felt the technical implementations are fairly straightforward. A lot of changes were made from the game design standpoint.

First design change I made is to let the user have more choices on what actions to take. Initially, when the player explores the planet, the events play out spontaneously. For example, when the roll determines that the crews would meet the prophet, the event occurs without additional user input. After some playtesting, I decided to give player more agency. So I let the player choose whether they want to investigate the area where they have a chance to meet her. I made similar changes in the other explore events as well. These changes made player felt like they had more freedom and choices during the gameplay.

Second design change I made is to make the game to have two separate endings. Original, the game only had one ending where the player has to travel through the black hole to complete the mission. I thought why not let the player have another option to complete the game. I added an additional event on Titan just before the black hole. When player explores Titan, An alien group appeared and asks the player if they would like to take an offer. By taking the offer the human would be saved, and the price to pay is that human can submit to the alien's rule. I feel this dilemma adds an additional layer to the game, and the player has a difficult decision to make.

Third design change I made is to remove the pathway back to Saturn after the player has reached Titan. The rationale is that the gravitational pull of the nearby black hole is too strong, so the ship is unable to escape once reaching Titan. In terms of implementation, this is achieved by setting the down pointer of Titan and up pointer of Saturn are set to null after reaching Titan. This changed made the gameplay more align with the story.

How problems were solved?

The biggest challenge for this project is the multiple roles that I had to take on to push the game to completion. In additional to being the game programmer, I also have to take the role of a game designer and a story writer. As a game designer, I had to envision what the game would be like, while also considers the time constraints and the level of complexity. For example, I originally envisioned to have more items in the game. But I decided that the added complexity doesn't help the storytelling or gameplay, so I decided to have only two items in the game.

As a story writer, I need to think about how to come up with an interesting story. For example, I could have made the game without a storyline or a limited storyline. I decided to think a little harder to come up with a more interesting story because I think it creates a more memorable experience. In terms of presenting the story, I decided to present the story in more covert ways. For example, rather telling the player very obviously on what they need to do to complete the mission. The game wants the player to explore each planet to collect the clues.

As a programmer, I need to make sure the program is free of error or memory leak while meeting the specifications. Some of the problems I had to solve including implementing draw_map function and planet exploration menu. For both implementations, a good understanding of linked structure proved to be crucial. Both problems were solved by using check conditions for if a pointer to

pointing to null and display different output according to it.

What I've learned?

This project is a great learning experience. Not only I had some programming practices, I also had to develop the game design and come up with the story. A key take-home message for me is that while it's great to have an ambitious goal, as an engineer one must also consider the constraints of time and resources. For a project like a game, the game design is definitely the most important step. After laying out the design, getting a working prototype as faster as possible is important. After that, the game can improve incrementally through each iteration. I realized how much work is put into a game. I gained a lot of respect for the indie game developers because they often have to take on multiple roles including programmer, designer, and artist.

Overall, this project concludes my term here taking cs162. I can see that the majority learning is done through doing the projects. I am definitely going to start my own side projects to keep learning.