Budapest University of Technology and Economics

Asteroid Mining

Subject: Software Project Laboratory
Course code: BMEVIIIAB06
Team name: Pied Pipers

Supervisor:
Dr. Balla Katalin

Members:

Abdelrahman Desoki	DOHDZF	adesoki@edu.bme.hu
Neda Radonjic	NQC17Y	neda.radonjic@edu.bme.hu
Tushig Bat-Erdene	QBI3JH	bat-erdene.tushig@edu.bme.hu
Chaitanya Arora	BWTFX8	chaitanya.arora@edu.bme.hu
Janibyek Bolatkhan	BOI6FK	janibyekbolatkhan@edu.bme.hu
Kasay Ito	LX5RFB	kasay.ito@edu.bme.hu

February 27, 2022

2. Requirements, project, functionality

2.1 Introduction

2.1.1 Goal

The goal of the document is to define a clear structure of the requirements, functionality and project plan. This will help us approach the tasks in a structured and planned manner, as we progress through the development.

2.1.2 Application domain

The aim of the game is to mine resources from asteroids, in order to build a space station. The settlers, controlled by players of the game, roam around in individual spaceships, looking for resources. In order to extract resources, settlers have to perform various operations such as travelling and drilling. The game also involves risks like solar flare which can affect the settlers' health, leading to losing the game.The game is won if the settler extracts at least 3 necessary units of each resource used to build the space station.

2.1.3 References

For the execution of this documentation we made used:

1. StarUML (Use case)
2. Google Docs (Drafting and Compilation)

Problem Definition: Here

2.1.4 Document content

Overview: describes the problem statements and the use cases of the game players
Requirements: describes functional requirements, resource requirements, and non-functional requirements of the project in detail.
Essential use cases:describes the form of dialog between the actor and system, what events the actor does and system does in response.
Glossary: describes the terminology in this documentation and the actual game
Project plan: describes the schedule of deliveries, as well as the organisational structure of the team
Protocol: takes note of the conducted meetings, and their purpose and outcome

2.2 Overview

2.2.1 General overview

The game is based on settlers, robots, and asteroids. Players control settlers, which can build robots, collect resources, and are aiming to build a space station. Resource extraction depends on drilling, which both robots and settlers can do, as well as building teleportation gates for easier travel between asteroids. Extreme weather conditions can kill settlers and damage robots. If they avoid these dangers successfully, and collect enough resources on an asteroid, the player wins the game.

2.2.2 Functions

Asteroid Mining

Humanity has decided to exploit the resources contained in asteroids. In order to do this, the settlers have to build a space station in the asteroid belt. The resources to build a space station would be too expensive to transport there, so the settlers have to mine them from the asteroids.

Players of the game control the settlers. Settlers wander in the asteroid belt with single person spaceships looking for resources.

Asteroids are covered in rocks. The depth of this rock mantle can vary from asteroid to asteroid. The important resources (water ice, iron, carbon, uranium, etc.) can be found in the core of the asteroids. Some resources (like uranium) are highly radioactive. There can be hollow asteroids, whose core is empty. The core of an asteroid is always homogeneous; it is always made of a single kind of a material.

A settler can do one operation in a single move. There are several operations, like travelling, drilling, mining, building robots, building teleportation-gates, etc. When a settler travels, they go to a neighbouring asteroid (an asteroid may have multiple hundreds of neighbours). When a settler drills, they deepen the hole in the mantle with one unit. When a settler mines, they extract the resource from the core of the asteroid. This is only possible if the mantle has been completely drilled through. A single settler can only carry 10 units of resources, that's the spaceships' capacity. A hollow asteroid, however, can be filled with a unit of resource, counting as an operation.

There are some dangers lurking in the asteroid belt. When a fully drilled asteroid with a radioactive core is at perihelion, the asteroid explodes and kills any settler on it. Radioactive material, therefore, can only be mined when the asteroid is at aphelion. Sometimes a sun storm reaches the asteroid belt, and it is also dangerous to the settlers. A settler can only survive a

sun storm if they hide in the core of a hollow asteroid. This, of course, is only possible if the mantle has been drilled through.

When a fully drilled asteroid with water ice in its core is at perihelion, the water ice sublimates (disappears).

Using up a unit of iron, a unit of carbon and a unit of uranium, the settlers can build autonomous robots controlled by artificial intelligence. These robots can only travel between asteroids and drill holes. Robots cannot mine because they are unable to transport things. Robots, however, can survive radioactive explosions, and in this case they land on a neighbouring asteroid. Sun storms do damage robots unless they hide in a hollow asteroid.

Using up two units of iron, a single unit of water ice and a single unit of uranium, the settlers can build a pair of teleportation-gates. The gates can be later deployed in the vicinity of the asteroid the settler is on. The two gates of a pair remain in contact for good, and entering either the traveller (settler, robot, etc) will be immediately transported to the other. The settlers can bring the freshly built gates with themselves, but at the same time a single settler can only bring two gates.

The game can end in two ways. If all the settlers die, the players lose. If, however, they can mine at least three units of each resource and they collect those materials on a single asteroid, they can build the space station and the players win the game.

2.2.3 Users

Users of the game are mainly gamers and normal people who are interested in asteroid mining and problem solving.

2.3 Requirements

2.3.1 Functional requirements

ID	Description	Check	Priority	Source	Use-case	Comment
R01	Settler must build a space station in the asteroid belt.	Demonstration	Must have	Problem description	Build	

R02	Players control the settlers	Demonstration	Must have	Problem Description	Move-Build-Mine-Drills MAntle	
R03	The depth of the rock mantle can vary from asteroid to asteroid	Testing	Must Have	Problem Description	Drill mantle	
R04	A settler can do one operation in a single move	Demonstration	Must have	Problem Description	Move-Build-Mine-Drills Mantle	
R05	When a settler travels, they go to a neighbouring asteroid	Demonstration	Should have	Problem Description	Move	
R06	When a settler drills, they deepen the hole	Demonstration	Should have	Problem Description	Drills Mantle	
R07	When a settler mines, they extract the resource from the core of the asteroid.	Demonstration	Should have	Problem Description	Mine	

R08	A settler can build a robot.	Demonstration	Should have	Problem Description	Build	
R09	A settler can build a teleportation gate.	Demonstration	Should have	Problem Description	Build	
R10	If a radioactive core is at perihelion, the asteroid explodes and kills any settler on it.	Testing	Must have	Problem Description	Determine Danger	
R11	Radioactive material,, can only be mined when the asteroid is at aphelion	Testing	Must have	Problem Description	Determine Danger	
R12	If sun storm reaches the asteroid belt, it might is dangerous to the settlers	Demonstration	Should have	Problem Description	Determine Danger	

R13	A settler can only survive a sun storm if they hide in the core of a hollow asteroid.	Demonstration	Must have	Problem Description	Settler	
R14	When a fully drilled asteroid with water ice in its core is at perihelion, the water ice sublimates (disappears).	Demonstration	Must have	Problem Description	Drills Mantle	
R15	These robots can only travel between asteroids and drill holes	Testing	Could have	Problem Description	Drills Mantle-Robot	
R16	Robots cannot mine because they are unable to transport things	Demonstration	Must have	Problem Description	Robot	
R17	Robots can survive radioactive explosions.	Demonstration	Could have	Problem description	Robot	

R18	The settlers can build a pair of teleportation-gates.	Demonstration	Could have	Problem description	Build	
R19	Gates can be deployed in the vicinity of an asteroid.	Demonstration	Should have	Problem description	Deploy	
R20	A single settler can only bring two gates.	Demonstration	Should have	Problem description	Settler	
R21	If all the settlers die, the players lose.	Testing	Must have	Problem description	Operating Game	
R22	If they collect those materials on a single asteroid, they can build the space station (Win game)	Demonstration	Must have	Problem description	Operating Game	

2.3.2 Resource requirements

ID	Description	Check	Priority	Source	Comment

R01	Installed version of Eclipse (4.10 or higher)	Eclipse IDE must be installed in the machine in order to compile the source code.	Should have	Google:	
R02	Java Development Kit	JRE/JDK 8+ must be installed on the device..	Must have	Problem statements	
R03	Operating System: Windows 7, 8, 10, 64-bit versions only; Mac OS X 10.12.4+; Linux;	In order to compile and run the program. User must have one of the operating system for action to be done.	Must have one of them	What are the minimum requirements of a PC to make a 2D Android game using Unity? (2019). Quora. https://www.quora.com/What-are-the-minimum-requirements-of-a-PC-to-make-a-2D-Android-game-using-Unity	
R04	Standard Mouse, Keyboard	System testing	Must have	The explanation during the second consultation	

R05	Graphical calculation power/ Graphics Card/GPU	System testing	Should have	M. (2019). PC requirements to run a simple 2D game built with UE 4.23. Unreal Engine Forums. https://forums.unrealengine.com/t/pc-requirements-to-run-a-simple-2d-game-built-with-ue-4-23/133142	
R06	Memory requirement	2GB for convenient game play	Should have	Progress KB - How to increase the amount. (2020). Progress Software Knowledgebase. https://knowledgebase.progress.com/articles/Article/P122464 Eclipse bug - Maximum heap is limited to 256MB if started without Xmx-Parameter and Java >= 11.0.(2019). https://bugs.eclipse.org/bugs/show_bug.cgi?id=553538	

R07	CPU processor requirements	Any processor that can run Eclipse IDE	Should have	https://wiki.eclipse.org/Eclipse/Installation#Install_a_JVM	

2.3.3 Non- functional requirements, Restrictions

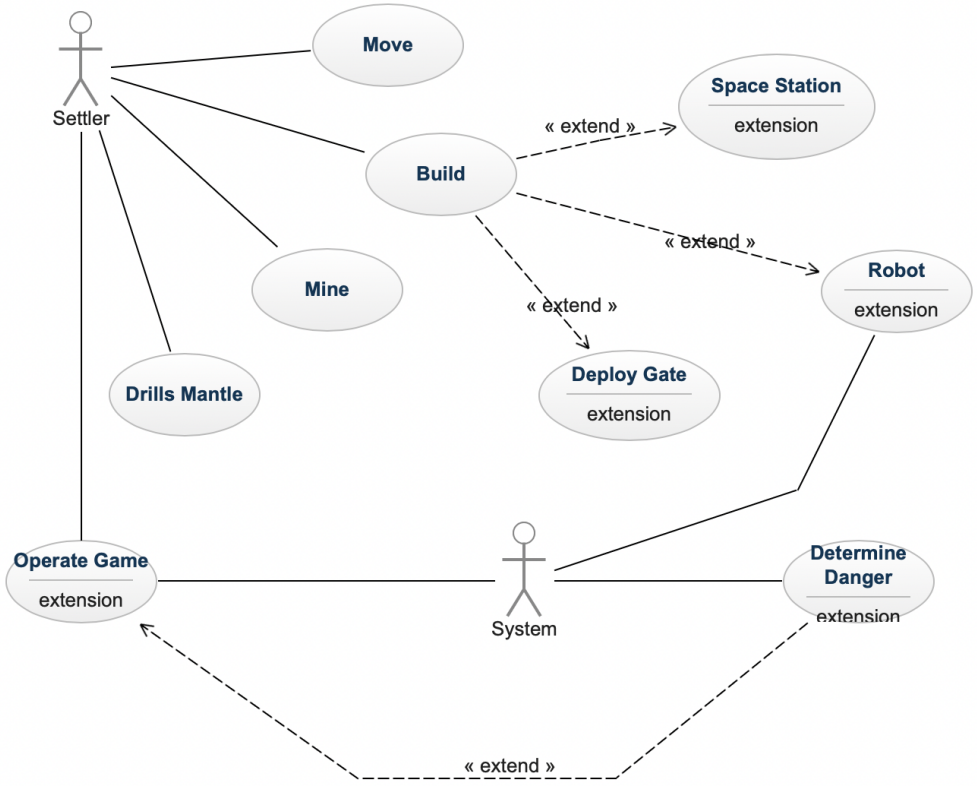
ID	Description	Check	Priority	Source	Comment
R01	Enough storage to download and run the software	Users must check the free storage before downloading/installing the game software.	Must have	Brainstorming	Testability

R02	Understanding Rules of the game	Make sure that the user has enough knowledge to know the rules of the game and play it. (+6 years old)	Must have	Brainstorming	Learnability
R03	Fault Tolerance of game	To make sure all edge cases involving different permutation and combination are included	Must have	Lecture note of Software Engineering subject	Availability
R04	Reliability of the game	Users shall play the game with high performance without errors or fall down.	Must have	Lecture note of Software Engineering subject	Reliability
R05	Coexistence Interoperability	Other developers can continue the developmental process if the environment	Should have	Lecture note of Software Engineering subject	Compatibility

		requirements are fulfilled.			

2.4 Essential use-cases

2.4.1 Use-case diagram



2.4.2 Use-case descriptions

Use-case Name	Operate Game

Short textual Description	The Operating Game involves certain operations like Starting game, ending game etc.The System controls this Use case and then the control goes to the settler
Actors	System, Settler
Main Case Scenario	The Project is run and the System calls the Operating game and the game is started
Alternate Scenario	The game is running and the user conditions to fail become correct and the operate game is called which ends the game
Use-case Name	Moves
Short textual Description	Move function is used to change the position of the actors which can be settler or robot
Actors	Settler
Main Case Scenario	The player moves with the spaceship in directions like left, right, up or down
Use-case Name	Mine
Short textual Description	Mining is a by the settler to mine the metals that they get in the asteroid

Actors	Settler
Main Case Scenario	The settler mines the asteroid metals and extracts the necessary materials
Use-case Name	Build
Short textual Description	Build use case give settlers ability to build Robots, Gate, and
Actors	Settlers
Main Case Scenario	Players can build Robots, Gate, and Space Station by using necessary units of resources.
Alternate Scenario	
Use-case Name	Drills Mantle
Short textual Description	Drills mantle is one of the function executed by the settler to drill the asteroid in the core in order to reach the metals
Actors	Settlers
Main Case Scenario	Settler drills the asteroids

Use-case Name	Space Station
Short textual Description	Building the Space Station will be the aim of the settler. Since it's one of the things that the settler builds, extend is used
Actors	Settler
Main Case Scenario	The settler extracts enough resources and builds the Space Station
Use-case Name	Deploy Gate
Short textual Description	Settler deploys teleportation-gates in the vicinity of the asteroid where settlers ison.
Actors	Settlers
Main Case Scenario	Teleportation-gates give settlers and robots the ability to transport between asteroids.
Alternate Scenario	Settlers can only bring two-gates.
Use-case Name	Robot

Short textual Description	The settler builds the Robot which itself will perform different functions like drilling and will be controlled by the system and not the settler
Actors	System
Main Case Scenario	The Settler is controlled by System to perform different tasks
Alternate Scenario	The Robot is built by the settler
Use-case Name	Determine Danger
Short textual Description	The major task of determining danger is to make sure of the health rate of the Settler and robot from various mishappenings like Blast, solar storm etc.
Actors	System
Main Case Scenario	The system checks the health of the robot and settler

2.5 Glossary

Term	Description	Type
Asteroid	Asteroids are objects that contain resources inside them.	Object

Space station	The objective of this game, namely, the construction of a space station is the goal of the user (the player).	Object
Settler	The main player of the game and miner with single spaceship	Object
Robot	Autonomous robots can be built by settlers and controlled artificially. They can travel between asteroids and drill holes. They are unable to mine and transport things. They must land on a neighbouring asteroid to survive radioactive explosions. But, sun storms can damage them unless they hide in a hollow asteroid.	Object
Mantle	Mantle covers the corresponding asteroids	Object
Drill	Operation to deepen the mantle of the asteroid.	Action
Mine	Operation to collect the resources when drilling is done.	Action
Build	Operation of settlers to launch space stations, robots, and the teleportation gates.	Action
Perihelion	The event that handles some of the dangers in the game. If perihelion is occured when there are revealed radioactive resources (e.g. uranium).	State
Teleportation-gates	Gate built by robots to transport between asteroids.	Object
Sun Storm	Dangerous scenes can cause damage.	Event
Resources	Materials (Uranium, Water ice, and carbon) are essential to build a station.	Object

Radioactive Explosion	An event that can harm the settlers and robots.	Event

2.6 Project plan

The purpose of this project is to put our coursework knowledge into hands-on practice and improve our team-work skills and experience with working software engineering methodology. In terms of the objective of this project, we carefully follow the problem description and its requirements to build a software which provides the customer's exact need.

We will continuously be working on the project, with expected deliveries almost every week, according to a schedule that can be seen below. Apart from the documentation which will continuously be created and delivered, the code will be delivered in 3 main stages- Skeleton, Prototype, and Complete program.

Mar 07.	Analysis Model - first version -- documentation
Mar 16.	Analysis Model - final version -- documentation
Mar 21.	Planning the Skeleton -- documentation
Mar 28.	Skeleton program -- documentation and source code.
Apr 04.	Concept of Prototype -- documentation
Apr 11.	Detailed plans -- documentation
Apr 25.	Prototype program -- documentation and source code
May 02.	User interface specification -- documentation
May 09.	Creating the Complete part
May 16.	Complete program -- documentation and source code.
May 20.	Summary

2.6.1 Organisational Structure

Name	Role	Demonstration
Abdelrahman Desoki	Testing, Back-end Developer	Responsible for the quality of the code and testing its main functionality. Checking the logic behind the code and developing the database and operating system side Contributing in the integration of the backend and frontend.
Neda Radonjic	Project Manager, Testing Engineer	Planning and overseeing the project to ensure the schedule is followed and deadlines are met. Designing and running tests and analysing their results to ensure the quality and functionality of the code.
Tushig Bat-Erdene	Front-End Developer, Software Architect	Responsible for developing and designing the front-end of the software in terms of graphical user interface development. Providing architectural blueprint of the development cycle for the team.
Chaitanya Arora	Project Lead, Backend Developer	Create and communicate a clear list of expectations and goals for team members to follow. Handling the backend implementation of the project and dealing with roadblocks in the implementation.
Kasay Ito	Software Architect, Supporter	Mainly designs how the system should be implemented and maintains the design with appropriate level. Due to the experience of development of the game previously, he can support other team members when

		they encounter issues including technical problems, understanding the requirements, managing the project, and the like.
Janibyek Bolatkhan	Quality Assurance, Front-End Developer	Responsible for developing and designing the front-end of the software in terms of GUI development. Testing and Improving the quality of the code by integrating backend and frontend interfaces.

2.7 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
15.02.2022	*40 min*	All team members	Our first meeting had the purpose of getting to know the people that we will be working with, and making initial, informal plans for the project and getting to know each other's strengths and weaknesses in different parts like frontend, backend etc. No specific decisions were made, since the topic was not announced at the time.

20.02.2022	1 hour	All team members	The second meeting involved us to understand and comprehend the game and its main functionalities to make sure we all were on the same page. It involved us to structure the problem and decide the problem statement and decide our future steps.
2/21 Monday use case diagram, requirements refinement	3 hours	All team members	This third meeting was an in person meeting conducted to brainstorm and work together. During this time we worked together and discussed the major parts of the submission like the requirements and use case diagram etc. After having to brainstorm the ideas and everyone's views we collaborated together to draft the Introduction, overview, Functional Requirements and use case descriptions
2/22 Tuesday consultation	1.5 hours	All team Members	Discussed the previously created use-case diagram with the supervisor and got clarification of some doubts.
16:00-17:30 Thursday	1.5 hours	All team Members	Wrapping up the work on our first piece of documentation. The chapters that we covered were resource requirements, non-functional requirements, project plan, protocol, and glossary.

10:00-11:30 Sunday	1.5 hours	All team members	Deciding on some last-minute changes regarding the organisational structure of the team and modifying the documentation accordingly.

4. Analysis model – version 2

4.1 Object catalog

4.1.1. Visitor

A visitor is either a robot or a settler, and depending on this, it has different responsibilities and capabilities. What is common to all visitors is that they travel between different asteroids in the asteroid belt, looking for resources. Making use of the OOPs concepts. We used inheritance.Settlers, they are capable of more-they can mine, unlike robots, and they can build robots and teleportation gates. Robots, once they are built by settlers, can perform the action of drilling the asteroids. Settlers are more affected by dangers - sun storms, and radioactive explosions, whereas robots survive explosions. Their responsibilities are highlighted more accurately in individual object descriptions and common ones are represented in the visitor class.

4.1.2. Settler

The settler is the main protagonist of the game with the goal of collecting enough resources from asteroids to build a space station. It can perform various tasks by itself like traveling in the

spaceship through the asteroid belt, wandering in the asteroid belt to find resources, drilling, mining, building robots, building teleportation-gates and using them for transport, and hiding in the asteroid during danger situations. Sun Storm and radioactive explosions can affect the settlers. Responsibilities of the settler involve finding adequate resources to build the space station and collecting them on one asteroid in order to win the game, keep himself safe from the various dangers that he/she might encounter.

4.1.3. Robot

If they collect one unit of each iron, carbon, and uranium, the settlers can build robots which are autonomous entities controlled by artificial intelligence. The robots can only travel/be teleported between asteroids and drill. Since they are not able to transport things, they can not mine. Robots are also affected by dangers, but they can survive a radioactive explosion, only landing on a neighboring asteroid, and they can avoid damage from the sunstorm if they hide in a hollow asteroid. They are responsible for drilling, and also hiding when sandstorms occur, in order to stay safe.

4.1.4. Spaceship

A one person spaceship is a vehicle used by each settler to travel between different asteroids in the asteroid belt. It is the main means of transport in the game. A spaceship has a capacity of 10 units of a resource. Its responsibility is getting the visitors from one asteroid to another, and transporting their resources as well. This entity differs from the settler because a spaceship exists graphically and the settler can get out of it and directly stand on the asteroid. This is needed when considering extensibility or requirement changes such as the spaceship has been affected by the sun storm or the settler can use the spaceship for hiding from the sun storm.

4.1.5. Resource

Resources are a variety of minerals which can be found in the core of the asteroids. Settler must mine and collect the necessary resources to achieve his goal of building the space station. The resources can also be used to build robots and teleportation gates. Resources can be collected and transported by the settler, who can carry at most 10 units of a resource at a time. The important resources that can be found include water, ice, iron, carbon, uranium. However, some resources, such as uranium, are highly radioactive and can cause explosions of asteroids that they make up, which is a danger to settlers.

4.1.6. Uranium

Uranium is one of the resources found in the core of asteroids. It differs from other resources because it is radioactive. Therefore, those asteroids that have a core made up from uranium can explode. Uranium is one of the resources responsible for making up the structure of robots and teleportation gates.

4.1.7. Water Ice

Water ice is one of the resources found in the core of asteroids. It differs from other resources, because when a fully drilled asteroid with a core made up from water ice is at perihelion, the water ice sublimates (disappears). Water ice is one of the resources responsible for making up the structure of teleportation gates.

4.1.8. Iron

Iron is one of the resources found in the core of asteroids. It is one of the resources responsible for making up the structure of robots and teleportation gates.

4.1.9. Carbon

Carbon is one of the resources found in the core of asteroids. It is one of the resources responsible for making up the structure of robots.

4.1.10. Place

Place has responsibility that it deals with common functionalities between Asteroid and Teleportation Gates. Therefore, the information related to neighboring places (asteroids or teleportation gates) and which visitor (settler or robot) is on there is described here, meaning that this entity is the way to know neighbors.

4.1.11. Teleportation Gates

Apart from the spaceships, another transportation option are teleportation gates. Settlers can build them, provided that they have the needed resources which make up the gates - iron, water ice, and uranium. They come in pairs, and are deployed by the settlers, which can only carry two gates at a time. A settler/robot who enters the gate at one end will be teleported to the other end by the gate. The gates are responsible for teleporting their users (visitors) between neighboring asteroids between which they are deployed.

4.1.12. Asteroid

In the asteroid belt, there are many asteroids between which the robots and settlers can travel. They are covered by rocks, and the depth of the rock mantle can vary from asteroid to asteroid.The asteroids either contain a single type of a resource/material (some of which are radioactive) in their core, or they are hollow. Settlers drill and mine the asteroids to extract resources, whereas robots can only drill them. If an asteroid is hollow, it can be filled by one unit of a resource by the settler. If an asteroid is fully drilled and has a radioactive core, it will explode when at perihelion, killing any settler on it. Hollow asteroids can serve as a shelter for settlers in case of a sun storm. Responsibilities of the asteroids are: being a source of useful resources, being shelters from danger, and hosting visitors.

4.1.13. Radioactive Asteroid

When the core of an asteroid consists of uranium, that asteroid is radioactive. Such structure of the core is responsible for explosions of asteroids, which can occur, killing settlers, and launching robots to different asteroids. The reason why this functionality does not depend on Resource is that in Object Oriented (OO) concept, the condition whether radio active or not is determined by state of the asteroid, according to the requirements which describes it as fully drilled and perihelion.

4.1.14. Sun Storm

The sun storm exists visually in the game and the user can notice that there is an sun storm. This entity has responsibility for existing so that the Game can compute the collision among other objects. If there is no such sun storm entity (i.e. class), then the user may not notice the event related to the sun storm and the Game cannot compute the collision between them. Consequently, this object is one of our assumptions.

4.1.15. Game

This entity has responsibility for starting and ending the game itself. Inside these functionalities, the condition checks and determinants of some hazards. The Game has a different concept from the Handler due to the fact that Game can do “operations” such as determining perihelion or not, creating the sun storm, and the like, rather than Handler handles each game object. The possible concerned extensibility or requirement changes are following: changes the condition of winning/losing the game plays, changes from single player game to multiplayers game

4.1.16. Handler

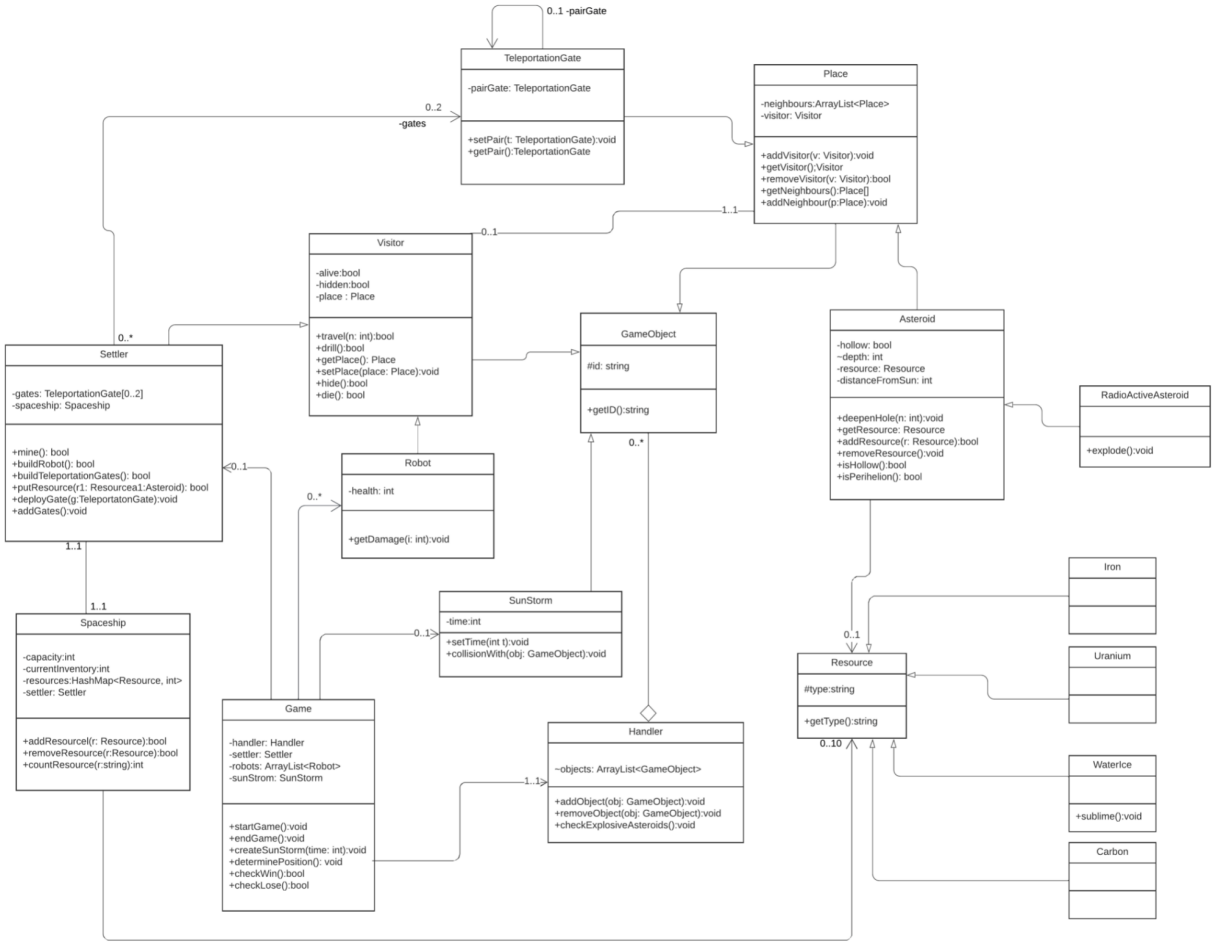
The Handler contains all necessary instances (Settler, Robot, Teleportation Gates, Asteroid) which relate to the played game. Moreover, it checks internal conditions of each game object (such as the game can know whether the asteroid is explosive or not) and add/removes GameObject in the game. This entity is essential because it must be possible to access all objects existing in the game in order to check the condition regarding events, and it makes the user able to notice the existence of the objects. The responsibility of this object is different from the Game, which is described above, because the former focuses on behaviors against the game objects, however, the latter deals with the game itself. The possible concerned extensibility or requirement changes are following: changes the computational/scientific method for collision (whether the collision occurred or not), changes number of objects that can exist in the game (e.g. up to 100 objects can appear in the game at same time)

4.1.17. Game Object

Game Object is an entity that has common functionalities among Visitor (Settler, Robot), Place (TeleportationGate, Asteroid) and Sun Storm. For example, unique id to distinguish, user view position, information needed for collision and extra computation are denoted here. This represents physical or graphical existence, namely, the concept of Game Object is differently categorized from logical existence such as Game, Handler or Resource because they are only

values that the user cannot understand the meaning of them directly. The possible extensibility or requirement changes are following: extra information added to all of Visitor, Place and Sun Storm. Besides that game object can be considered as the higher level wrapper of the whole game which is used to manage the tasks and differentiate the individual entities inside the game.

4.2 Static structure diagrams



4.3 Class description

4.3.1. Visitor

Responsibility

The Visitor class is a parent class of Settler and Robot classes, since they share many attributes and features. It stores information about the asteroid the visitor is currently on (place), whether the visitor is hidden in a hollow asteroid, and if it is alive at all. It contains methods responsible for traveling, drilling, mining, and getting and setting the place of each visitor.

Attributes

- **bool alive:** specifies if the settler is alive or not at the moment, to ensure extendability, in case of requirements change (e.g. multiple settlers), this condition will be later changed according to the events and things that take place.

- **bool hidden**: indicates if the visitor is hidden in the core of a hollow asteroid.It is an important attribute to understand the state of the settler as it can have limitations based on them.
- **Place place:** stores information about the place of the visitor(the asteroid that it is on)

Methods

- **bool travel**- The settler attempts to travel to a neighboring asteroid. If the asteroid really is a neighbor and the operation is successful, the truth is returned.
- **bool drill**- The settler drills through the mantle of an asteroid.
- **Place getPlace -** Gets the current position of a visitor.
- **void setPlace**- Sets the position of a visitor.
- **bool hide** - Checks if a visitor is hidden in a hollow asteroid. If not, the action of hiding a visitor is performed, and a true is returned.
- **bool die -** If the health of a robot reaches zero, it is killed, with no effect to the rest of the game. If a settler is not hidden in an explosion, it is killed, and the game ends.

Superclasses

- **GameObject**

4.3.2 Settler

Responsibility

The settler class inherits many attributes and methods from the visitor class. The Settler class has a few additional attributes and methods, since a Settler has some different capabilities and responsibilities compared to the other type of visitors-robots. This class stores information about the teleportation gates and spaceship belonging to a specific settler. It contains methods exclusive to settlers - such as mining, building robots and teleportation gates, storing resources and deploying gates.

Attributes

- **TeleportationGate gates:** Stores information about any teleportation gates that the settler has built and deployed.
- **Spaceship spaceship:** Stores information about the personal spaceship of this settler that it uses for transportation.

Methods

- **bool mine()** - If the mantle of the asteroid has been drilled through, the settler now mines the asteroid for useful resources.
- **bool buildRobot()** - We check if there are enough resources to build a robot. If so, a robot is instantiated, and true is returned.
- **bool buildTeleportationGates()** - We check if there are enough resources to build a teleportation gate. If so, a gate is instantiated, and a true is returned.
- **void putResource** - The settler can fill a hollow asteroid with one unit of a resource.
- **void deployGate(TeleportationGate g):** deploys the corresponding available teleportation gate.
- **void addGates(Teleportation g):** adds new gates to the list gates which Settler has right now.

Superclasses

- **Visitor->GameObject**

4.3.3 Robot

Responsibility

This class is responsible for managing robots, which are a type of visitors. It stores information about the health of the robot, which is a metric indicating if a robot has been affected by dangers. It also contains a method that marks a robot as damaged, decreasing its health if it is affected by a danger.

Attributes

- **int health:** the robot survives sun storms with damage only, so this is a metric of the damage made to the robot during a storm.

Methods

- **void getDamage(int n):** In the case of a sun storm, a robot that was not hidden inside a hollow asteroid will be marked as damaged.

Superclasses

- **Visitor->GameObject**

4.3.4 Spaceship

Responsibility

The responsibility of this class is to store attributes regarding inventory, capacity, the resources stored in the spaceship, and the settler that the spaceship belongs to. The methods deal with inventory- adding resources, removing them, and counting them.

Attributes

- **int currentInventory:** stores the number of units of resources that the settler has stored in the spaceship already
- **HashMap resources:** describes the resources that the settler currently has. The key of the map is the type of the resource, and the value is the amount of the resource.
- **int capacity:** Indicates the capacity of the spaceship when it comes to storing resources.
- **Settler settler:** Indicates which settler owns the spaceship.

Methods

- **bool addResource (Resource r):** Checks if there is enough space for an additional unit of resource, and if so, adds it to the spaceship, and returns true.
- **bool removeResource(Resource r):** Removes a unit of resource from a spaceship and returns true.
- **int countResource(string r):** Counts the number of units of resource in a spaceship and returns an integer value.

4.3.5. Resource

Responsibility

This class is a parent class to four different resources- uranium, iron, carbon and water ice. It contains the attribute type, that indicates the name/type, and a method to access that name and have it returned as a string.

Attributes

- **string type:** Stores the name of the resource name/type.

Methods

- **string getType:** Gets the name of the individual resource and returns it as a string.

4.3.6. Uranium

Responsibility

The Uranium class is a child of the Resource class. It stores information about one type of the resources- Uranium.

Superclasses

- **Resource**

4.3.7 Water ice

Responsibility

Water ice is a child class of the Resource class. It stores information about one type of the resources- Water Ice. It also contains a method for water sublimation (disappearance).

Methods

- **void sublime()**: When an asteroid with water ice in its core is at perihelion, this water ice sublimates(evaporates).

Superclasses

- **Resource**

4.3.8 Iron

Responsibility

Iron class is a child of the Resource class. It stores information about one type of the resources- Iron.

Superclasses

- **Resource**

4.3.9 Carbon

Responsibility

Carbon class is a child of the Resource class. It stores information about one type of the resources- Carbon.

Superclasses

- **Resource**

4.3.10 Place

Responsibility

This class is a parent class of Asteroid and TeleportationGate. These two entities are places where a Visitor can be, and the Place class deals with them. It stores information about its neighbors, and the visitors of the places. It inherits its id from the GameObject class.

Attributes

- **ArrayList<Place> neighbors:** All neighbors of a certain Place are stored in a list-neighbors are Asteroids or Teleportation Gates on either side of another Place.
- **Visitor visitor:** Stores information about which visitor is currently visiting a place.

Methods

- **void addVisitor (Visitor v):** adds a visitor v as a current visitor to the place.
- **Visitor getVisitor:** gets information about which visitor is currently visiting a place, and returns it.
- **bool removeVisitor(Visitor v):** removes a current visitor when it leaves the place, and returns true.
- **Place[] getNeighbours:** gets neighbors of a certain place-the asteroids or teleportation gates on either side of it.
- **void addNeighbour(Place p):** adds a new neighbor to a place.

Superclasses

- **GameObject**

4.3.11 Teleportation gate

Responsibility

This class stores information about teleportation gates that were deployed by settlers. These come in pairs, so it is important for two gates of a pair to be linked with one another, which is why we need a pairGate attribute, and methods to set and get a gate's pair. This is a child class of the Place class, since it is one of the two available types of places, and through Place, it inherits its ID from Game Object.

Attributes

- **TeleportationGate pairGate:** Stores information about what other gate is connected with this gate, since they come in pairs.

Methods

- **void setPair(TeleportationGate t):** Assigns one gate to another, as they come in pairs.
- **TeleportationGate getPair():** Gets and returns the pair of a gate

Superclasses

- **Place->GameObject**

4.3.12. Asteroid

Responsibility

This class is a child class of Place, and through place it inherits an ID from Game Object. It is the parent class of RadioActiveAsteroid class, since radioactive asteroids are one distinct type of asteroid. This class stores information about whether asteroids are hollow, the depth of their rock mantle, and which resource the core is made from. Also, it contains methods such as deepening the rock mantle, adding and removing resources, determining whether the asteroid is hollow, and whether or not it is at perihelion.

Attributes

- **bool hollow:** stores information about whether an asteroid has a hollow core or not. A core is hollow when there is no resource.
- **int depth:** the depth of the rock mantle varies, and this attribute stories the depth of the mantle.
- **Resource resource:** stores information about the type of the resource which makes up the core of an asteroid.

- **int distanceFromSun:** Indicates the distance between a radioactive asteroid and the Sun. This is important to be able to determine when the asteroid is at aphelion or perihelion.

Methods

- **void deepenHole(int n):** The rock mantle is deepened by one unit. The int parameter would be useful in case of requirement change, so the mantle could be deepened by a different amount.
- **Resource getResource():** Returns the name of the resource that makes up the asteroid core.
- **bool addResource(Resource r):** Adds resource r to the core of an asteroid and returns true if the action is performed successfully.
- **void removeResource():** Removes resource from the core of an asteroid.
- **bool isHollow():** Checks if an asteroid has a hollow core.
- **bool isPerihelion():** used to check if an asteroid is at perihelion position. Important for radioactive asteroids, and those with water ice in their core.

Superclasses

- **Place->GameObject**

4.3.13 Radioactive Asteroid

Responsibility

This is a child class of Asteroids. It is needed because one type of asteroid- those with uranium in their core are radioactive, and can explode when at perihelion. Since it's a special behaviors of a particular asteroid and it as a whole retains properties of a normal asteroid we used inheritance

Methods

- **void explode():** When an asteroid with a radioactive core is at perihelion, it will explode, killing settlers, and displacing robots.

Superclasses

- **Asteroid->Place->GameObject**

4.3.14. Sun Storm

Responsibility

This class stores information about the sun storm- when it occurs. It also contains the methods needed for the collisions with objects of the game. The sun storm inherits its ID from the Game object.

Attributes

- **int time:** stores a time value for setting the time of sun storm occurrence.

Methods

- **void setTime(int t):** Sets the time t when the sun storm will occur.
- **void collisionWith(GameObject obj):** Performs the collision of the sun storm with one of the objects in the game.

Superclasses

- **GameObject**

4.3.15 Game

Responsibility

Only one GameSystem object can exist in a game. This class has the responsibility to start and end the game. These include condition checks, creating major entities and events.

Attributes

- **Handler handler:** This is an object of the class Asteroid belt this will; be used to create an asteroid belt in the game
- **Settler settler:** This is an object of the class settler which will later be used in the diagram.
- **ArrayList<Robot> robots:** Game system must know all robots since they are controlled by the system.
- **SunStorm sunStorm:** Sun storm object which is used when a sun storm occurs.

Methods

- **void StartGame():** This function is used to initialize all major elements in the game.
- **void EndGame()**: this function is majorly invoked when the conditions which the settler can die become true.

- **void createSunStorm(int time):** this function can be invoked randomly and creates an occurrence of a sunstorm in the game
- **void determinePosition():** Determines if an asteroid is at perihelion or aphelion, based on the attribute distance from Sun. This is important for water ice asteroids and radioactive asteroids.
- **bool checkWin():** Checks if the needed resources have been collected on an asteroid, and if so, the game is won.
- **bool checkLose():** Checks if the settler has been killed by an explosion, and if so, the game is lost.

4.3.16 Handler

Responsibility

Handler is a container that contains all necessary instances related to the played game. A settler, asteroids, robots and teleportation gates are contained. Furthermore, this has the responsibility for checking whether the asteroid is explosive or not.

Attributes

- **ArrayList<GameObject> objects:** all the list of the objects that currently exist on the asteroid belt, ranging from visitor to asteroid.

Methods

- **void addObject(GameObject obj):** Adds corresponding object to the **objects** list
- **void removeObject(GameObject obj):** removes corresponding object from the list.
- **void checkExplosiveAsteroids():** calls explode() if the radioactive asteroid is on the perihelion.

4.3.17 Game Object

Responsibilities

GameObject is responsible for dealing with existing objects in the game. During the extension or requirement changes, we will add the functions which are commonly used among derived classes. Many classes inherit their unique ID from the Game Object class, since it is their superclass.

Attributes

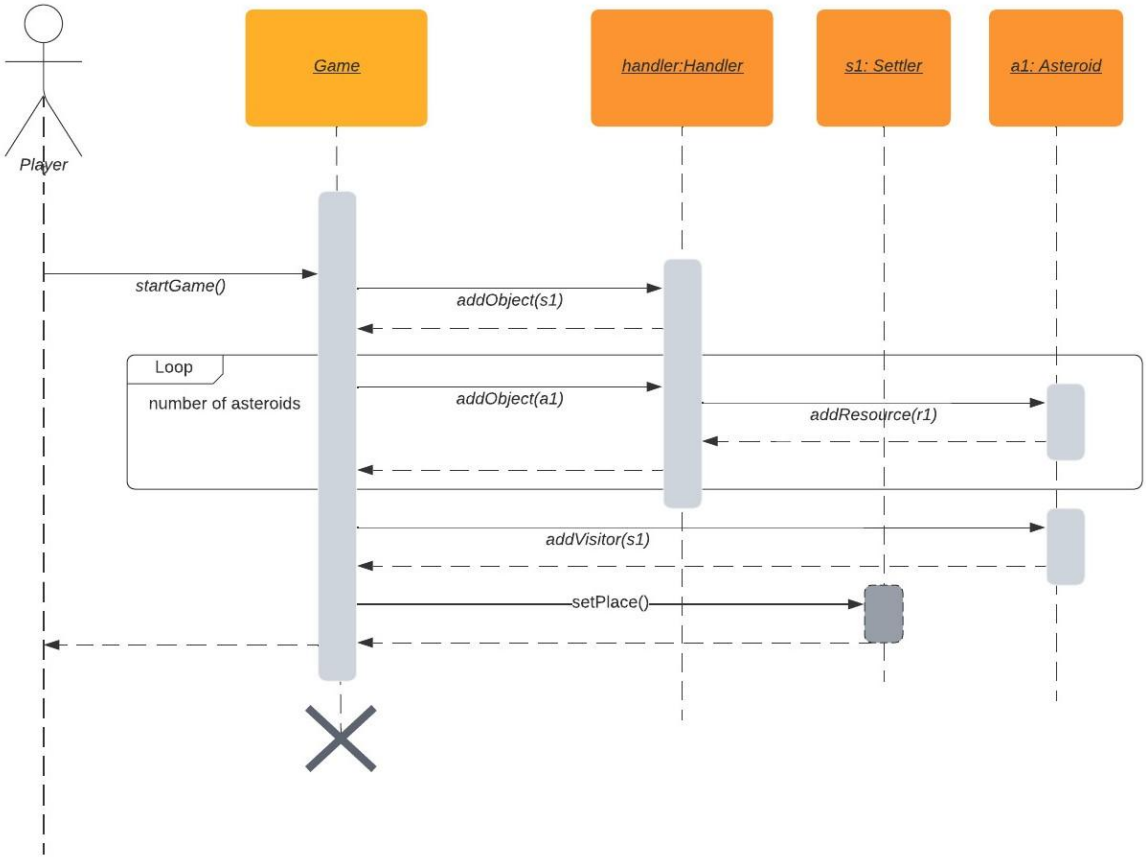
- **string id:** stores the unique ID of objects in the game

Methods

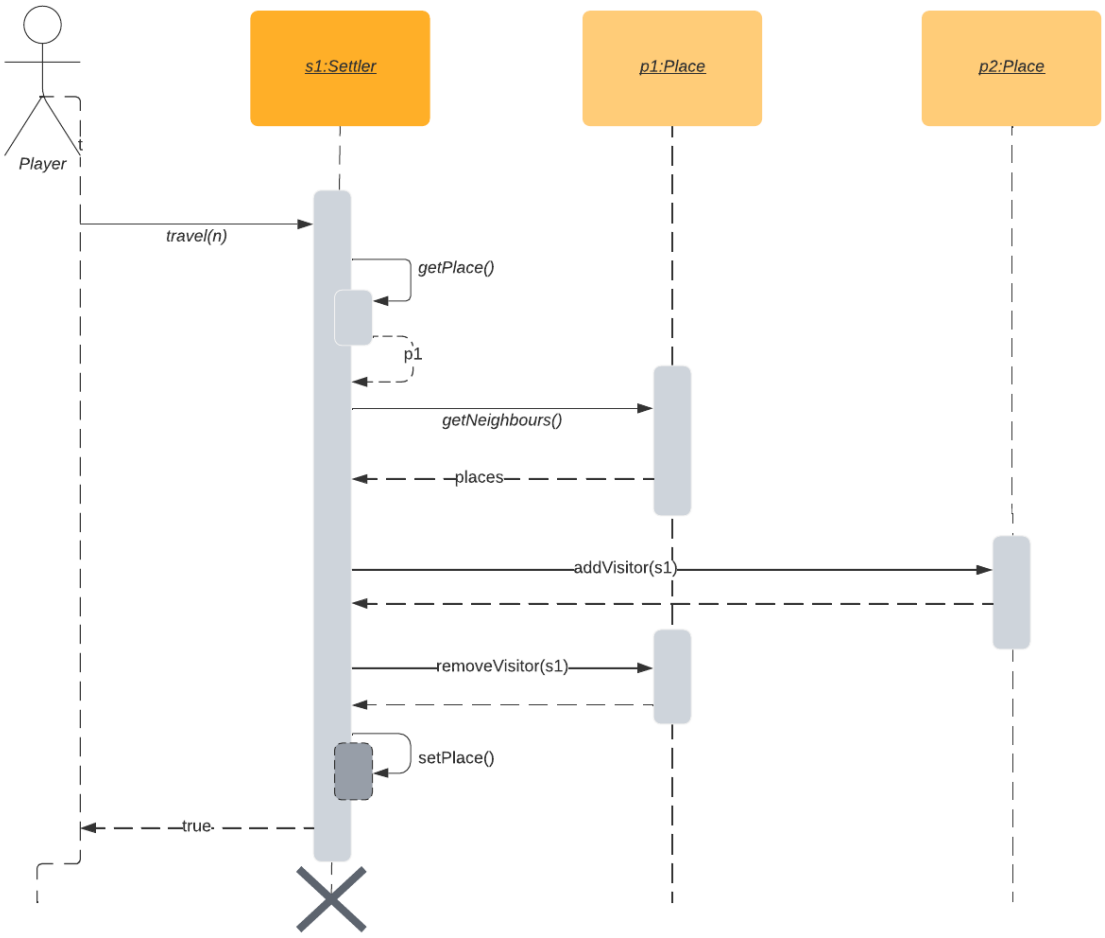
- **string getID():** gets the unique ID of an object

4.4 Sequence diagrams

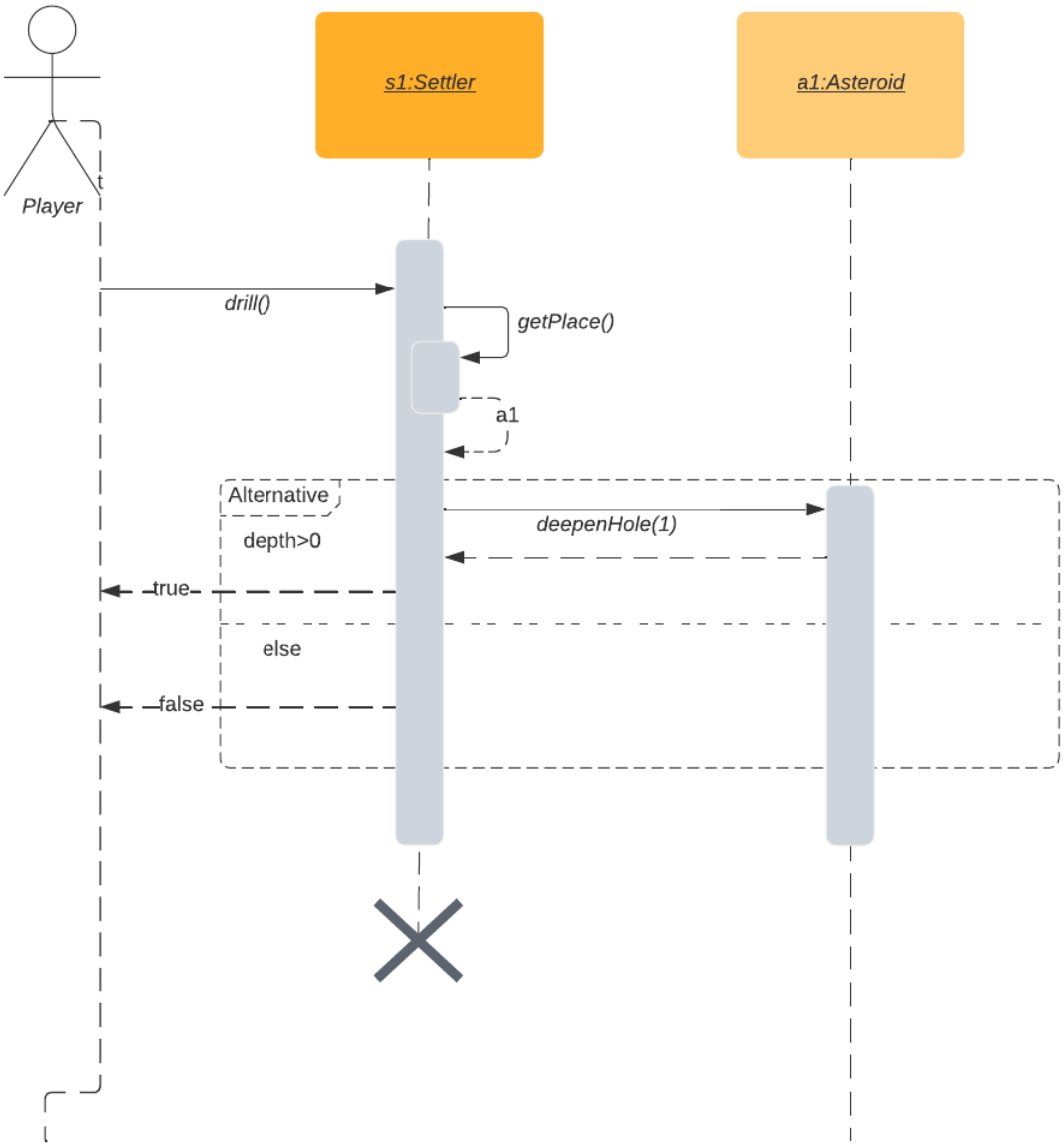
1. **Begin Game (Initialization & Settlers are positioned on the asteroids as start point of the game):**



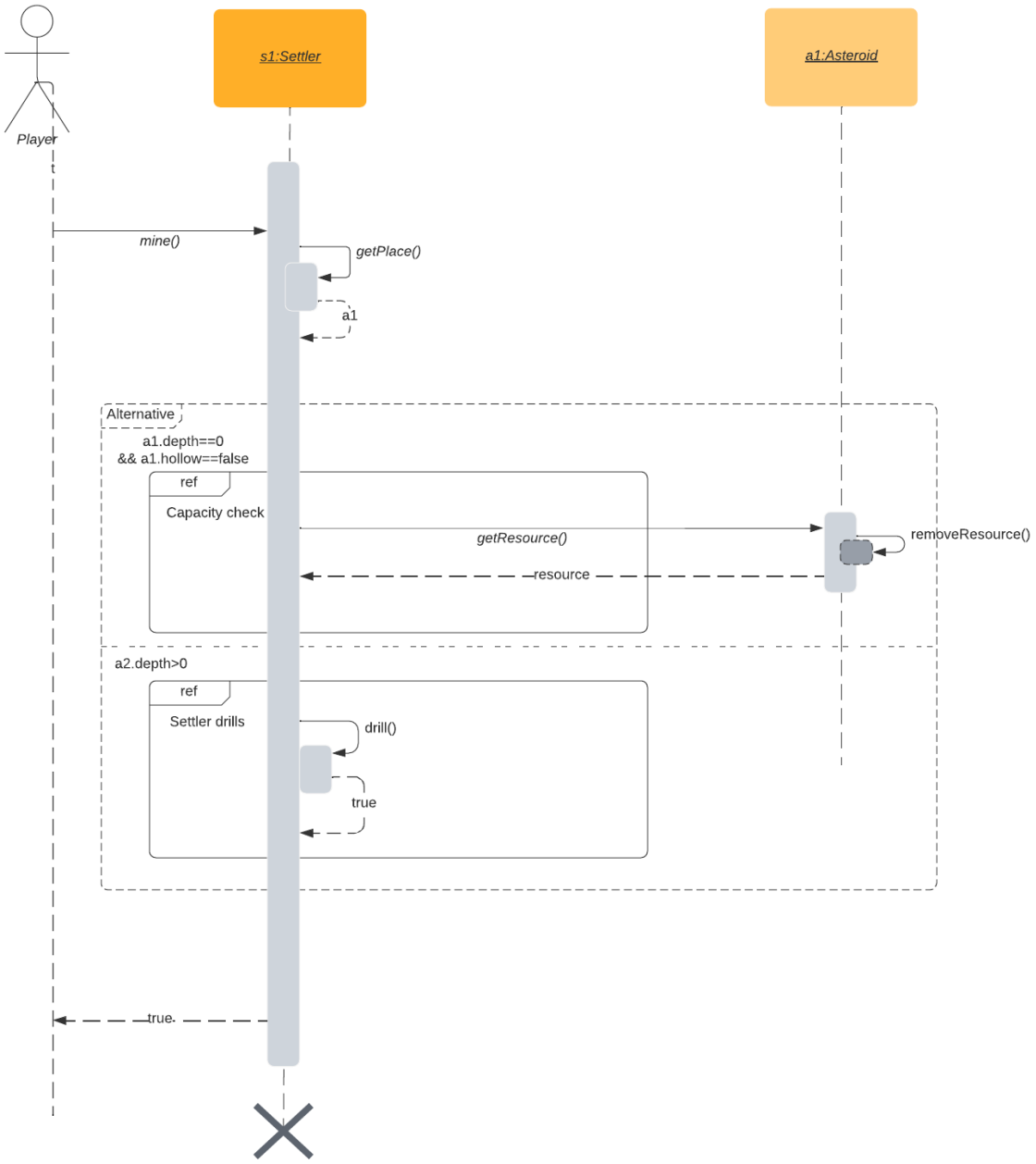
2. Settler moves



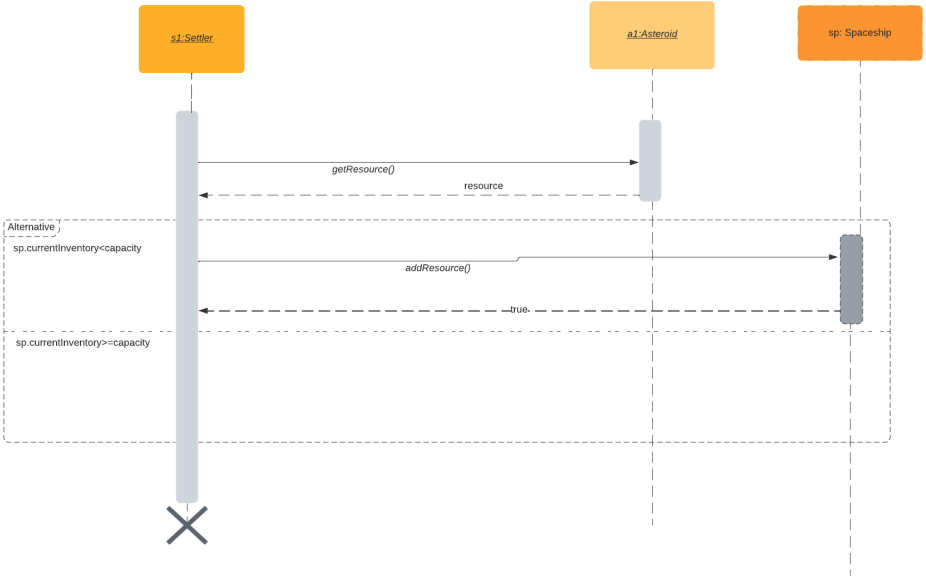
3. Settler Drills:



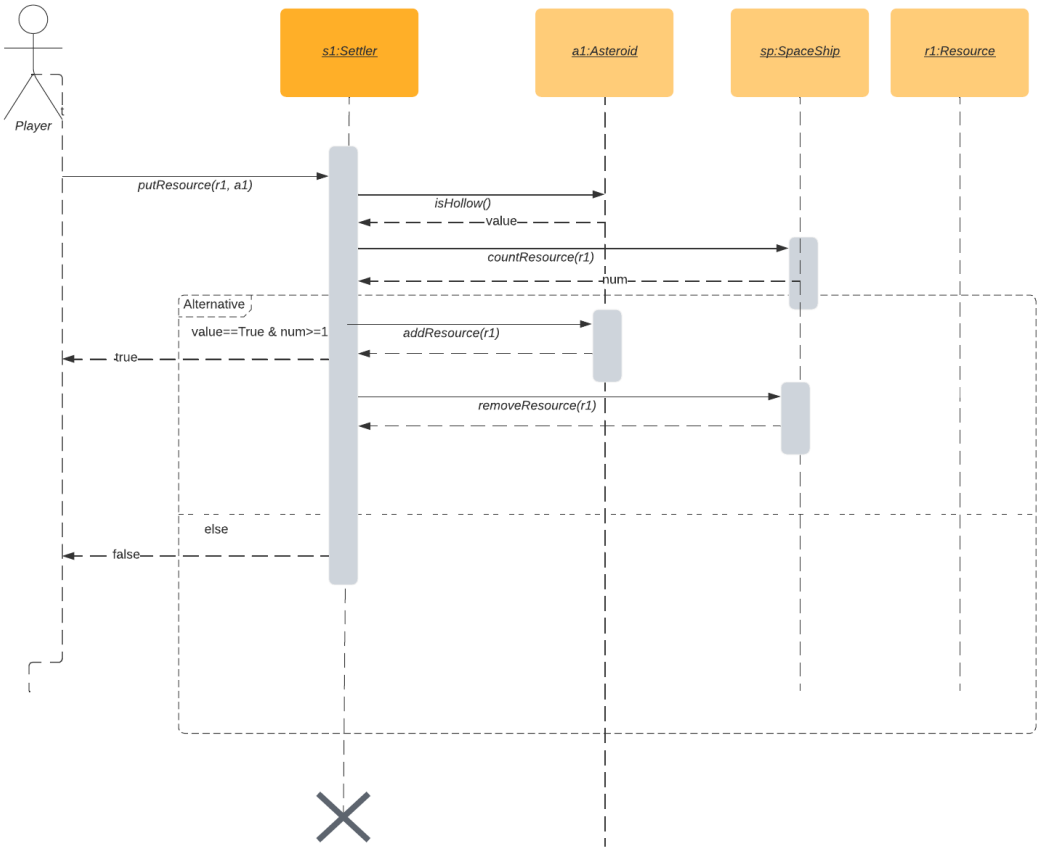
4. Settler Mines:



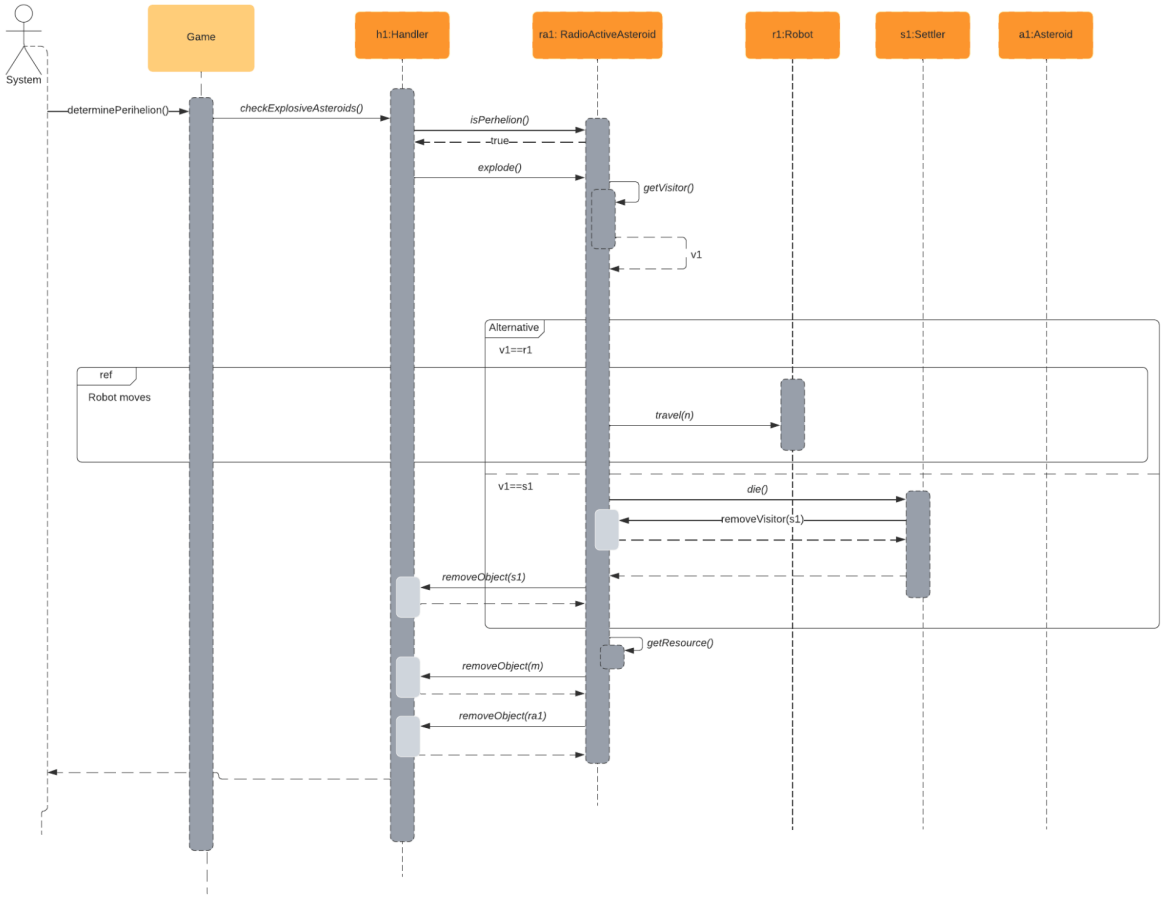
5. Capacity Check



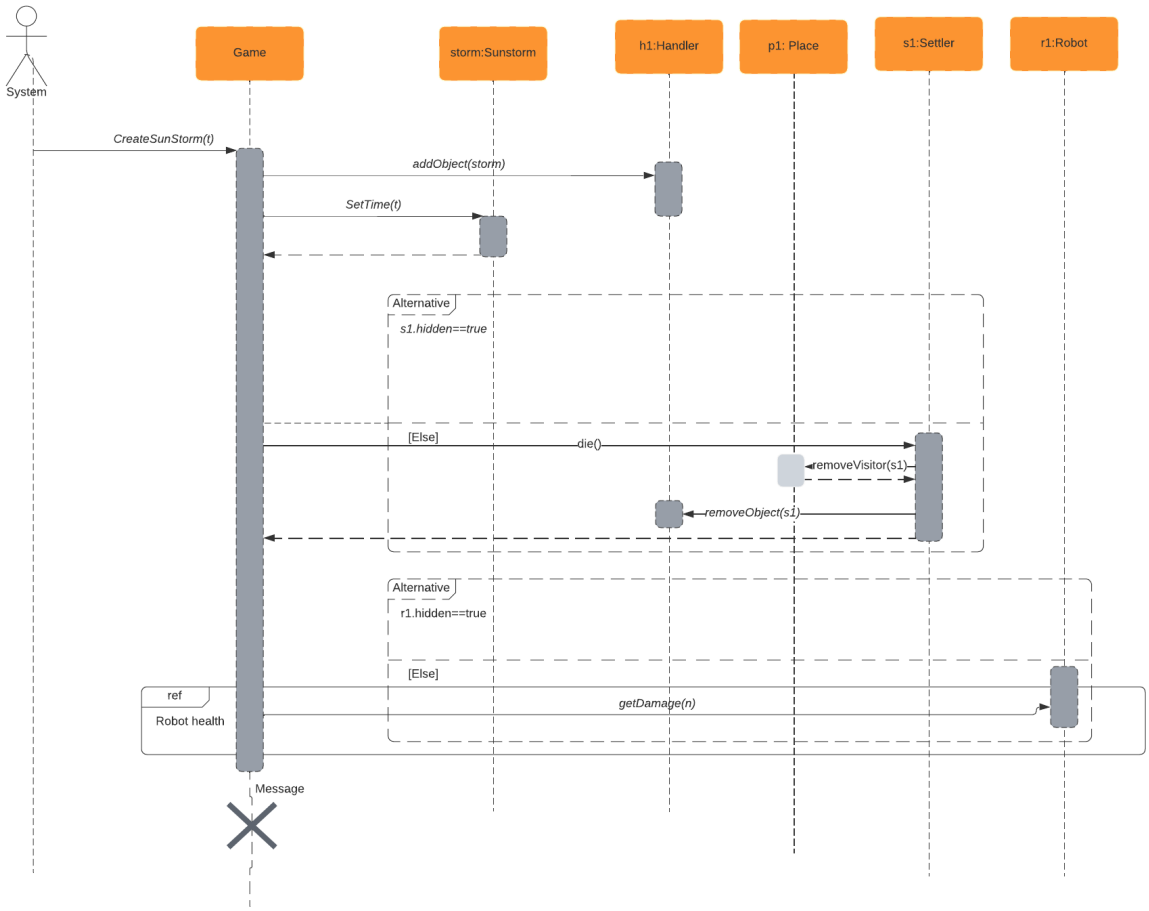
6. Settler puts resource down:



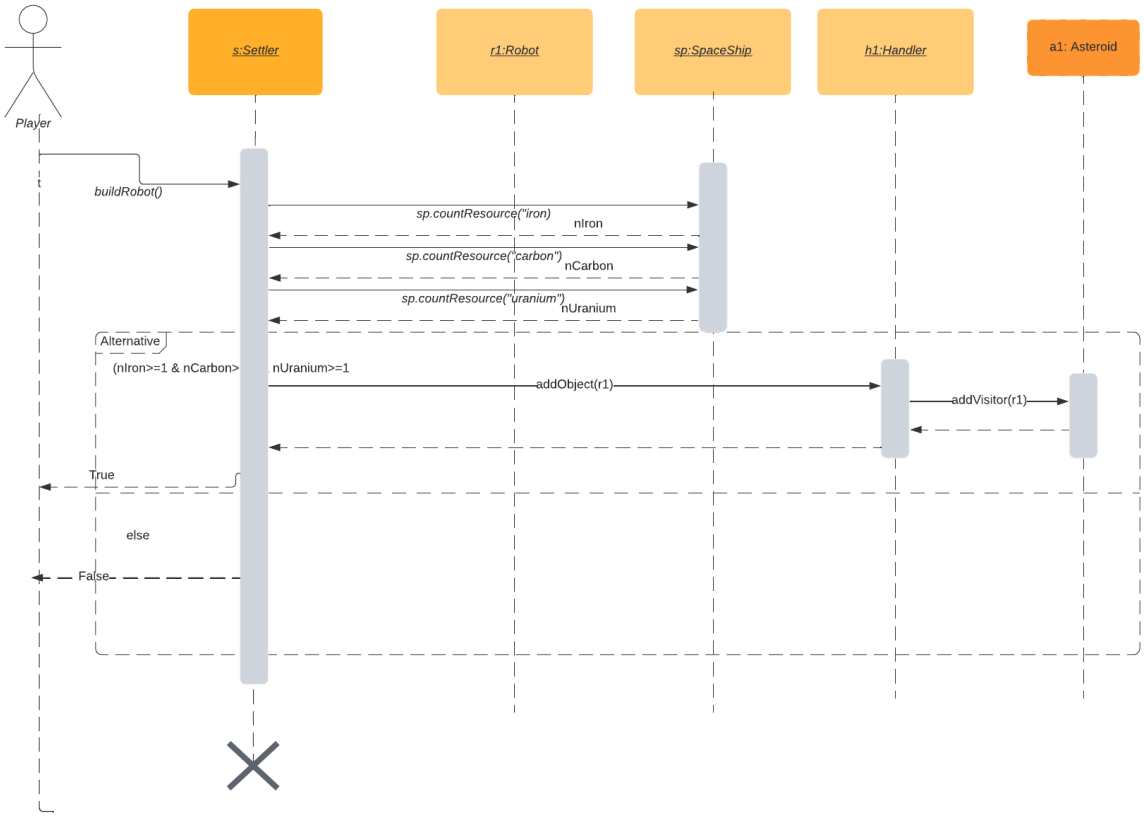
7. Asteroid Explodes (perihelion):



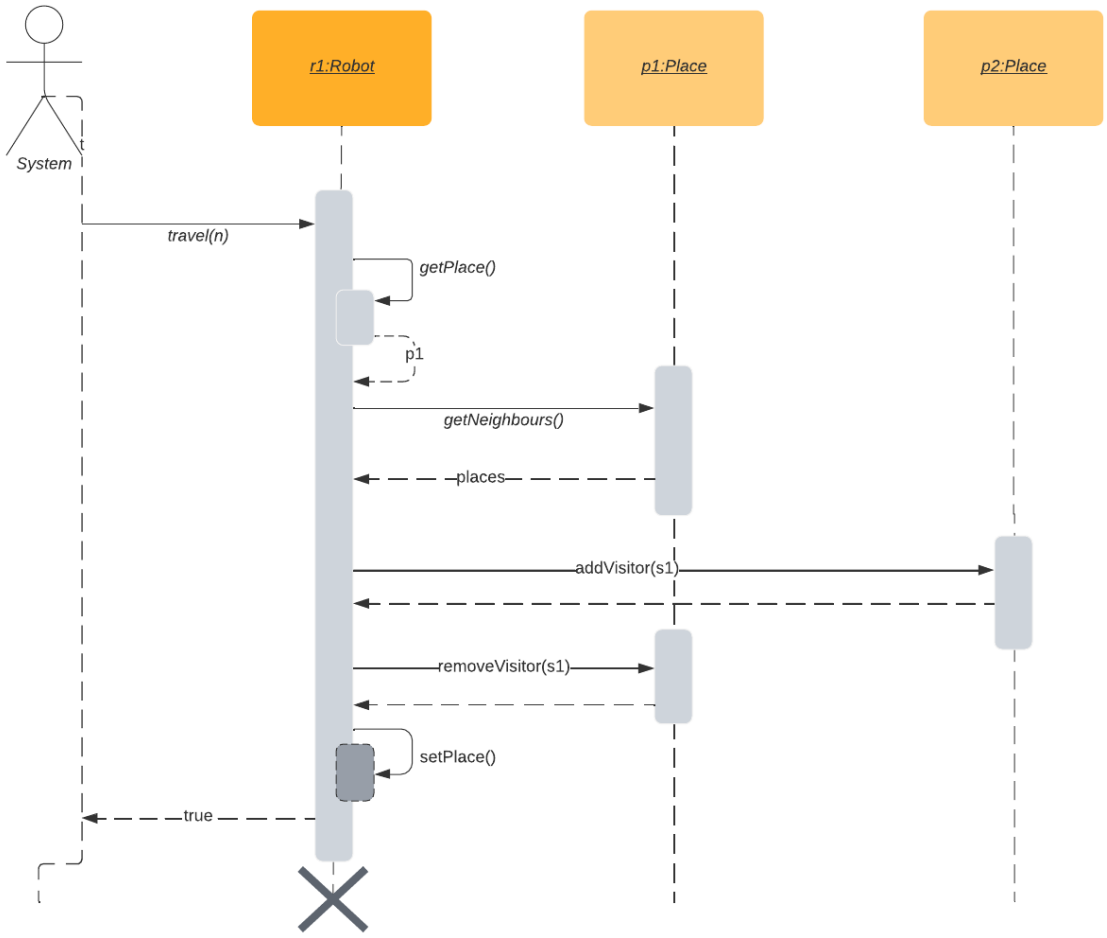
8. Sunstorm occurs:



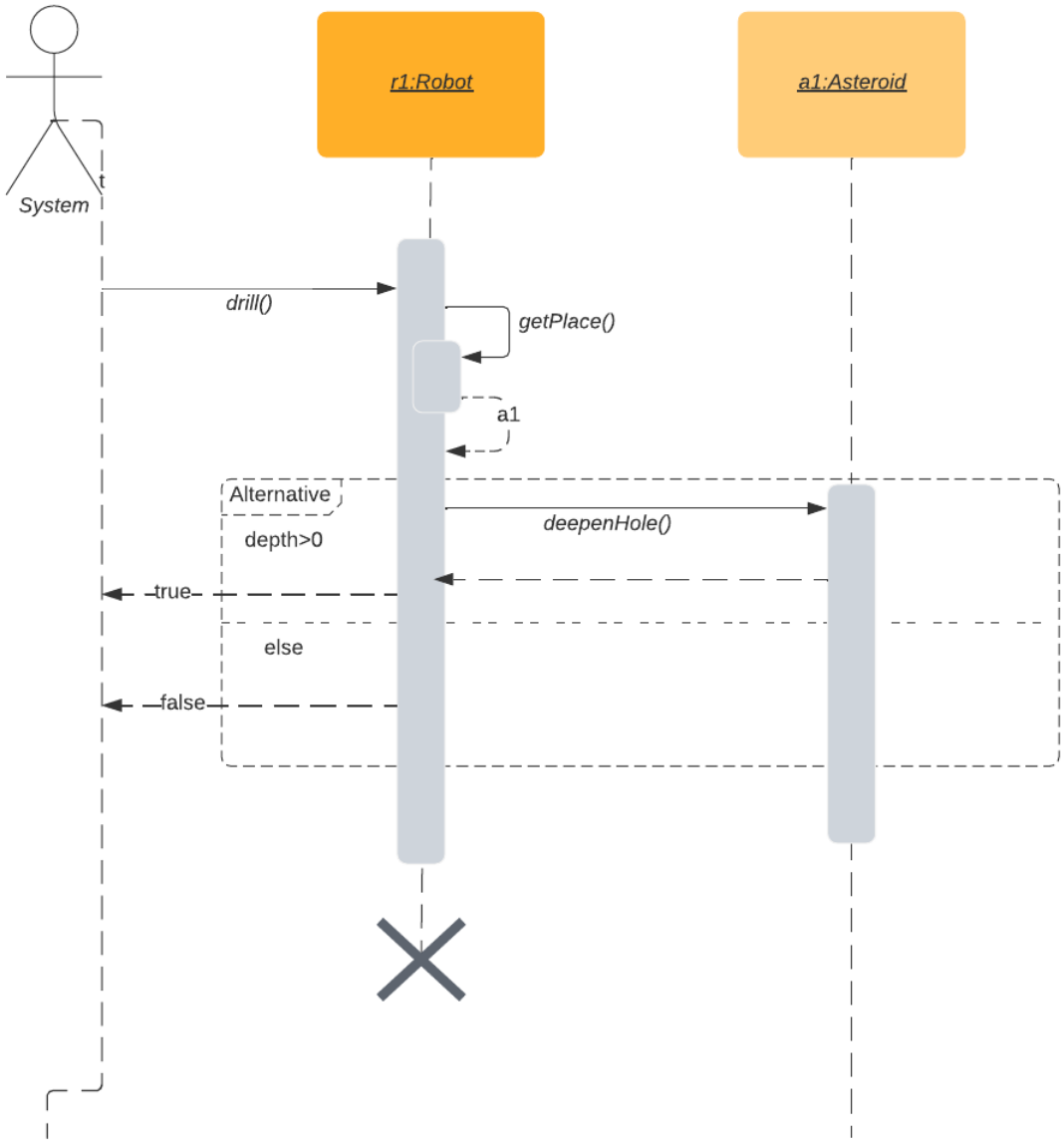
9. Robot is created:



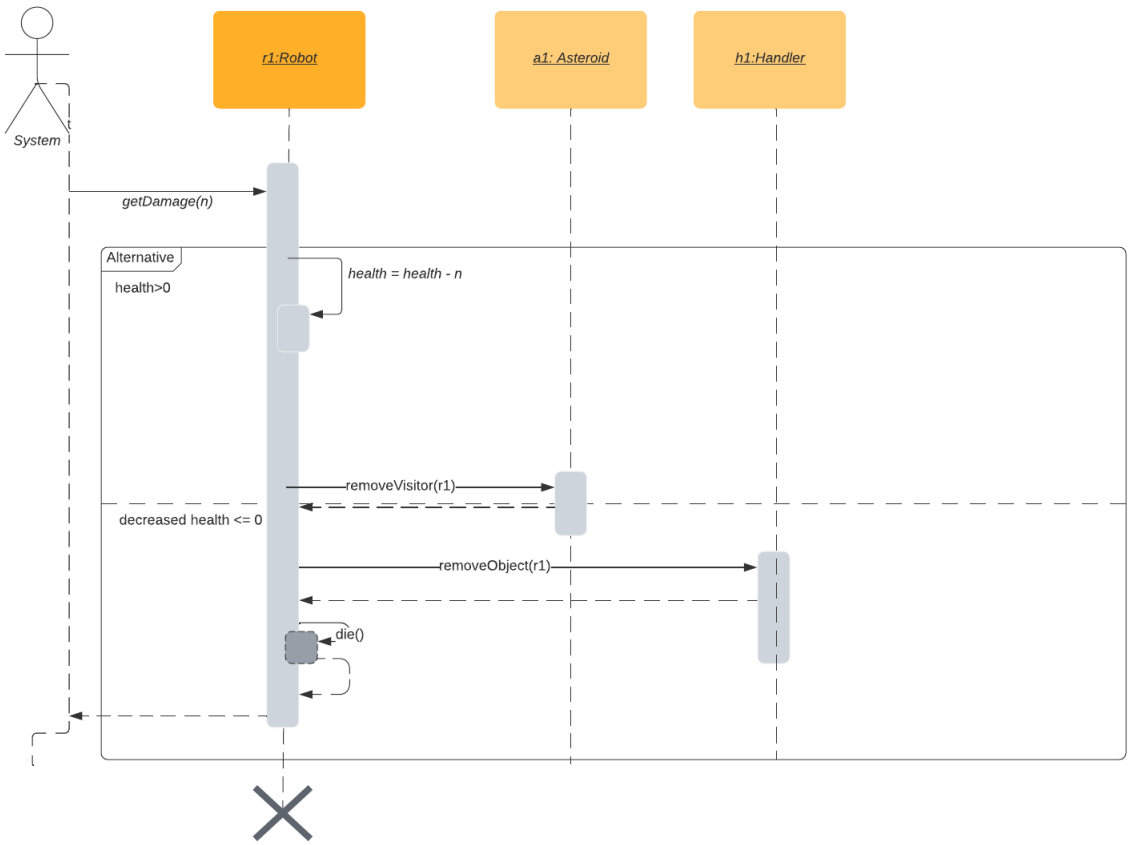
10. Robot moves:



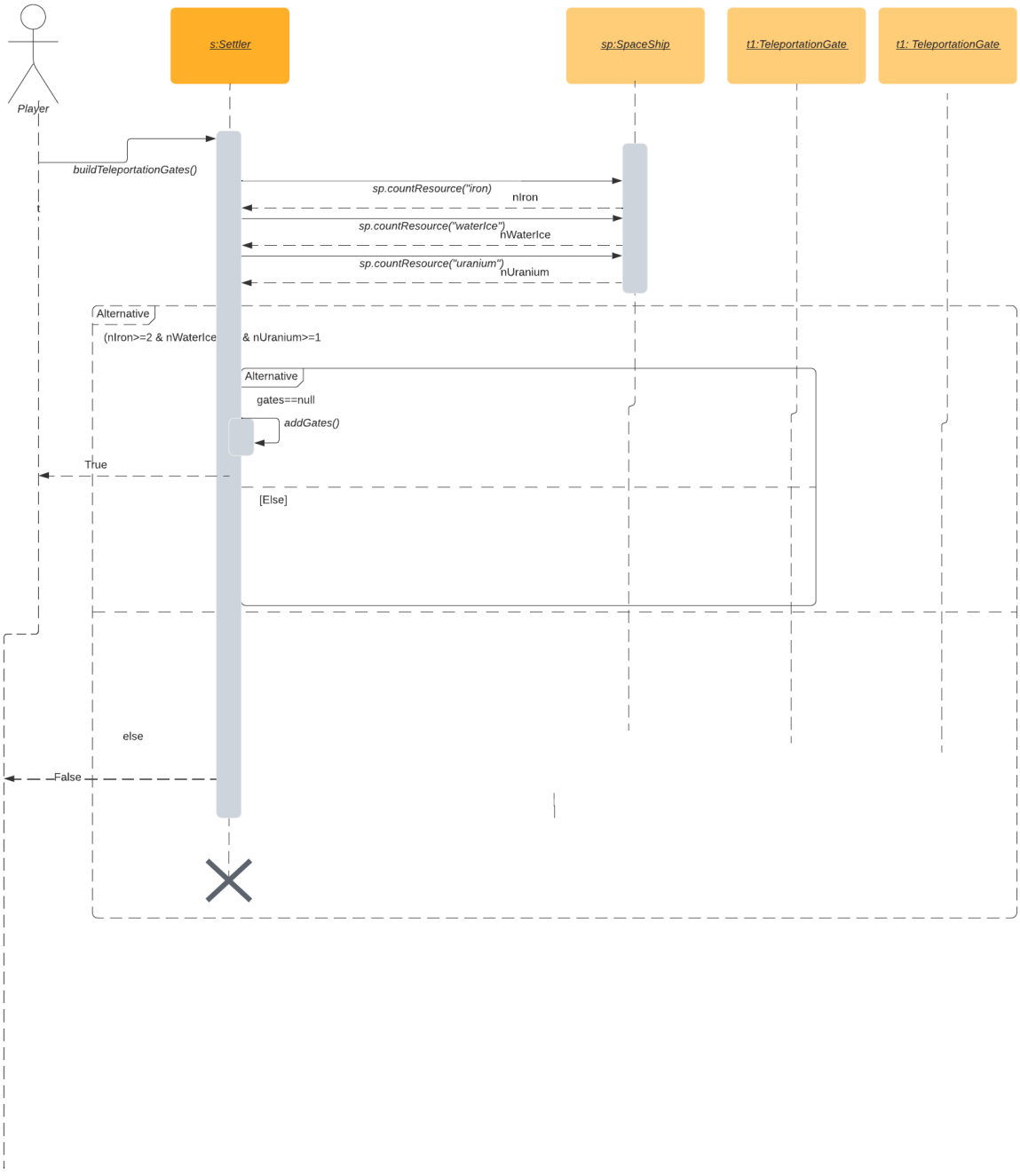
11. Robot drills:



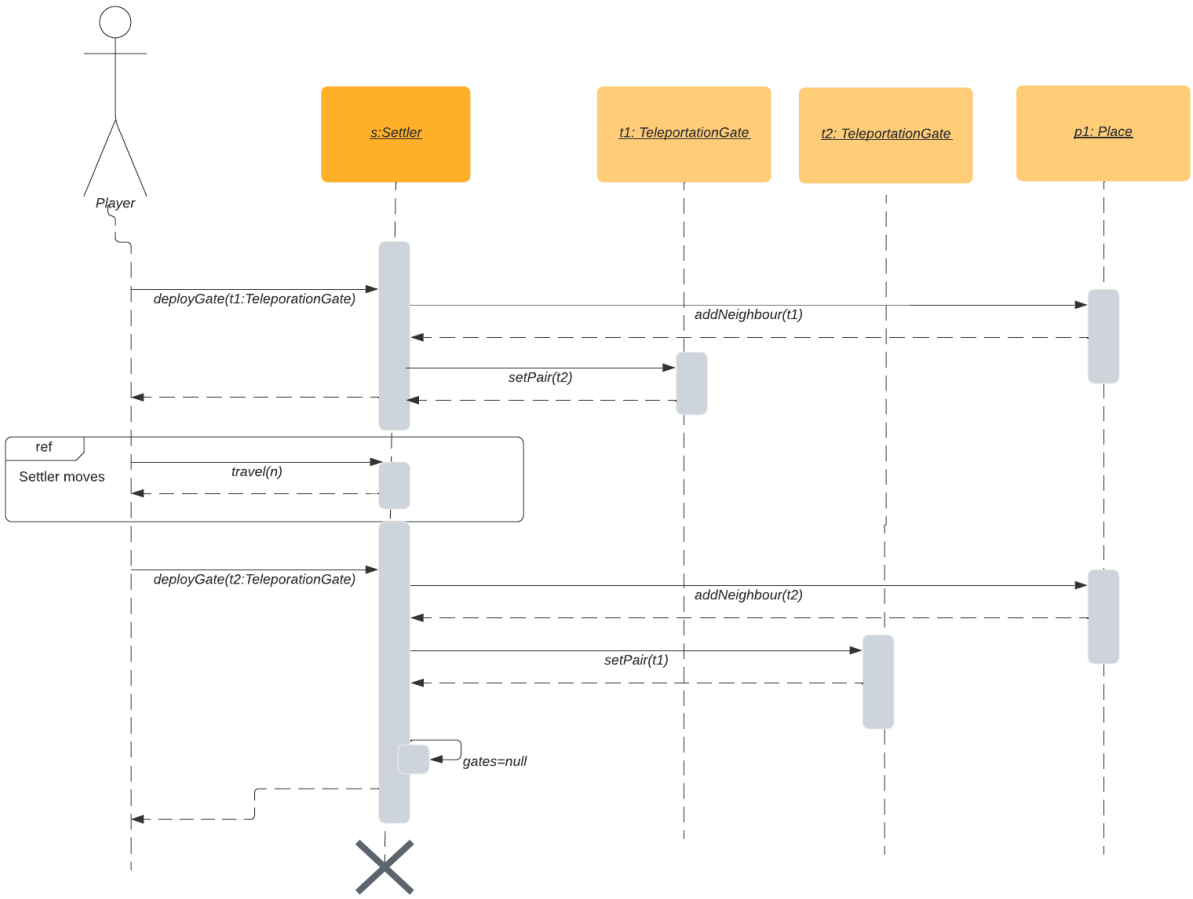
12. Robot health:



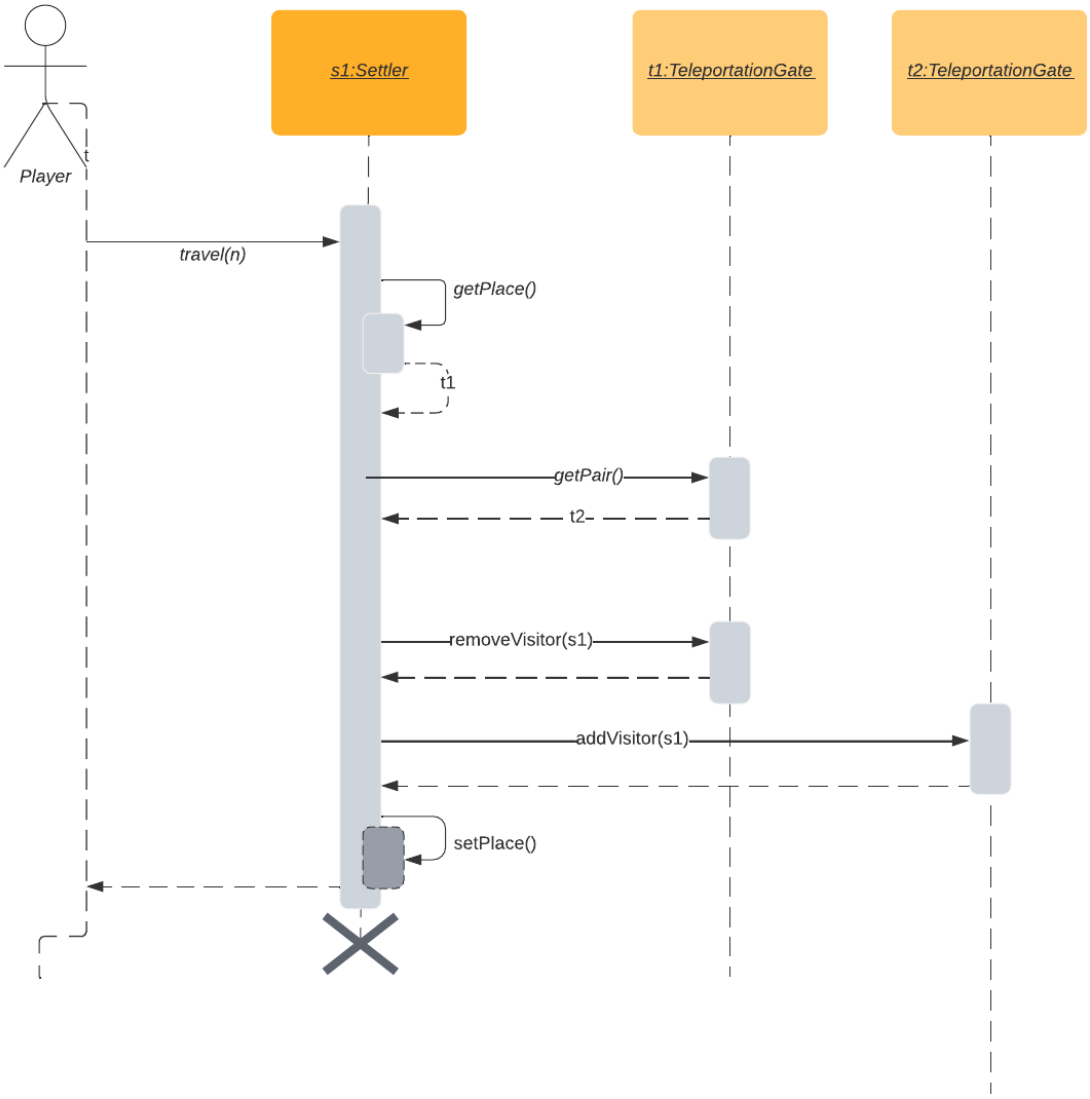
13. Pair of Teleportation-gates are created:



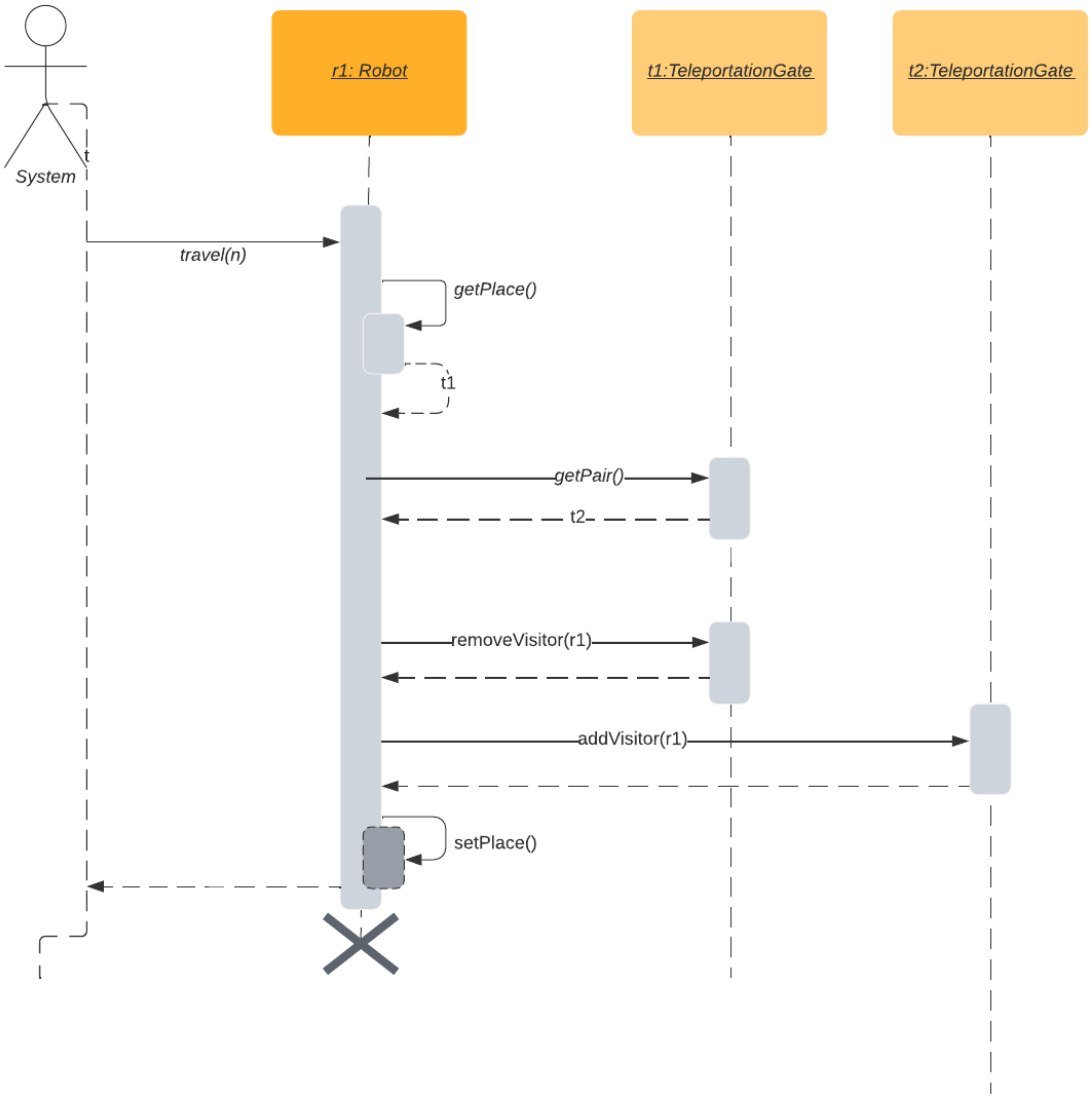
14. Teleportation gates are deployed:



15. Settler passes through teleportation-gate:

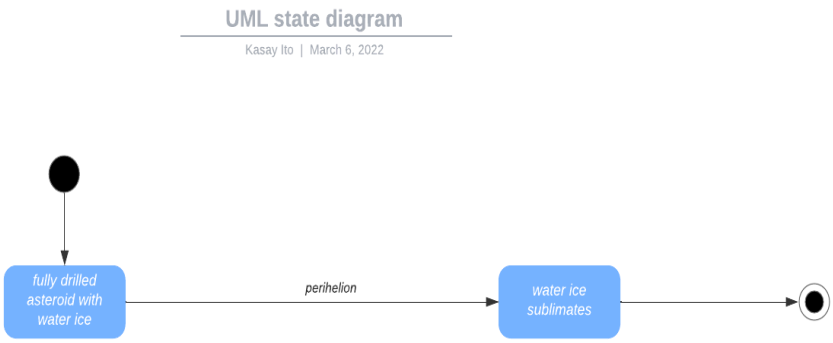


16. Robot passes through teleportation-gate:

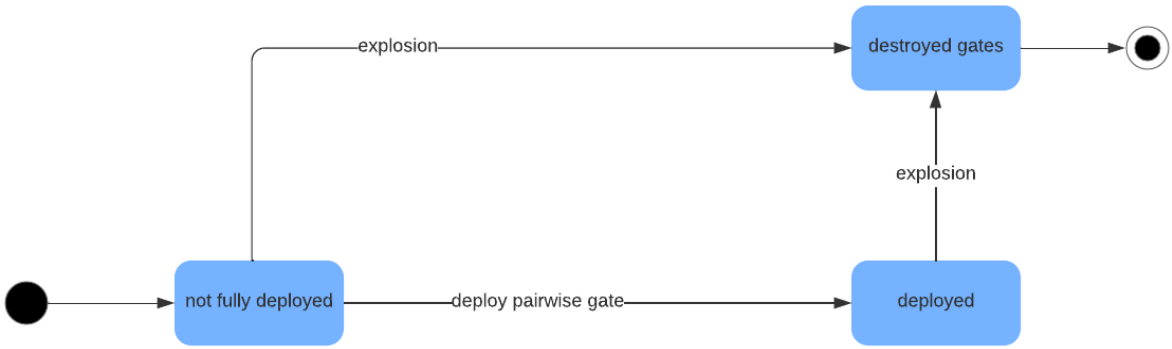


4.5 State-charts

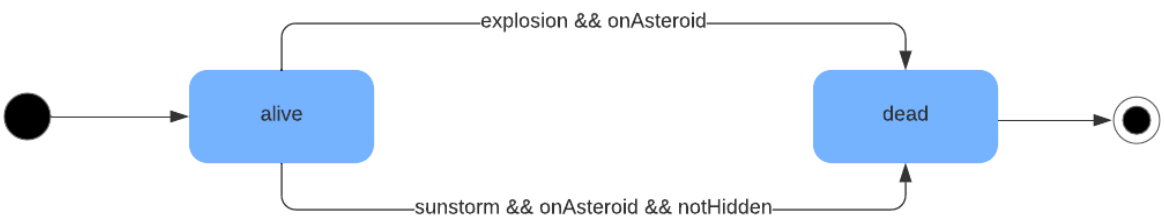
1. Ice sublimates in asteroid



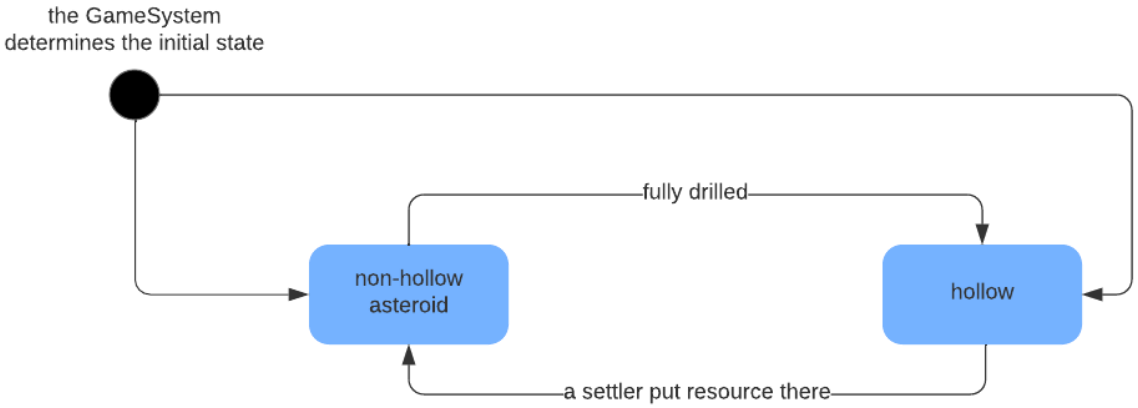
2. Teleportation gates deployment



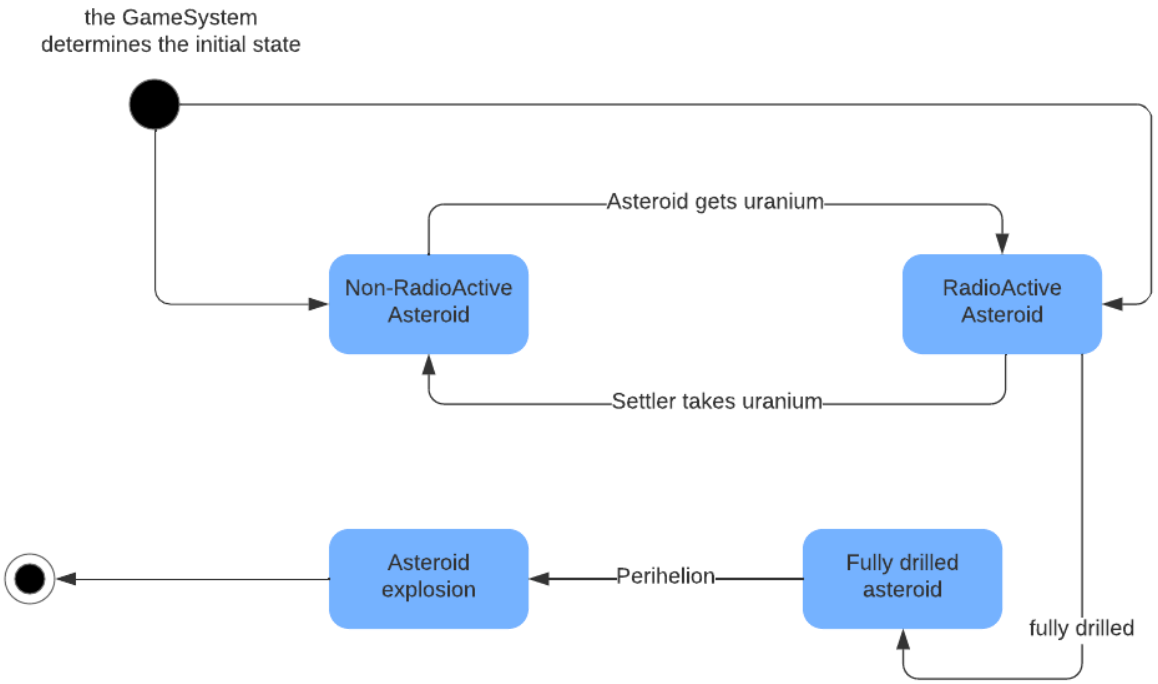
3. Settler lives



4. Hollow Asteroid

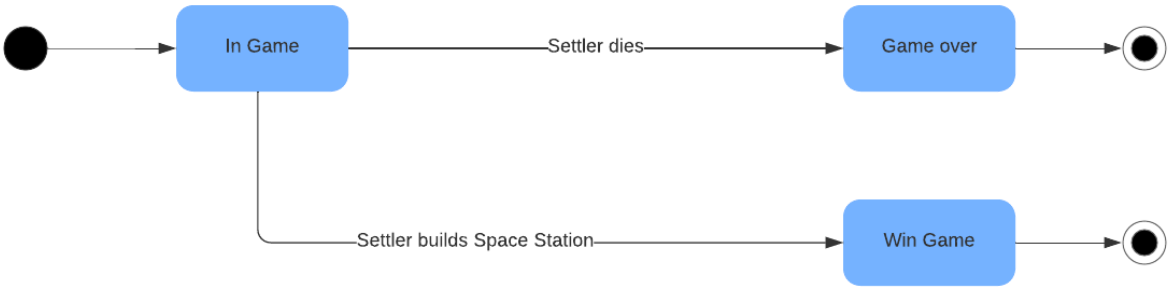


5. Asteroid explodes



6. **Robot dies**

7. **Game ends**



4.6 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
10/03/2022	4 hours	All team members	Class diagram refinement based on the feedback from the mentor, distributing tasks for each member - split into pairs that

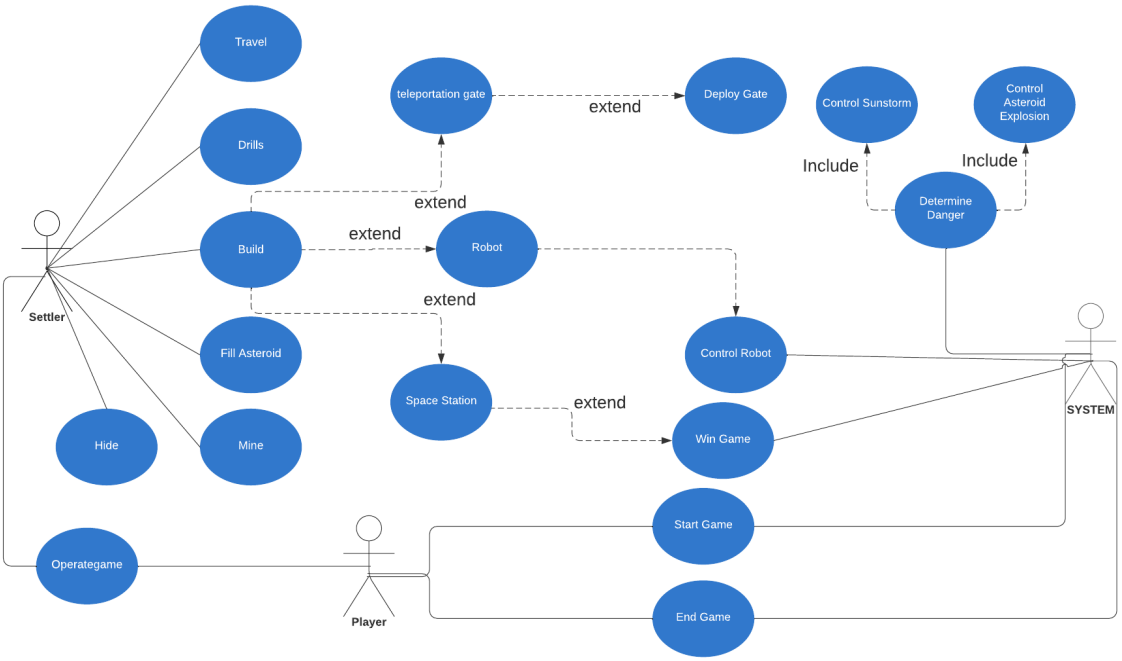
Online meeting on teams			worked together, discussed what tool to use
14/03/2022 Online meeting on teams	4 hours	Janibyek,Chaitanya, Desoki	Made changes to the class diagram. And worked on the sequence diagram by drawing required diagrams from the professor. Checked consistency of methods among class and sequence diagrams.
14/03/2022 Online meeting on teams	2.5 hours	Kasay and Neda	Kasay and Neda had the responsibility of editing the documentation. Object catalog was finished, and class descriptions were started, alongside checking consistency of class/sequence diagrams.
14/03/2022	2.5 hours	Kasay	Making small edits to class and sequence diagrams, properly labeling and ordering them, and adding them into the documentation.
15/03/2022	3h	Tushig	We decided to use the holiday to tackle some future tasks in advance, so Tushig did research for potential game engines, and some UI design-background and asteroids.

15/03/2022	2.5 hours	Neda and Tushig-	Finishing the class descriptions, and changing some details in the class diagram on the way.
16/03/2022	1.5hours	Chaitanya	Started refining the contextual portion of the documentation. Stressing more upon the mentors' comments and highlighting the relevant changes we made and why they were necessary.

5. Planning the skeleton

5.1 Real use-cases of the skeleton model

5.1.1 Use-case diagram



5.1.2 Use-case descriptions

Use-case name	Operate game
Short textual description	The Operating Game includes all controls made by the user playing the game to the settler which is the main protagonist of the game. The user controls the settler.
Actors	Settler, Player

Dialog, scenario	Player controls the actions that are executed by Settler. 1.Player can control the Settler to travel between different Places in the game. 2.Player can control the Settler to drill the rock mantle of an asteroid. 3.Player can control the Settler to Build a Teleportation Gate, Robot, or Space Station. 4.Player can control the Settler to Fill an Asteroid with a unit of resource. 5.Player can control the Settler to mine resources from the core of a drilled asteroid.

Use-case name	Travel
Short textual description	Travel function is used to change the location of the settler from where it is to someplace else which most of the time is on another asteroid. This will be initiated by the command of the player but executed by the settler
Actors	Settler
Dialog, scenario	1. The Settler can travel from one asteroid to the other using a spaceship 2. The settler can travel between asteroids through teleportation gates

Use-case name	Mine
Short textual description	Mining is one of the tasks that only the settler has to execute for the extraction of metals.Mining can involve the mining of different elements like iron, uranium etc. This will be initiated by the command of the player but executed by the settler and not the robot.
Actors	Settler
Dialog, scenario	1.The Settler mines the asteroid which contains Iron, Carbon, Uranium or Water Ice. 2.The Settler mines an asteroid containing Water Ice at Perihelion leading to it getting sublimed. 3.The Settler mines an asteroid containing Uranium at Perihelion, leading to an explosion.

Use-case name	Build
Short textual description	Build use cases give settlers ability to build Robots, Gate, and a Space Station.
Actors	Settler

Dialog, scenario	1. A Settler can build a robot if it has enough resources to do so. 2. A Settler can build a pair of Teleportation Gates if it has enough resources. 3.A Settler can build a Space Station if it collects enough resources on one asteroid, and in this case, the Player wins the game.

Use-case name	Drill
Short textual description	Drills mantle is one of the function executed by the settler to drill the asteroid in the core in order to reach the metals
Actors	Settler, robot
Dialog, scenario	Drill action could be done by both Settler and Robot. 1. Player controls Settler to execute this action Robot is controlled by the system itself, as an AI, to execute this action. If drill action is being executed while radioactive asteroid is on the perihelion, it will explode and kills the Settler and displace the Robot to another neighboring asteroid

Use-case name	Hide
Short textual description	Hide is one of the crucial functions that is executed by settler and robots to escape from Sunstorm.
Actors	Settler, robot
Dialog, scenario	Hide action could be done by both Settler and Robot. 1. Player controls Settler to execute this action Robot is controlled by the system itself, as an AI, to execute this action. Settler and Robot can only hide in hollow Asteroid.

Use-case name	Space Station
Short textual description	Building the Space Station will be the aim of the settler. Since it's one of the things that the settler builds, extend is used

Actors	Settler
Dialog, scenario	1- Player can control a settler which has to build the space station when it mines at least three units of resources then gather them in one asteroid, and it is one of the cases where the settlers can win the game. 2- Player cannot execute the build space station action unless necessary resources are collected.

Use-case name	Deploy Gate
Short textual description	Settler deploys teleportation-gates in the vicinity of the asteroid where settlers is on.
Actors	Settlers
Dialog, scenario	Player executes the action of Settler to deploy the teleportation gates one at a time in different places. 1. After deployment of pair gates, Settler and Robots could pass through these gates to travel to neighboring asteroid.

Use-case name	Robot
Short textual description	The settler builds the Robot which itself will perform different functions like drilling and will be controlled by the system and not the settler
Actors	System
Dialog, scenario	1. Players can control a settler to build an autonomous robot with necessary resources. 2.Robot is controlled by a system. 3.Robots can only travel in the range of Asteroids and only drill holes. 4.Sunstorm can damage robots unless they are hidden in a hollow asteroid.

Use-case name	Determine Danger
Short textual description	The major task of determining danger is to control all the possible dangers that would happen on an Asteroid Belt.
Actors	System
Dialog, scenario	1. System creates a sun storm at some time of the game. 2. System controls all the explosive asteroids.

Use-case name	Fill Asteroid
Short textual description	Settlers can fill hollow asteroids with resources.
Actors	Settler
Dialog, scenario	If the asteroid is hollow, the player can control the settlers to fill the hollow asteroid with a unit of resource.

Use-case name	Teleportation gate
Short textual description	Settlers can build a pair of teleportation gates by using up two units of iron, a single unit of water ice and a single unit of uranium. This is deployed on an asteroid and is used to transfer the settler from one point to another.
Actors	Settler, Robot
Dialog, scenario	1. Players can control a Settler which can build the teleportation gates with sufficient resources. 2. The gates can be deployed in the vicinity of the asteroid. 3. Settlers can carry the freshly built gates with themselves, but at the same time a single settler can only bring a pair of gates.

Use-case name	Control Robot
Short textual description	The Robots, made by Settlers, are actually controlled by the system through AI, majorly the robots are created to ease the task of the settler as it can survive a sun storm and explosion and can help the settler in drilling.
Actors	System, Robot
Dialog, scenario	1.The Robot can drill the asteroids. 2. Sun storm occurs and the robot gets damaged if he is not hidden. 3. Sun storm occurs and the robot is saved because he is hidden. 4. If an explosion occurs then the robot is sent to another neighboring asteroid with no damage.

Use-case name	Start Game
Short textual description	System starts the game.
Actors	System, Player
Dialog, scenario	When the Player starts the game, the System initializes the important objects in the game.

Use-case name	End Game
Short textual description	When the players die or they achieve the goal, the game will finish.
Actors	System, Players
Dialog, scenario	The game can end in two different ways. 1. If all the settlers die, the system will end the game. 2. If settlers mine at least three units of each resource and collect the materials on a single asteroid, they can build a space station and the player wins the game.

Use-case name	Control Sun Storm
Short textual description	Sunstorm is one of the possible dangers that would happen on Asteroid Belt.And it is dangerous for the Settlers and Robots.
Actors	System
Dialog, scenario	1- System will generate a sunstorm after a random amount of time and it is dangerous when it arrives in the asteroid belt.

	2- A settler can remain alive if it hides in the core of the hollow asteroid. A Sun Storm can kill a settler if the settler is not hidden in the core of a hollow asteroid. 3- If a robot is hit by a sunstorm, its health will be decreased by some value. A Sun Storm can damage a robot if the robot is not hidden in the core of a hollow asteroid.

Use-case name	Determine Danger
Short textual description	This is an abstract way to present the major dangers the settler and robot can come across during the game. This further extends to specified cases like explosion and sun storm.
Actors	System
Dialog, scenario	1. A sun storm occurs in the game. 2. An explosion occurs in the game.

Use-case name	Control Asteroid Explosion
Short textual description	It controls what will happen when settlers at perihelion and it mines to find Uranium (radioactive element).

Actors	System
Dialog, scenario	1. Asteroids become explosive when they contain uranium. 2. A completely drilled radioactive asteroid explodes at perihelion. 3. When the asteroid explodes, it kills any Settler on it and displaces the Robots to the neighboring asteroid.

5.2 Plans of the skeleton's UI, dialogs

5.2.1 Player starts the game

After the user opens the game, A screen will appear to the user: "Click your mouse to start the game"

1. **User input:** $Mouse_Click$

   System output: StartGame()

   addObject(s1)

   addObject(a1)

   addResource(r1)

   addVisitor(s1)

   setPlace(a1)

Remark: addObject(a1) and addResource(r1) are called multiple times (the number of corresponding asteroids in the game)

5.2.2 Settler traveling

To travel through the asteroid belt, user has to press $UP, DOWN, RIGHT, LEFT$ keys.

1. **User input:** $UP, DOWN, RIGHT, LEFT$

   System output: travel()

   getPlace(): p1

   getNeighbours(): places

   Click on a place to travel!

2. **User input:** $MOUSE_CLICK$ on place p2

   System output: addVisitor(s1)

   removeVisitor(s1)

   setPlace()

5.2.3 Settler Drills

To drill the not hollow asteroid, user has to press D_KEY.

1. **User input:** D_KEY

   System output: drill()

   getPlace(): a1

   Is depth greater than 0?

2. **User input:** yes

   System output: deepenHole(1)

5.2.4 Settler Mine

To mine the asteroid at aphelion, user has to press m key to exploit the resources.

1. **User input:** M_KEY

   System output: mine()

   getPlace(): a1

   Is depth 0 and is the asteroid non-hollow?

2. **(case A) User input:** yes

(case A) System output: getResource(): r1

removeResource()

See sequence diagram 4.4.5. “Capacity Check”.

(case B) User input: no

(case B) System output: drill()

5.2.5 Fill Asteroid

In order to Settler fill the hollow asteroid, the user(player) first selects a resource by mouse click and then presses F_KEY to fill it.

1. **System output:** selects r1 parameter as a value
2. **User input:** $MOUSE_CLICK$ on resource r1

   System output: A resource is selected.

3. **User input:** F_KEY

   System output: putResource(r1)

   getPlace(): a1

   isHollow(): value

   countResource(r1): num

   Is the asteroid hollow and do you have enough resources?

4. **User input:** yes

   System output: addResource(r1)

   removeResource(r1)

5.2.6 Building the Robot

To build a robot user has to press the R_KEY if he has necessary resources available.

1. **User input:** R_KEY

   System output: buildRobot()

   countResource("iron”): nIron

countResource(“carbon”): nCarbon

countResource(“uranium”): nUranium

Do you have enough resources?

2. **User input:** yes

   System output: addObject(r1)

   addVisitor(r1)

   Robots are automatically controlled by the system from now on.

Remark: Robots are automatically controlled by the system since when it is built by the settler. Therefore, the test cases related operation against the robots is not denoted.

5.2.7 Building teleportation Gates

To build a robot user has to press the T_KEY if he has necessary resources available.

1. **User input:** T_KEY

   System output: buildTeleportationGates()

   countResource("iron”): nIron

   countResource(“waterIce”): nWaterIce

   countResource(“uranium”): nUranium

   Do you have enough resources?

2. **User input:** yes

   System output: addGates()

5.2.8 Deploying the Gate

To deploy the already built pair of gates, user has to press G_KEY. User can deploy one gate at a one place and another to other place.

1. **User input:** G_KEY

   System output: deployGate(t1)

   addNeighbour(t1)

   setPair(t2)

Please travel somewhere!

2. **User input:** $UP, DOWN, RIGHT, LEFT$

   System output: travel()

3. **User input:** G_KEY

   System output: deployGate(t2)

   addNeighbour(t2)

   setPair(t1)

   gates=null

5.2.9 Settler Hides

To hide the settler at the core of the hollow of the asteroid which the settler is stepping on, the user has to press H_KEY.

1. **User input:** H_KEY

   System output: hide()

   getPlace(): a1

   a1.isHollow(): bool

   Is this asteroid hollow?

2. **User input:** yes

   System output: isHidden = true

   Succeeded to hide.

3. **User input:** no

   System output: isHidden = false

   Failed to hide.

5.2.10 Sunstorm occurs

Sun Storm is created by a system with fixed time. In order to escape from sunstorm, User can hide settler to core of hollow asteroid which it is stepping on by pressing H_Key

1. **User input:** test sunstorm

   System output: createSunstorm(10) ~ System created sunstorm with fixed time 10 sec

   addObject(sunstorm)

   setTime(t)

   SunStorm is occurring. Do you want to hide?

2. **(case A) 1. User input:** yes

   (case A) System output: Please press the “H” key!

   (case A) 2. User input: H_Key

   (case A) System output: hide()

   (case B) User input: no

   (case B) System output: Nothing has happened.

3. **System output:** colisionWith(a1)

   getVisitor(): visitor

   Are you there and not hidden?

4. **User input:** yes

   System output: die()

   See sequence diagram 5.3.12. “Losing the game”.

Remark: colisionWith(a1) and getVisitor() are called multiple times (the number of corresponding asteroids in the game)

Remark: This test case is not commited to the authentic use case, however, you can demonstrate the functionality of it by the user input “test sunstom”. Furthermore, createSunstorm() will be called by the System after the skeleton.

5.2.11 Control Asteroid Explosion

Fully Drilled and RadioActive Asteroids explode at the perihelion.

1. **User input:** test explosion

   System output: determinePerihelion()

   h1.checkExplosiveAsteroids()

   ra1.isPerhelion(): bool

   Is it on perihelion?

   System: yes (automatically checked by the system because none of user inputs are required)

   ra1.getVisitor(): v1

   Are you there?

2. **User input:** yes

   System output: die()

   See sequence diagram 5.3.12. “Losing the game”

Remark: isPerhelion() is called multiple times (the number of corresponding radioactive asteroids in the game)

Remark: This test case is not commited to the authentic use case, however, you can demonstrate the functionality of it by the user input “test explosion”. Furthermore, determinePerihelion() will be called by the System after the skeleton.

5.2.12 WaterIce sublimates

If the asteroid containing water ice is on the perhelion and fully drilled, then there must be a transition to sublimated state.

User input: test sublimate

System output: determinePerihelion()

a1.isPerhelion(): bool

a1.getResource(): Resource

Is it on perihelion, WaterIce and fullyDrilled?

System: yes (automatically checked by the system because none of user inputs are required)

Remark: isPerhelion() and getResource() are called multiple times (the number of corresponding asteroids in the game)

Remark: This test case is not commited to the authentic use case, however, you can demonstrate the functionality of it by the user input “test sublimate”. Furthermore, determinePerihelion() will be called by the System after the skeleton.

5.2.13 Winning the Game

The player wins the game by building a spacestation. To build a spacestation, pressing S_KEY is required.

1. **User input:** S_KEY

   System output: buildSpaceStation()

   countResource("iron”): nIron

   countResource(“waterIce”): nWaterIce

   countResource(“carbon”): nCarbon

   countResource(“uranium”): nUranium

   Do you have enough resources?

3. **User input:** yes

   System output: endGame()

5.2.14 User exits the Game

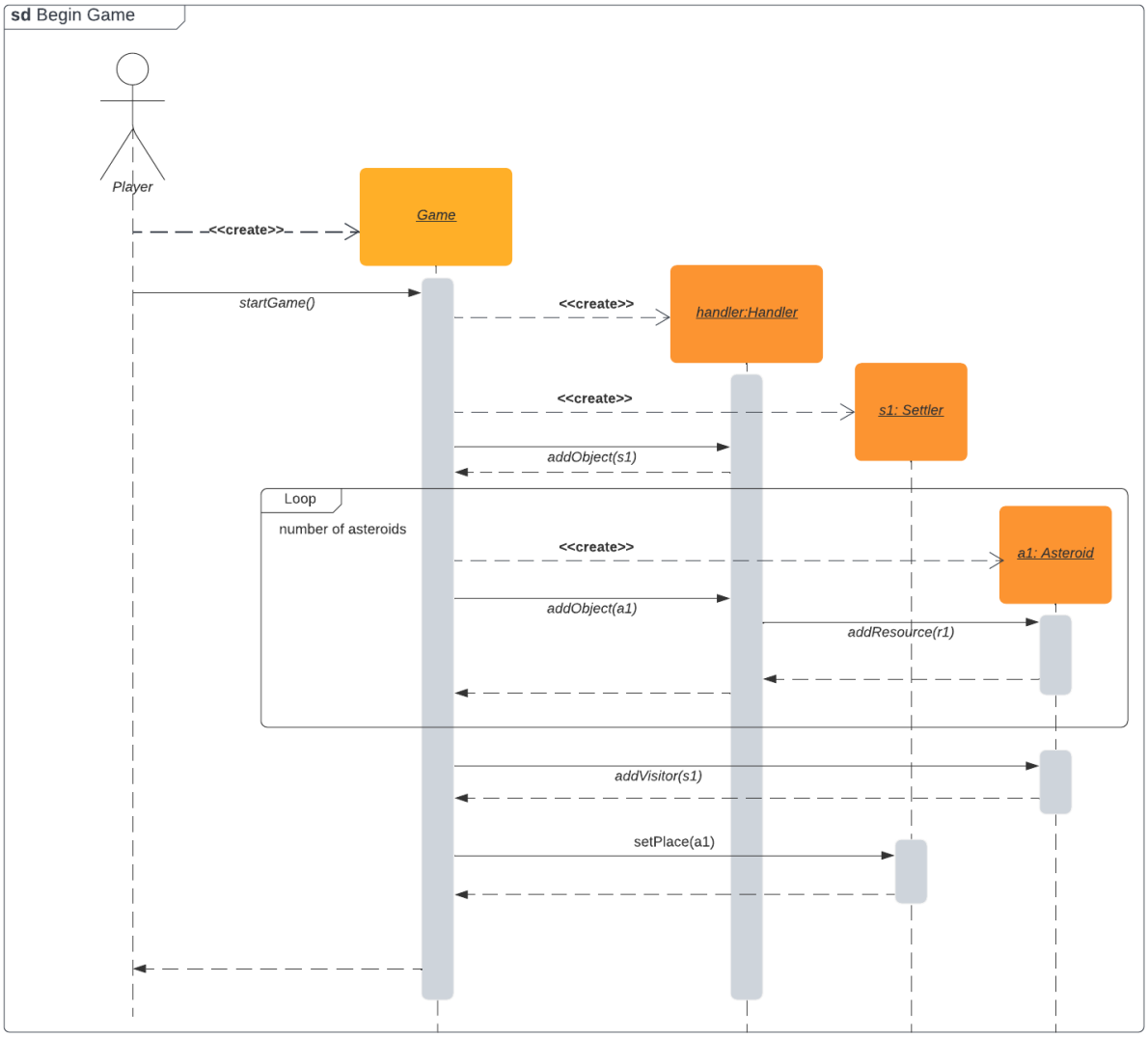
User can exit the game by pressing $ESCAPE_KEY$.

2. **User input:** $ESCAPE_KEY$

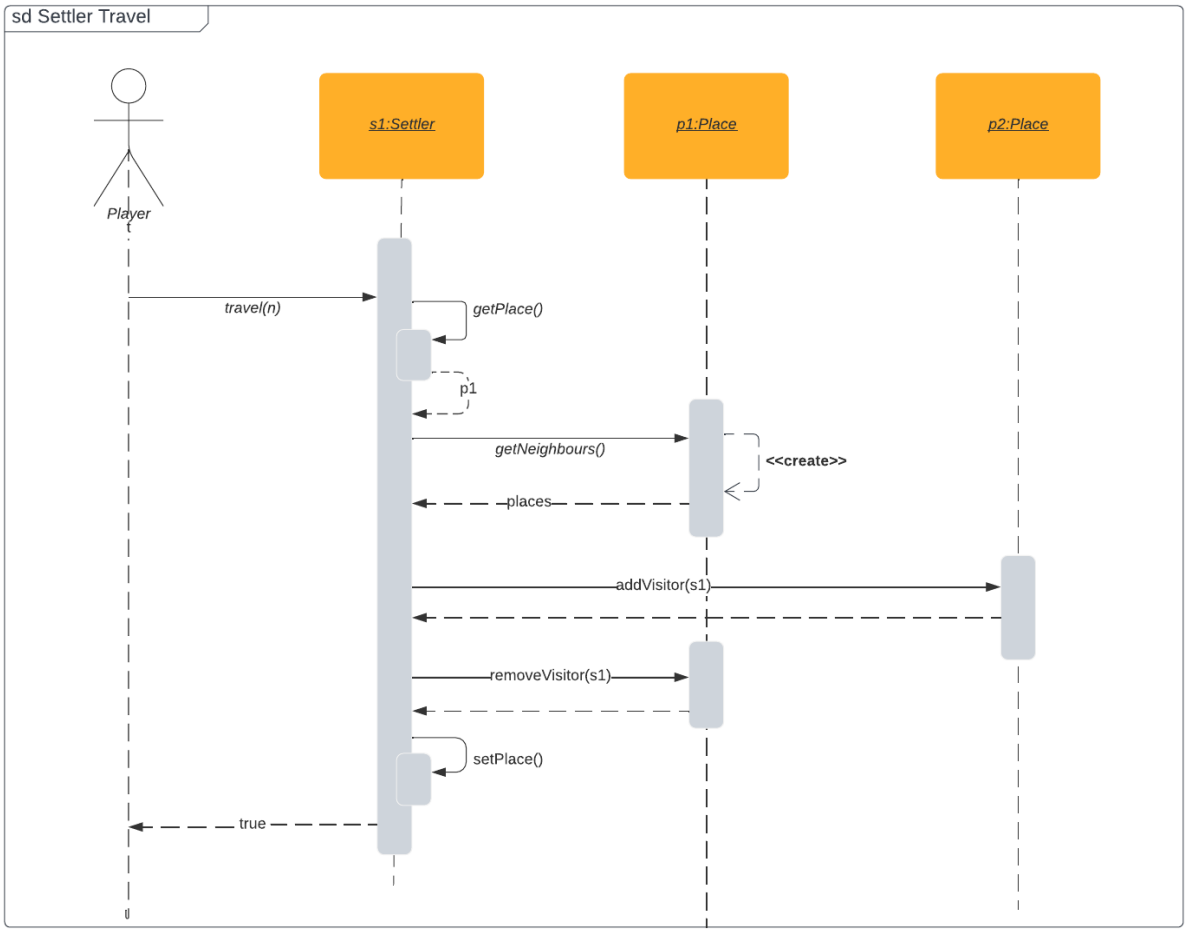
   System output: endGame()

5.3 Detailed sequence diagrams for internal activities

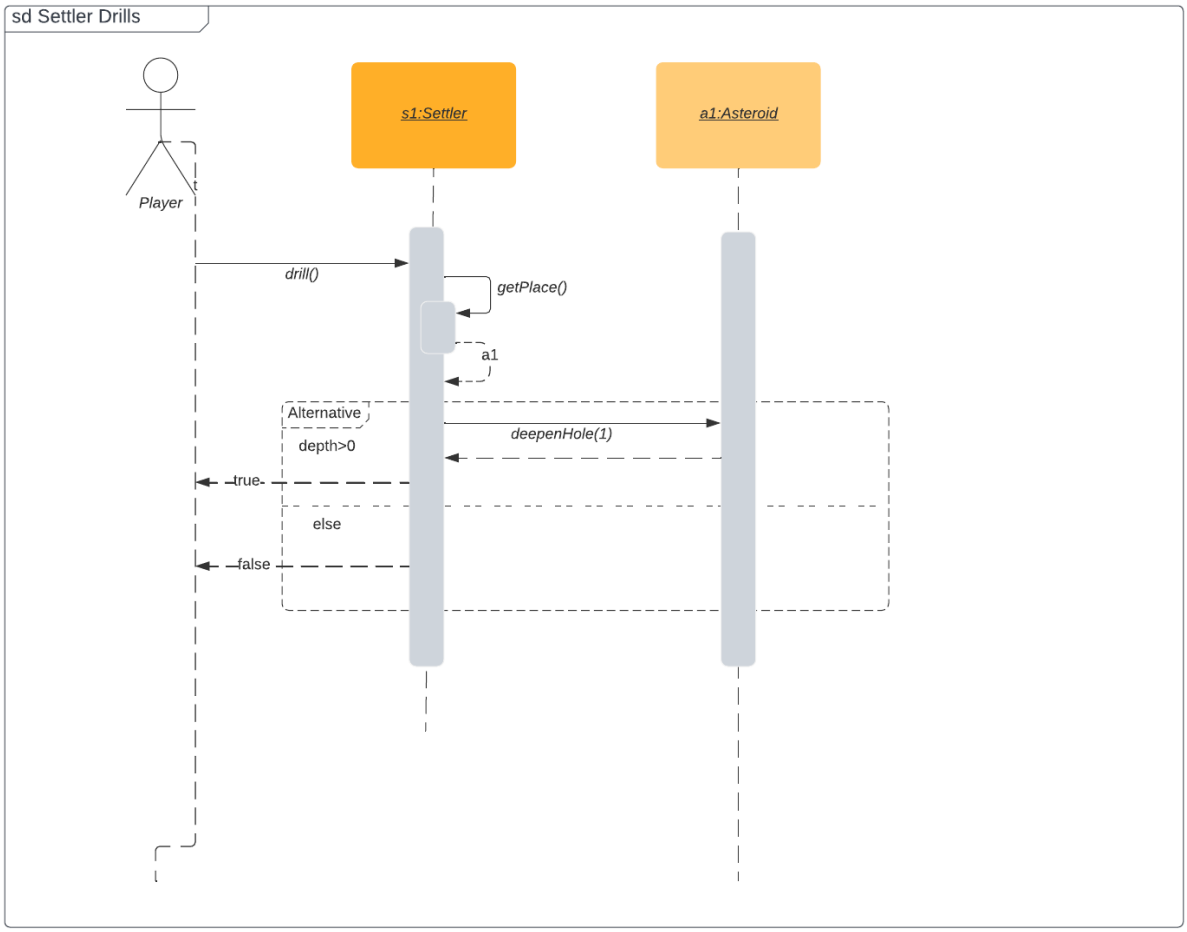
5.3.1 Initialization: Starting the game:



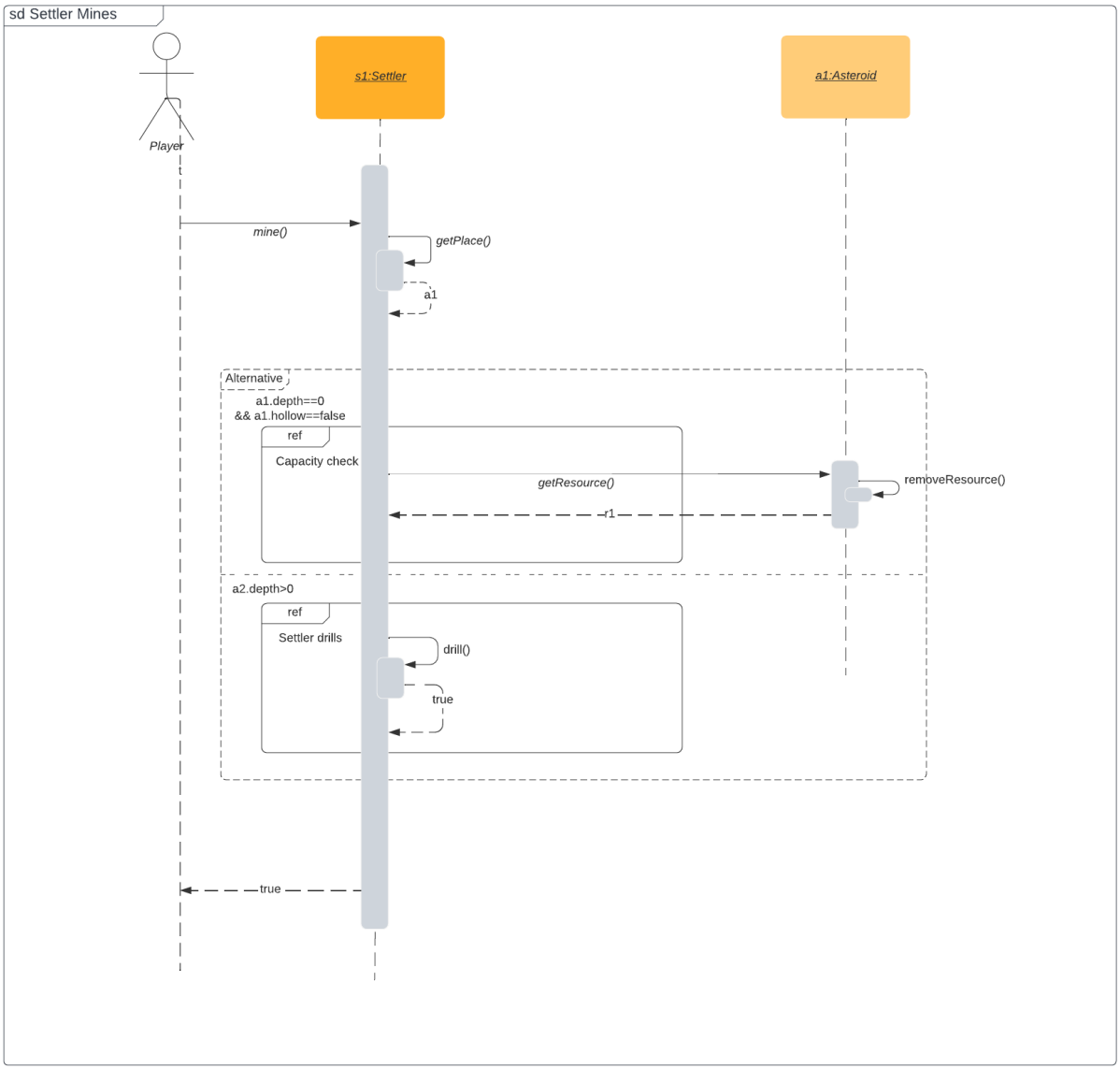
5.3.2 Settler Travels:



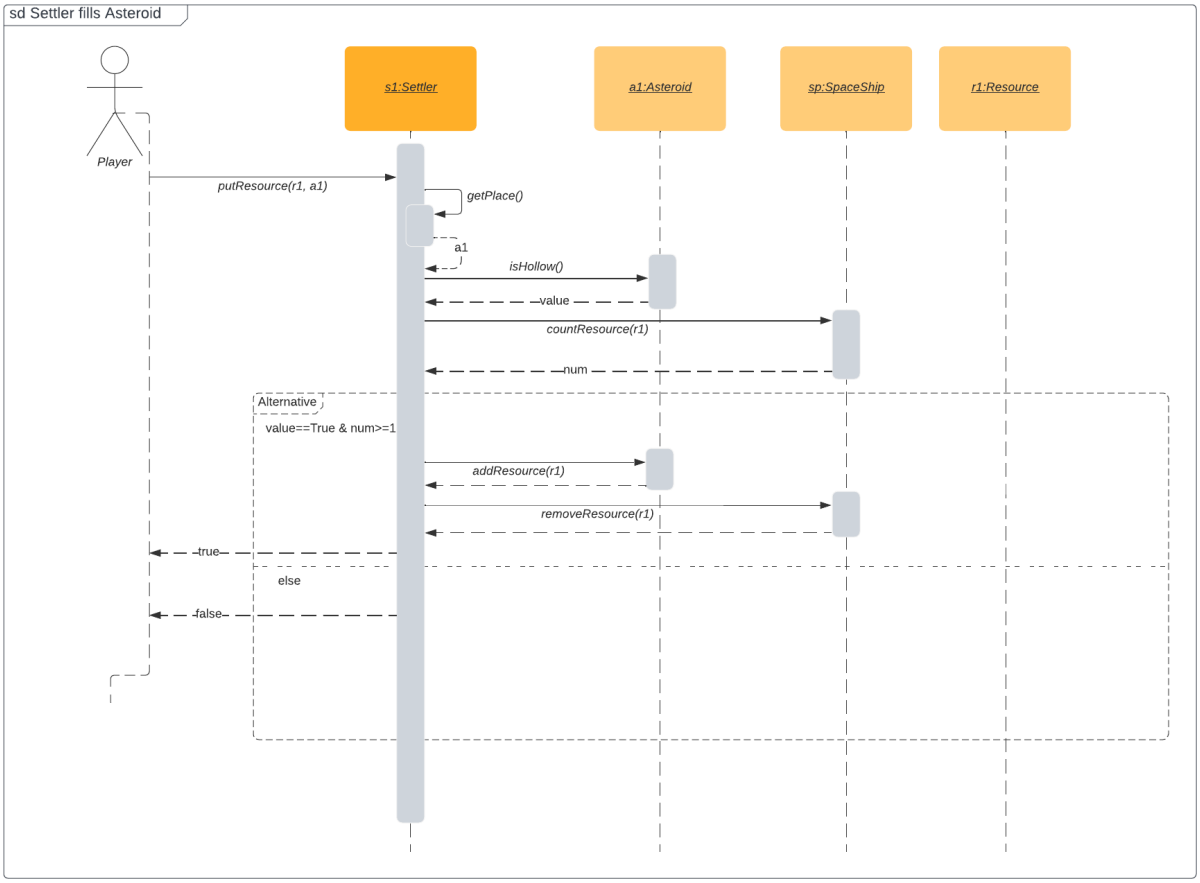
5.3.3 Settler Drills:



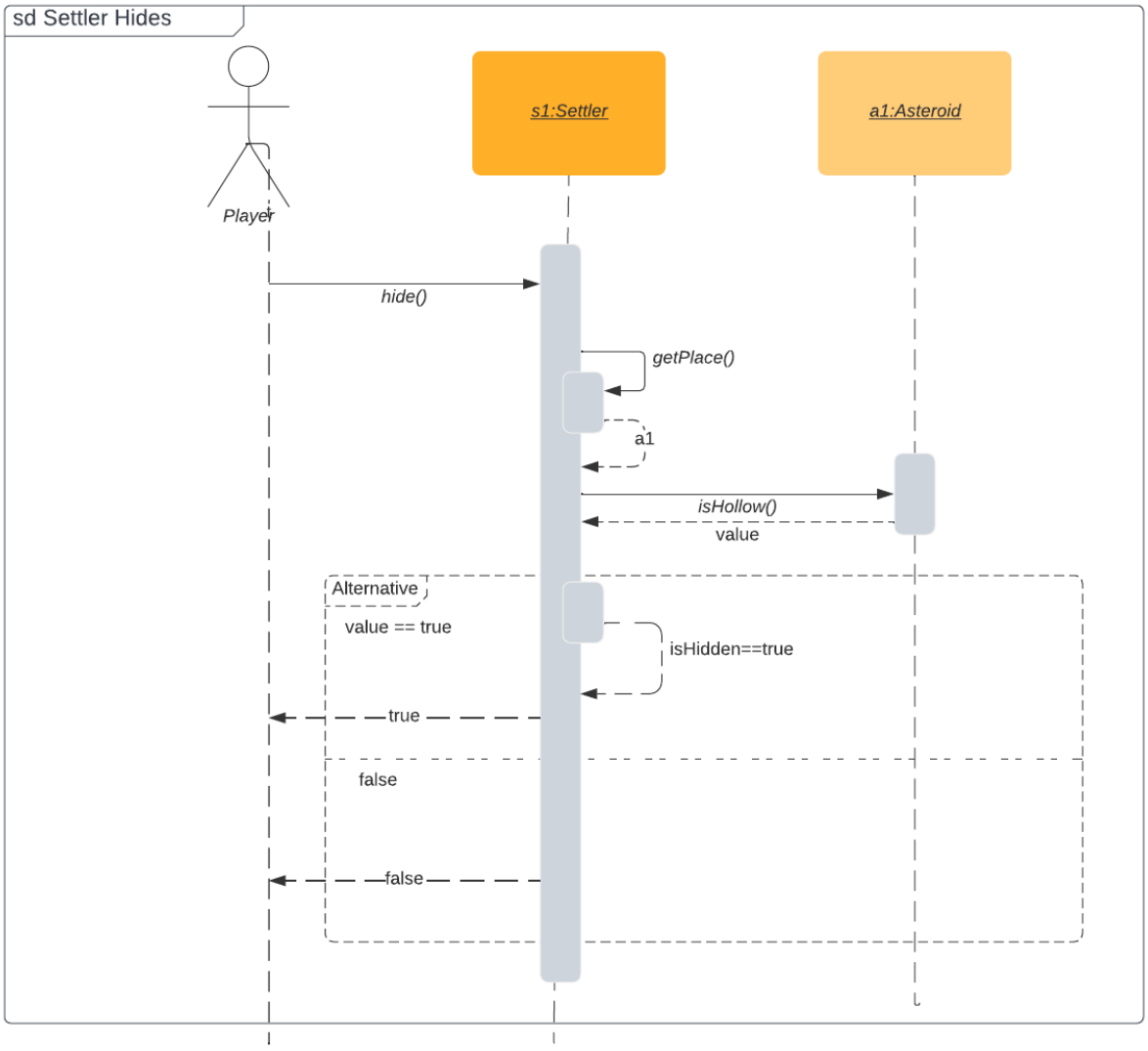
5.3.4 Settler Mines:



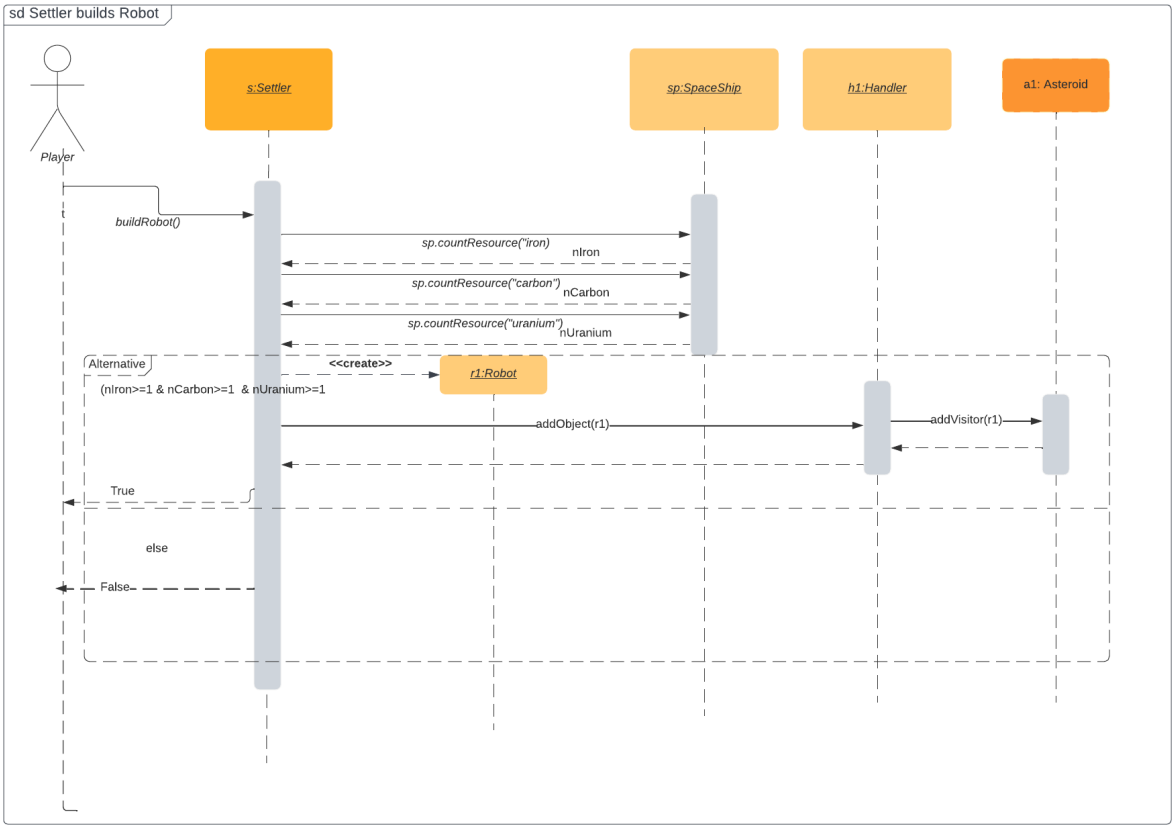
5.3.5 Settler Fills Asteroid:



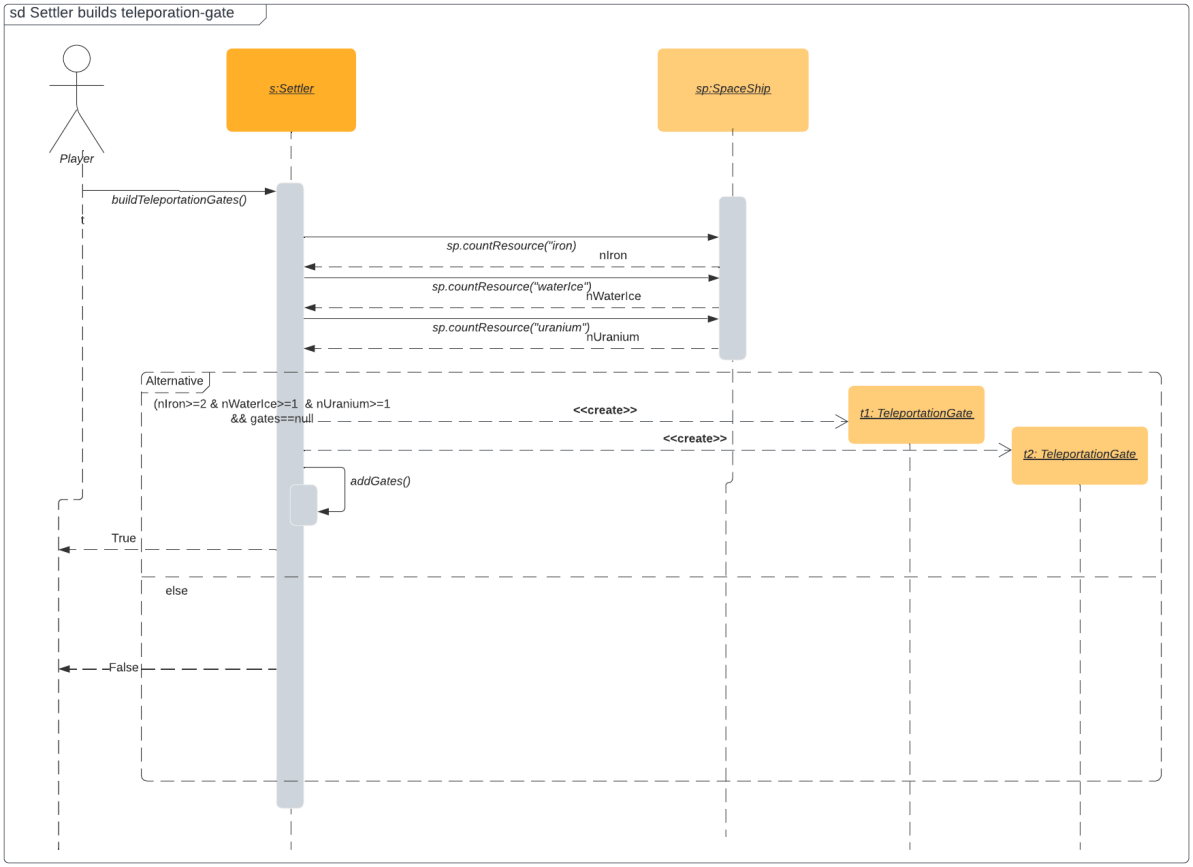
5.3.6 Settler hides in hollow asteroid:



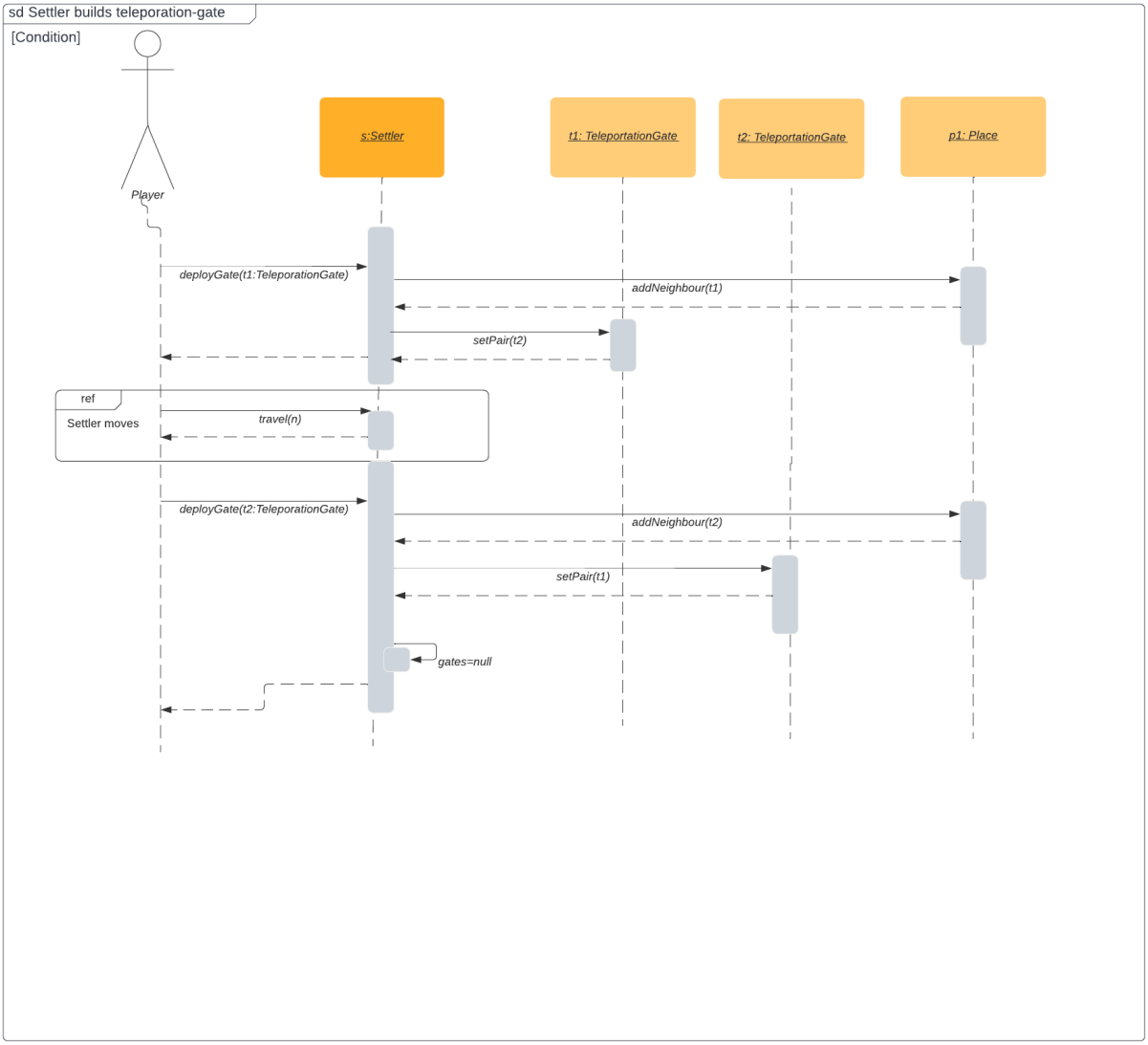
5.3.7 Settler builds Robot:



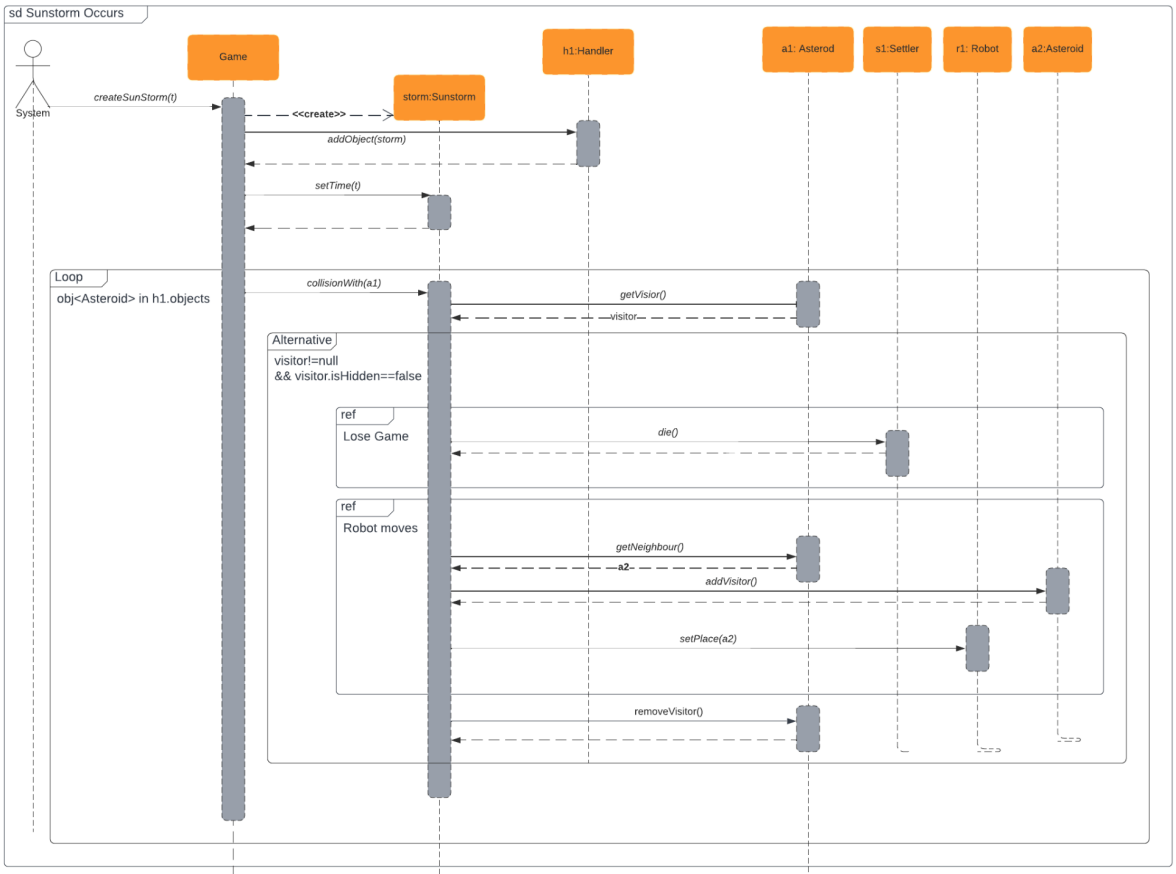
5.3.8 Settler builds Teleportation-Gate:



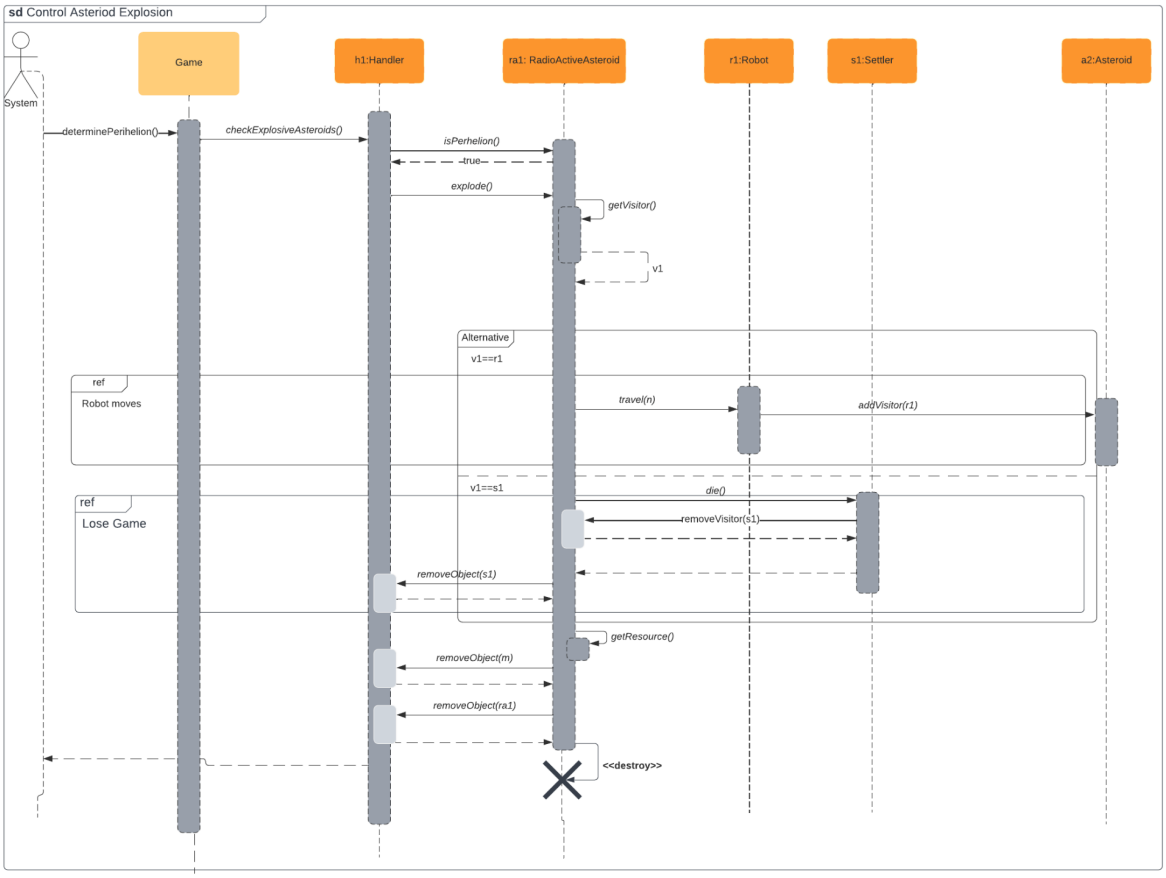
5.3.9 Settler deploys Gates:



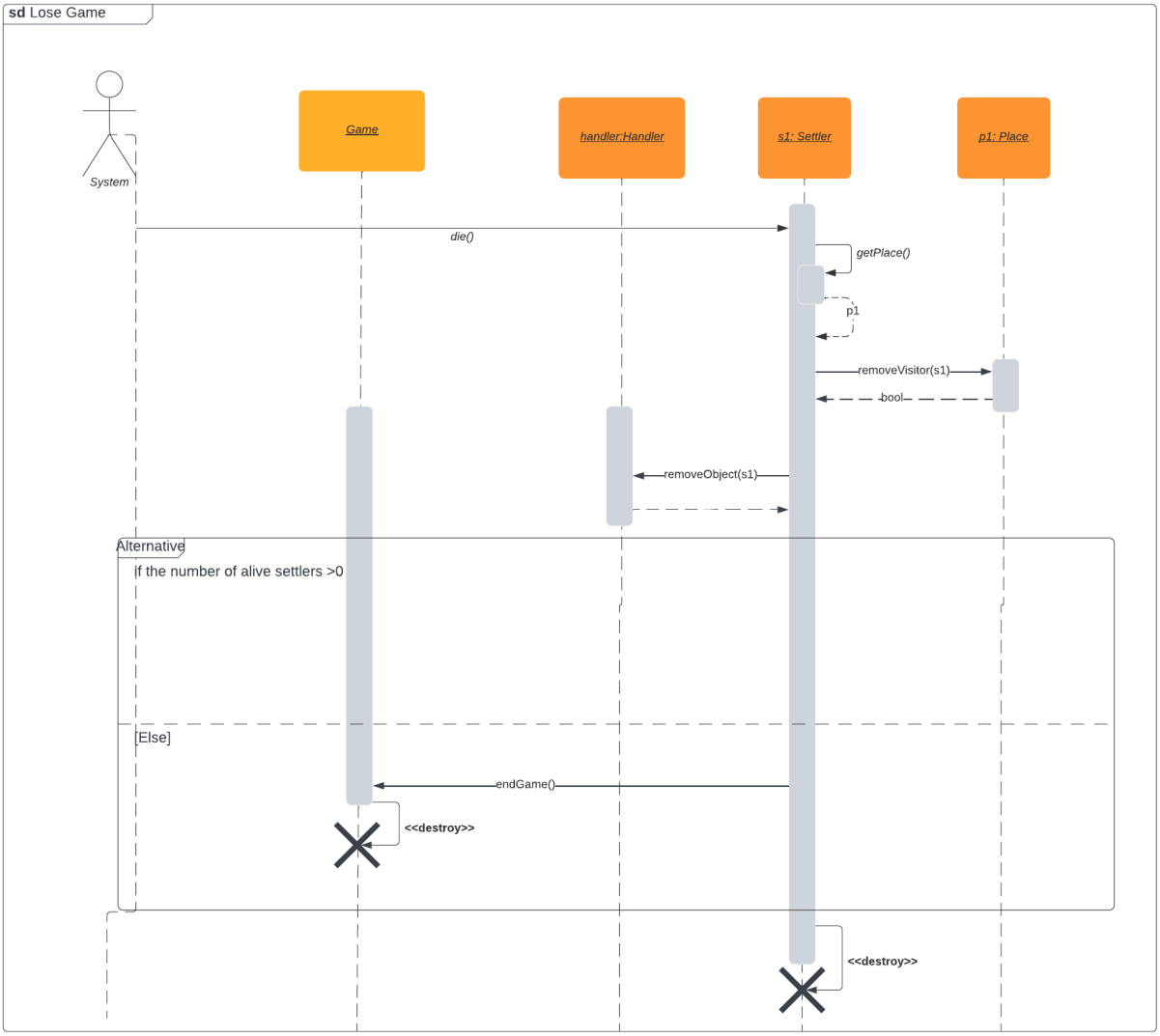
5.3.10 SunStorm occurs:



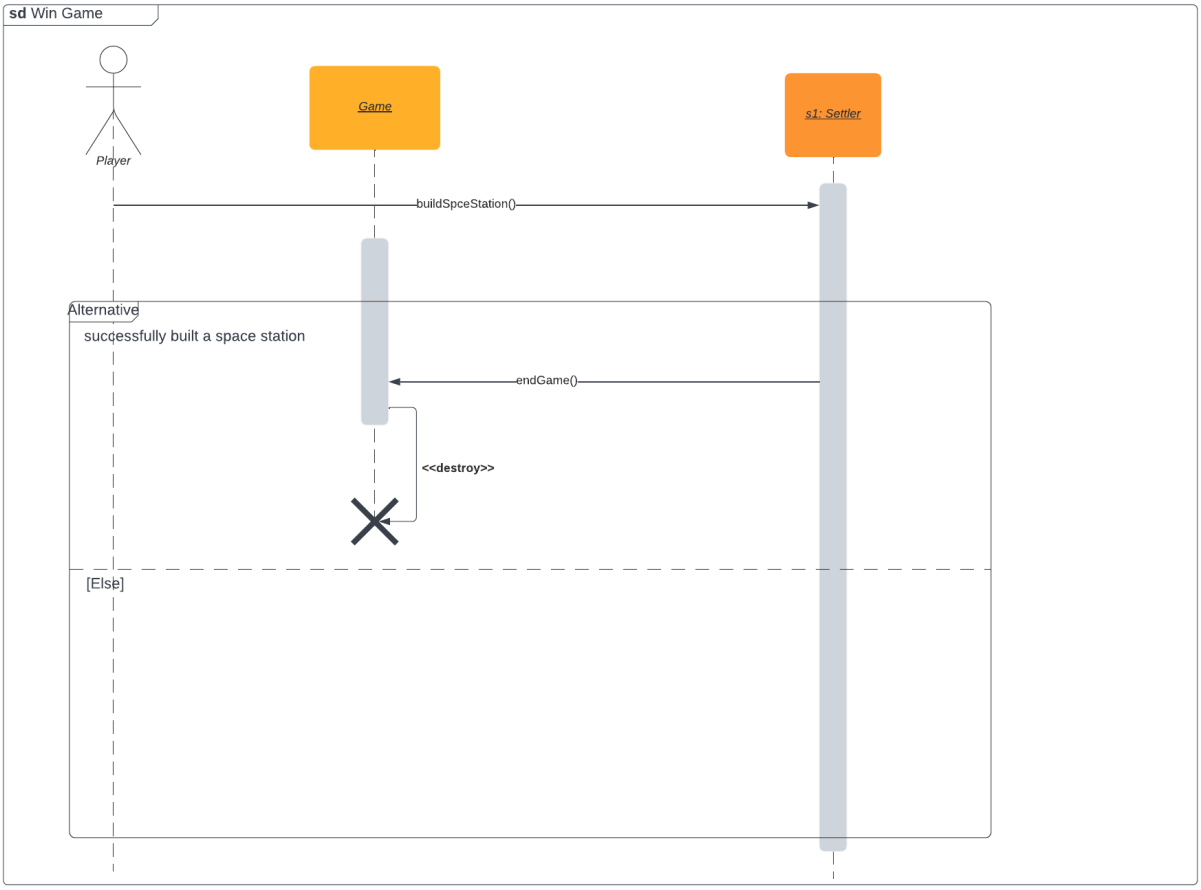
5.3.11 Asteroid Explosion:



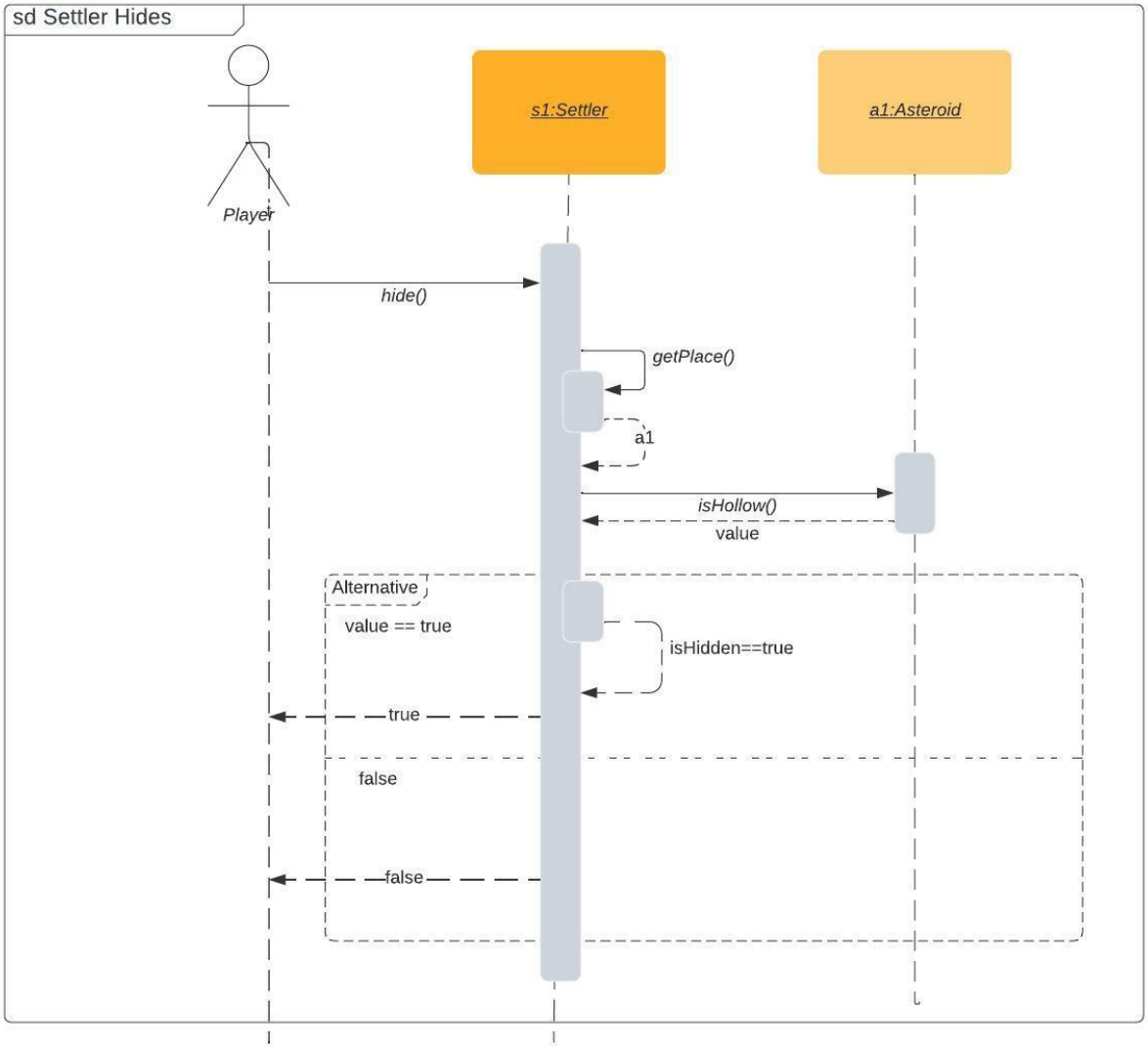
5.3.12 Losing the game:



5.3.13 Winning the game:

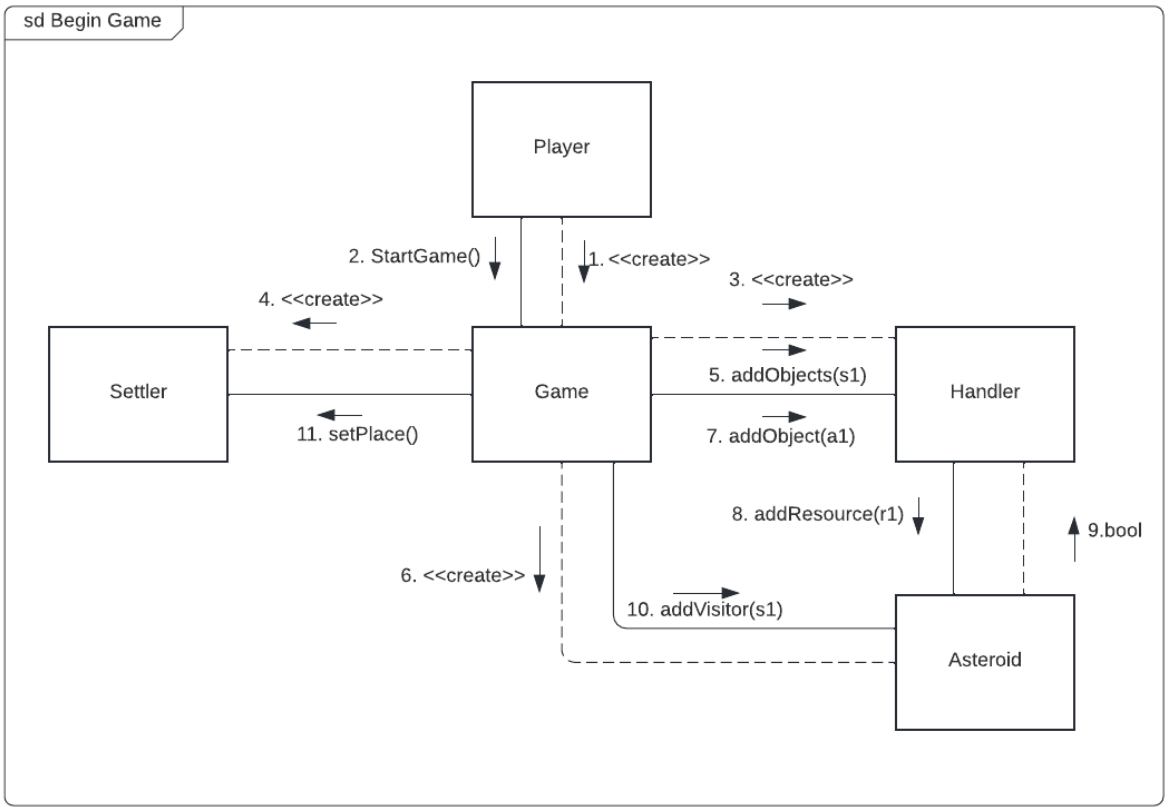


5.3.14 Settler Hides:

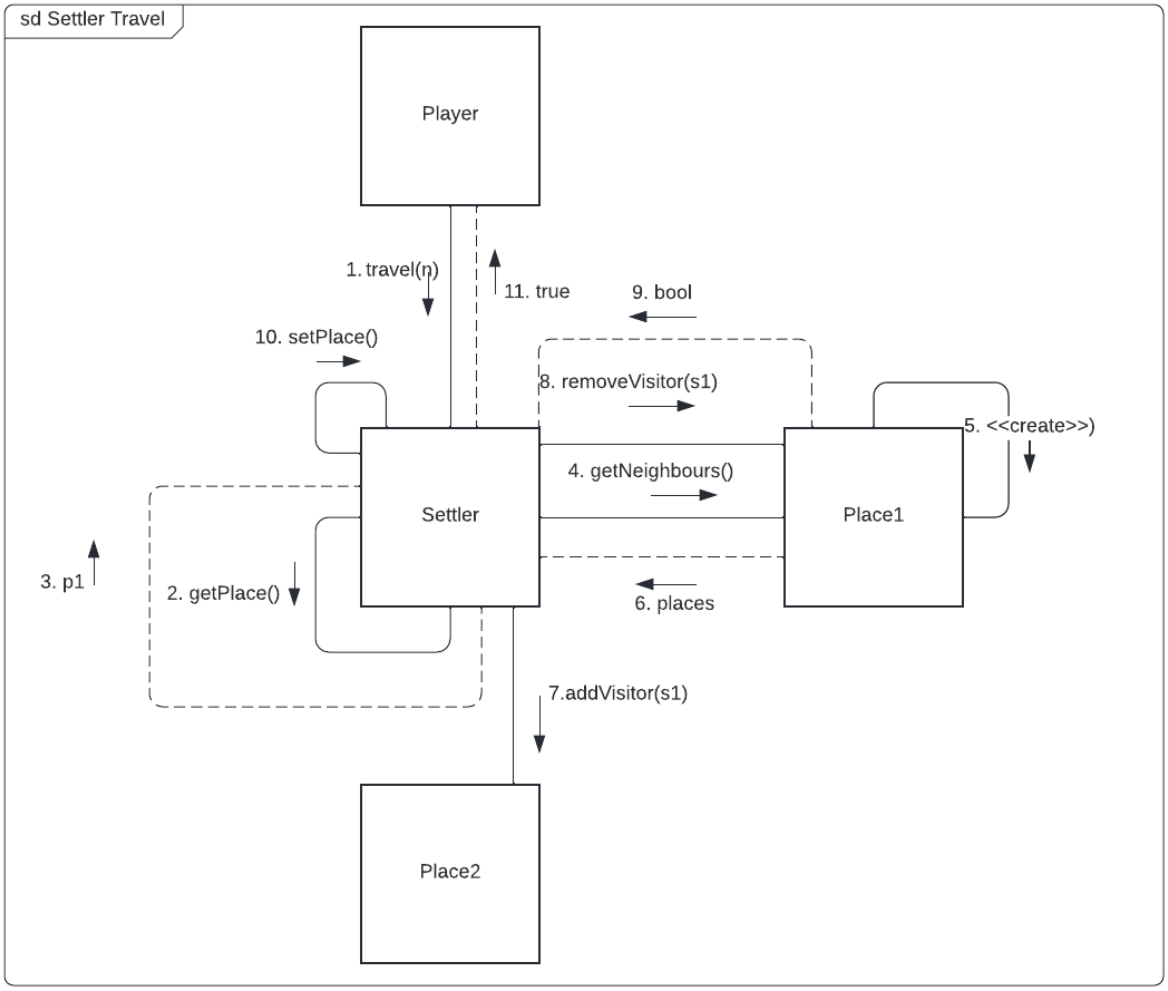


5.4 Communication diagrams

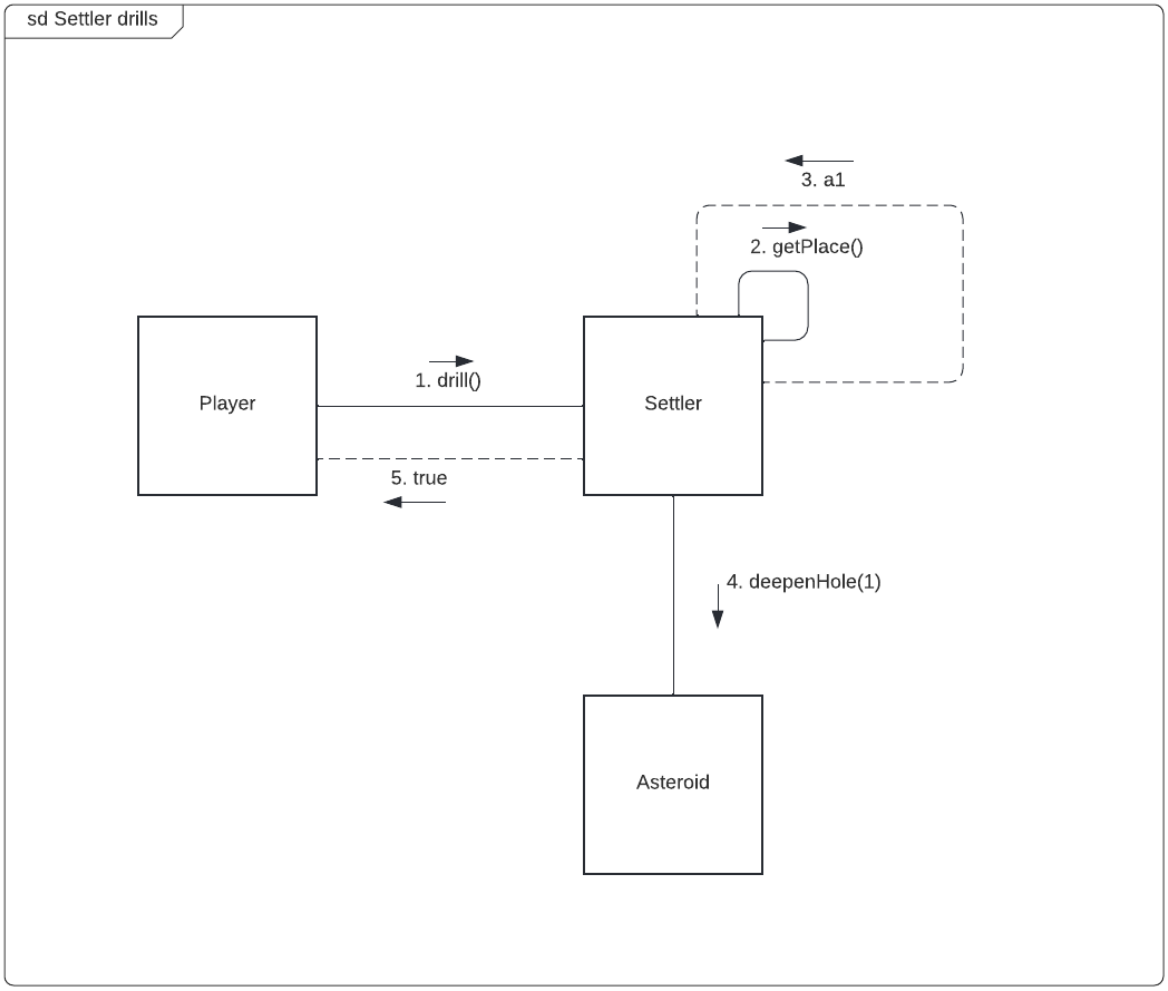
5.4.1 Initialization: Starting the game:



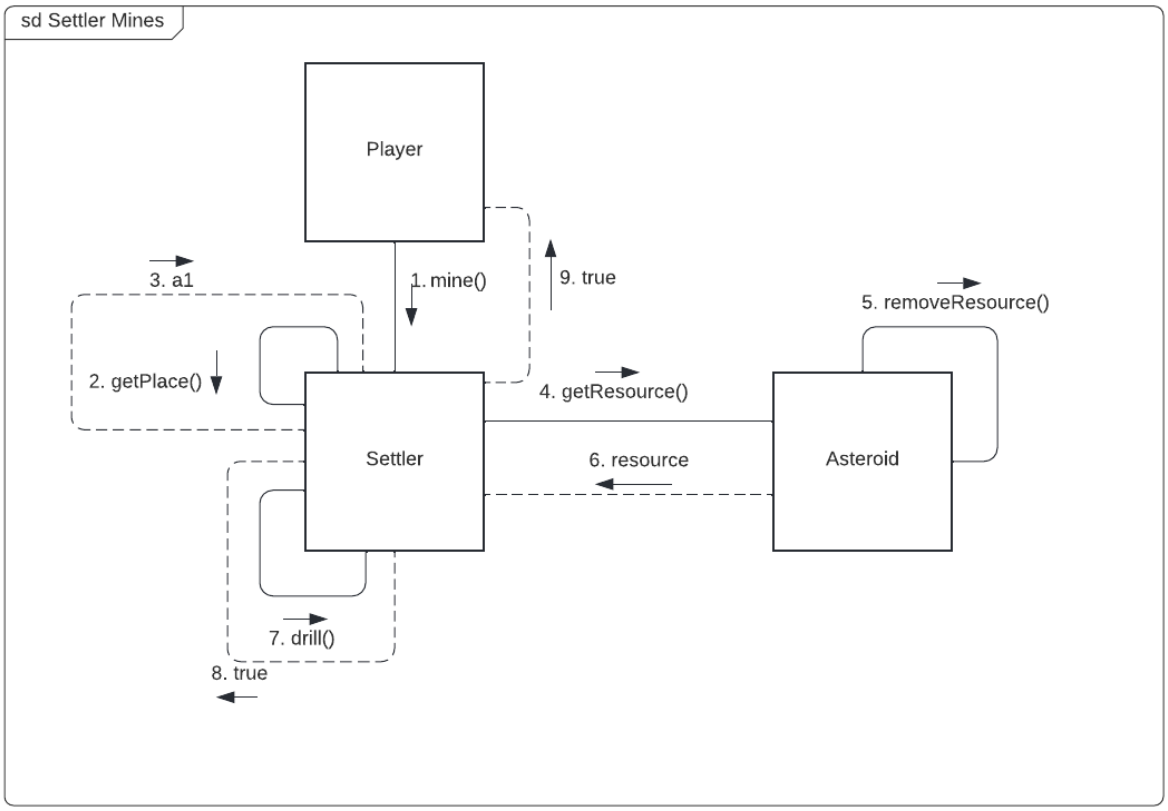
5.4.2 Settler travels:



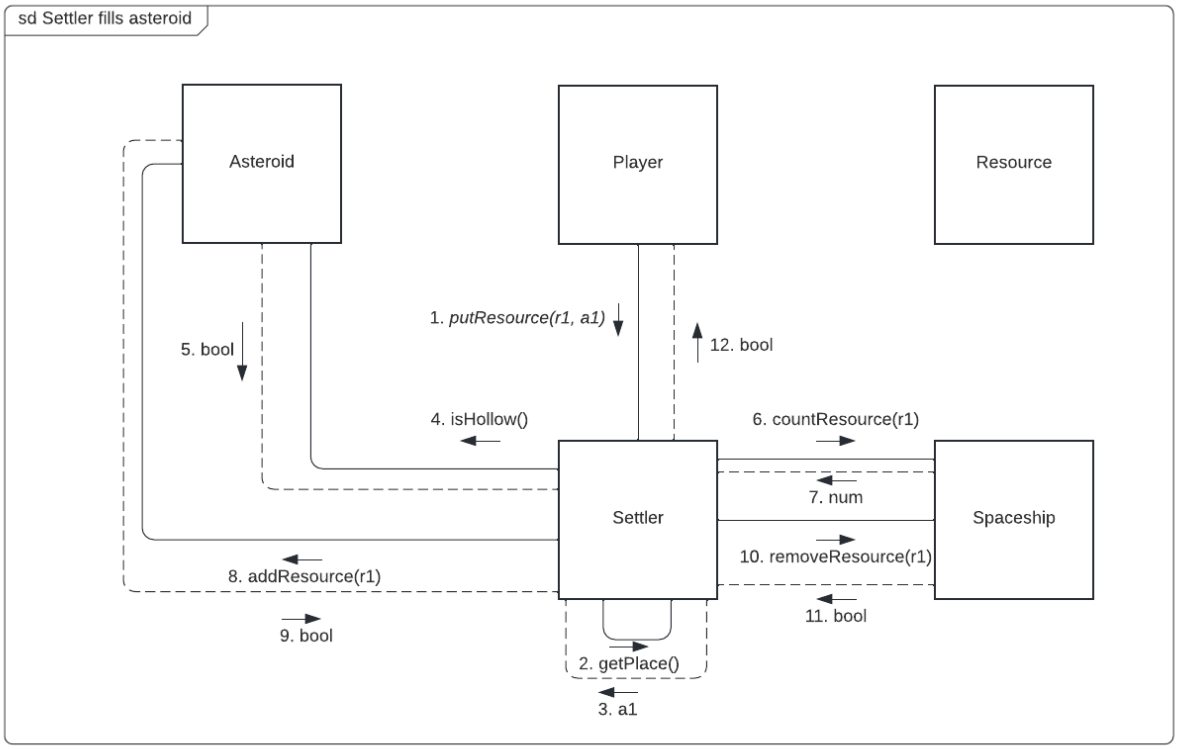
5.4.3 Settler drills:



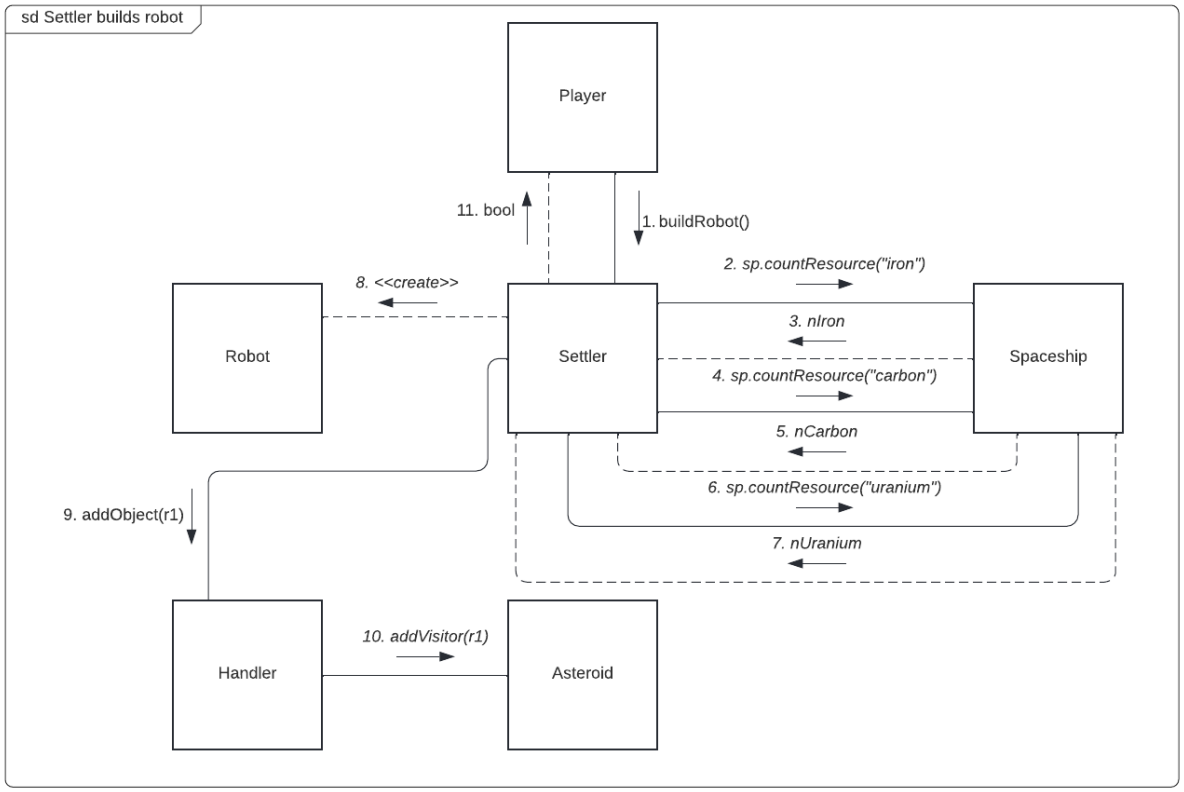
5.4.4 Settler mines:



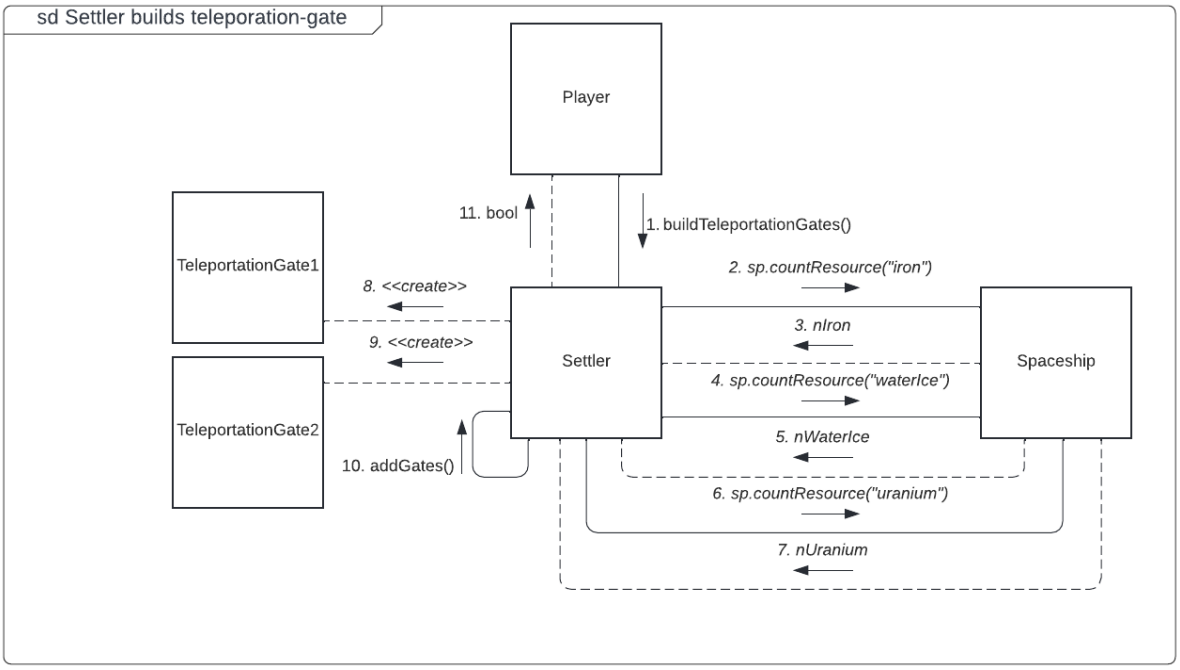
5.4.5 Settler fills Asteroid:



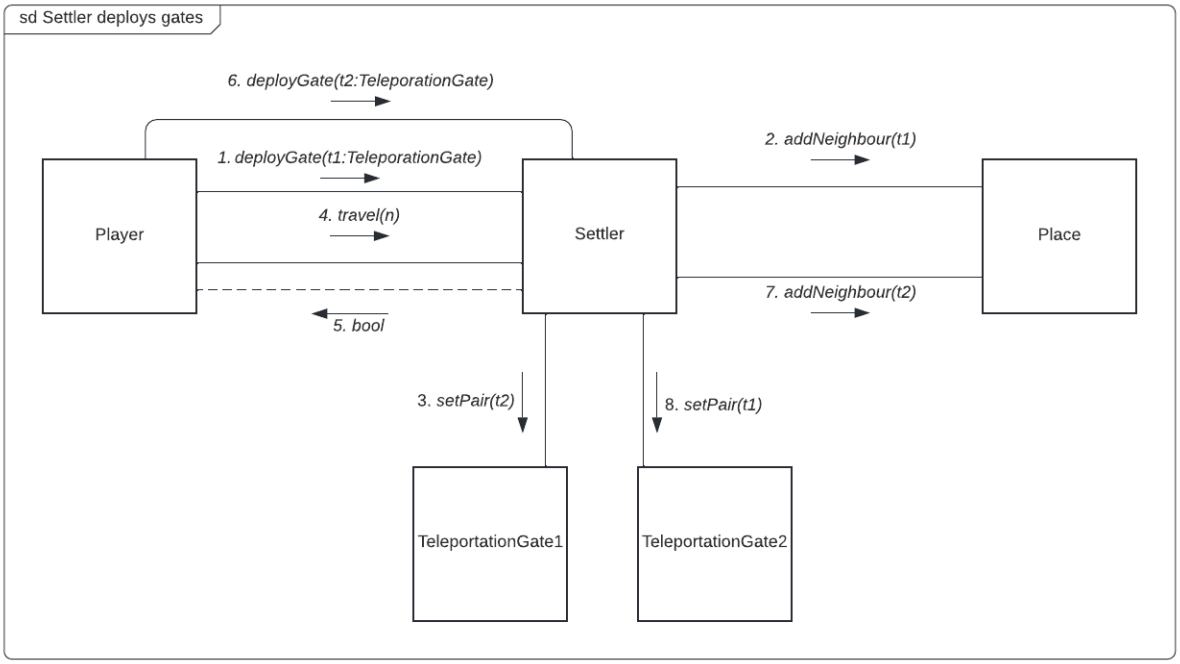
5.4.7 Settler builds Robot:



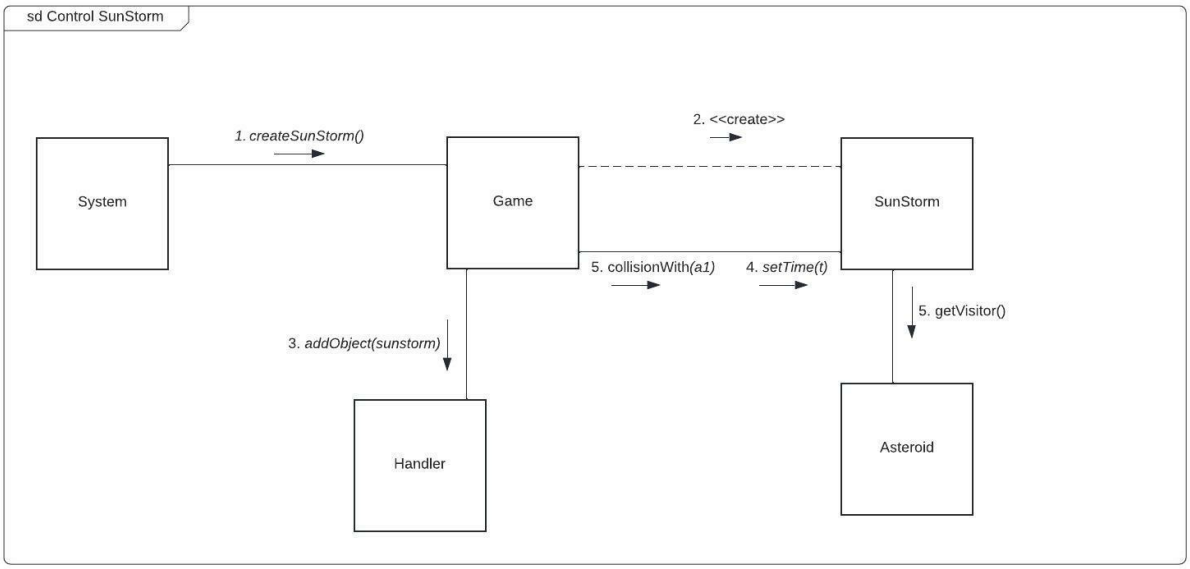
5.4.8 Settler builds Teleporation-Gate:



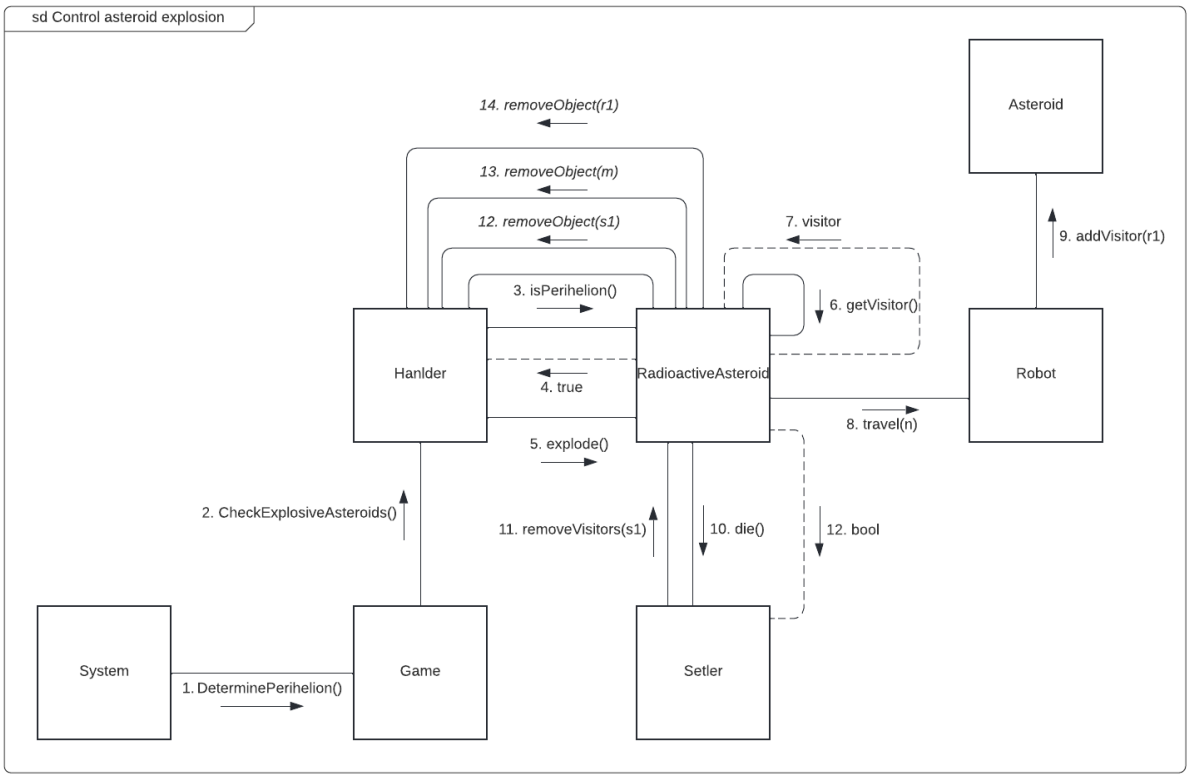
5.4.9 Settler deploys gates:



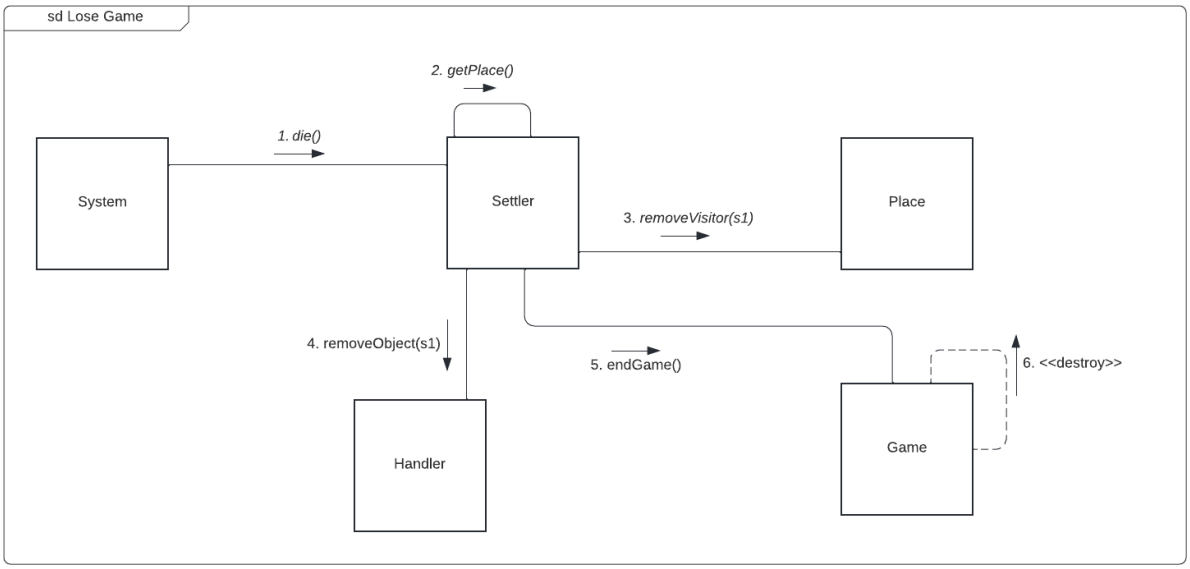
5.4.10 Sunstorm occurs:



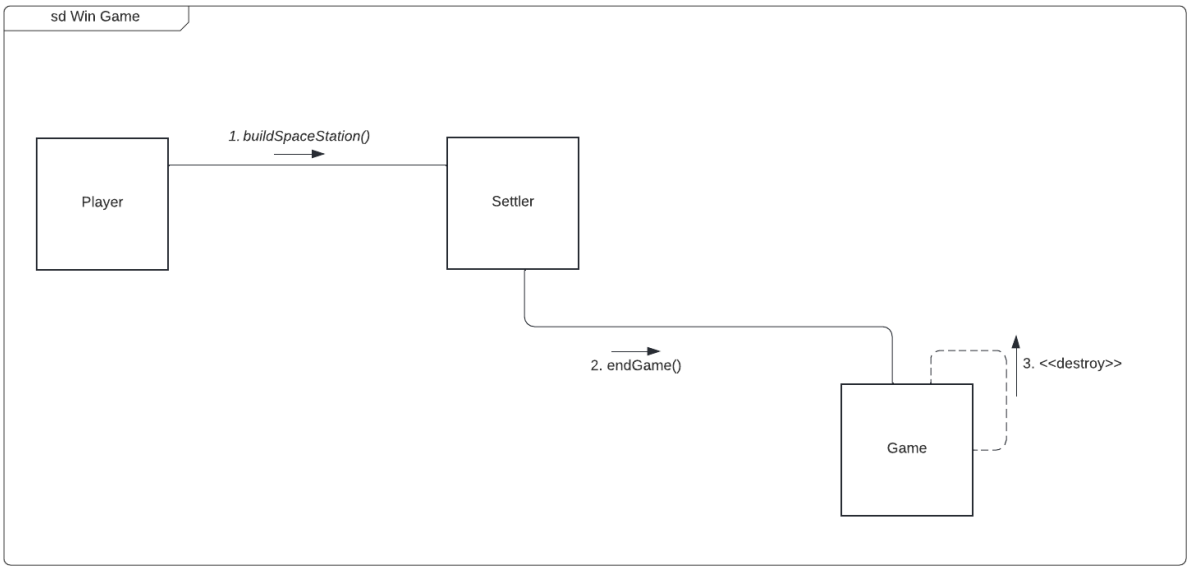
5.4.11 Asteroid explosion:



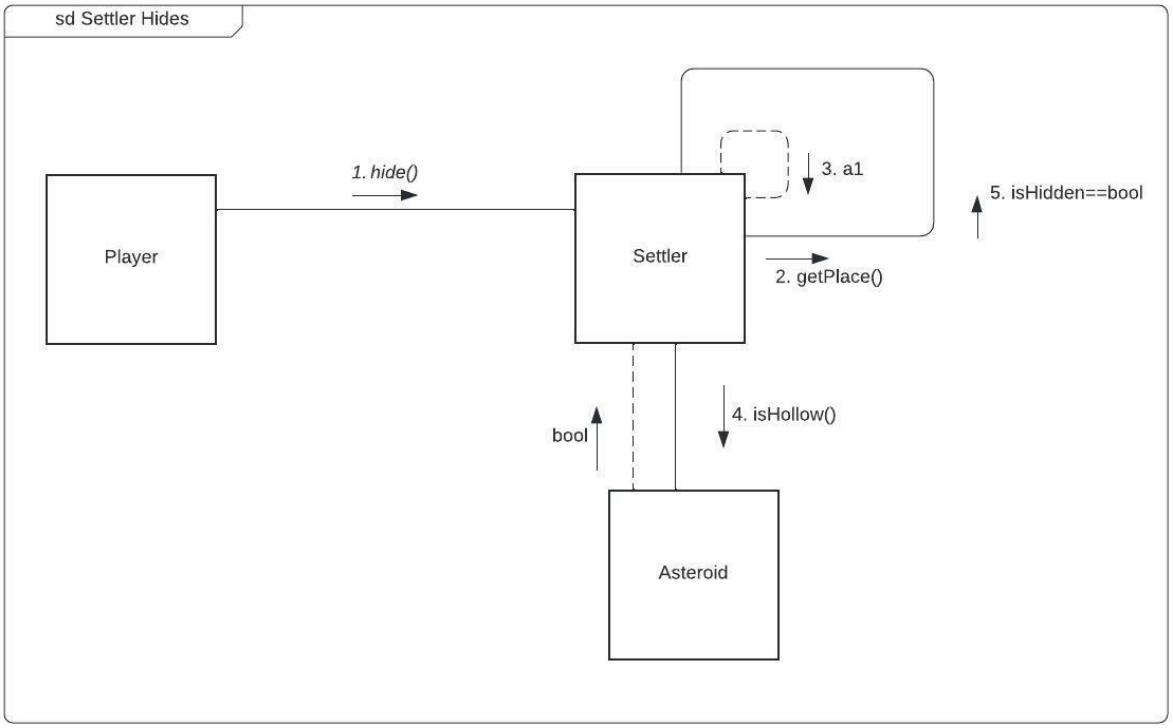
5.4.12 Losing the Game:



5.4.13 Winning the Game:



5.4.14 Settler Hides:



5.5 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
16/03/2022 22:00	*2 hours*	Whole team	*Starting work on the new version of the use-case diagram and the descriptions of its cases, based on the previous version from the first submission. Discussing how to approach the rest of the skeleton plan, and agreeing to learn more about communication diagrams individually, so we would be ready to work productively in the next meeting.*

17/03/2022	4 hours 40 mins	Kasay, Tushig, Neda	Planned and discussed strategy for this documentation. Had look through sequence diagrams and communication diagrams.
17/03/2022 22:00	1.5 hours	Chaitanya, Janibyek	Worked on the new version of the sequence diagram by using the old perfect diagram. Compressed and increased quality of diagrams considering new use-cases.
18/03/2022 *00:21*	3 hours	Tushig	Planned and implemented communication diagrams.
18/03/2022 14:00	1 hour	Kasay	Verifying sequence diagram and communication diagram
18/03/2022 22:00	2 hours	Desoki, Janibyek	Did some brainstorming about the UI and its new features. Worked on the Skeleton's UI, dialogs.
20/03/2022	2 hours 39 mins	Kasay	Verifying sequence diagram and communication diagram. Also suggested some UI catalogues regarding test cases.

6. Skeleton program

6.1 Deployment guide

6.1.1 List of files

File name	Size	Date	Content
Asteriod.java	1.62KB	2022.03.23	The class contains the GUI and major attributes of the class Asteroid like depth, hollow, distancefromSun etc, and some functions required in rendering the file, getting resources, deepen hole etc which covers most of the use case related to asteroid
Carbon.java	73 Bytes	2022.03.23	Carbon class is a child class of resource
Direction.java	101 Bytes	2022.03.24	The direction is an enum containing the movements up. Down, left right, which is used to handle the movement of the spaceship
Game.java	3.31 KB	2022.03.24	The game is one of our main classes, which is initiating the game by beginning the threads when the game begins and renders all objects like asteroids, spaceships etc

			which need to be displayed. It contains functions like render, tick and run which control the main structure of the game.
GameObject.java	1004 Bytes	2022.03.24	Besides having the normal getter,setter. This function is used to change the movements of the objects like asteroid and spaceship and check for collisions
Handler.java	984 Bytes	2022.03.24	Handler class is for getting neighbour place, adding and removing objects, getting settler, and checking if the asteroid is explosive.
ID.java	230 Bytes	2022.03.24	Enumeration of Settler, Robot, Asteroid, RadioActiveAsteroid, TeleportationGate, SunStorm, Iron, Carbon, Uranium, and WaterIce.
Iron.java	71 Bytes	2022.03.23	Iron class is a child class of resources.
Place.java	764 Bytes	2022.03.26	The place class is superclass of

RadioactiveAsteriod.java	244 Bytes	2022.03.24	RadioActiveAsteroid is a superclass of Asteroid with explode method.
Resources.java	145 Bytes	2022.03.24	Class defines the type of resources.
Settler.java	1013 Bytes	2022.03.26	The settler class is the GUI of the settler and will involve the movements and basic rendering.
Robot.java	353 Bytes	2022.03.24	The robot class is the graphical user interface of the robot which involves the movements, basic rendering, and it can get damage.
Spaceship.java	239 Bytes	2022.03.24	This class is responsible for initializing spaceship with its capacity and current inventory.
Sunstorm.java	320 Bytes	2022.03.24	SunStorm class is GUI implementation sunstorm itself.
Teleportationgate.java	520 Bytes	2022.03.24	The TeleportationGate class is the graphical user interface of the gates.

Uranium.java	74 Bytes	2022.03.23	Uranium class is a child class of resource
Visitor.java	979 Bytes	2022.03.26	Visitor class extends the game object and deals with the what things can be done when a settler visit the asteroid and similar functions like drill, hide etc.
WaterIce.java	110 Bytes	2022.03.23	WaterIce class is a child class of resource. It differs with sublime() method from other resources.

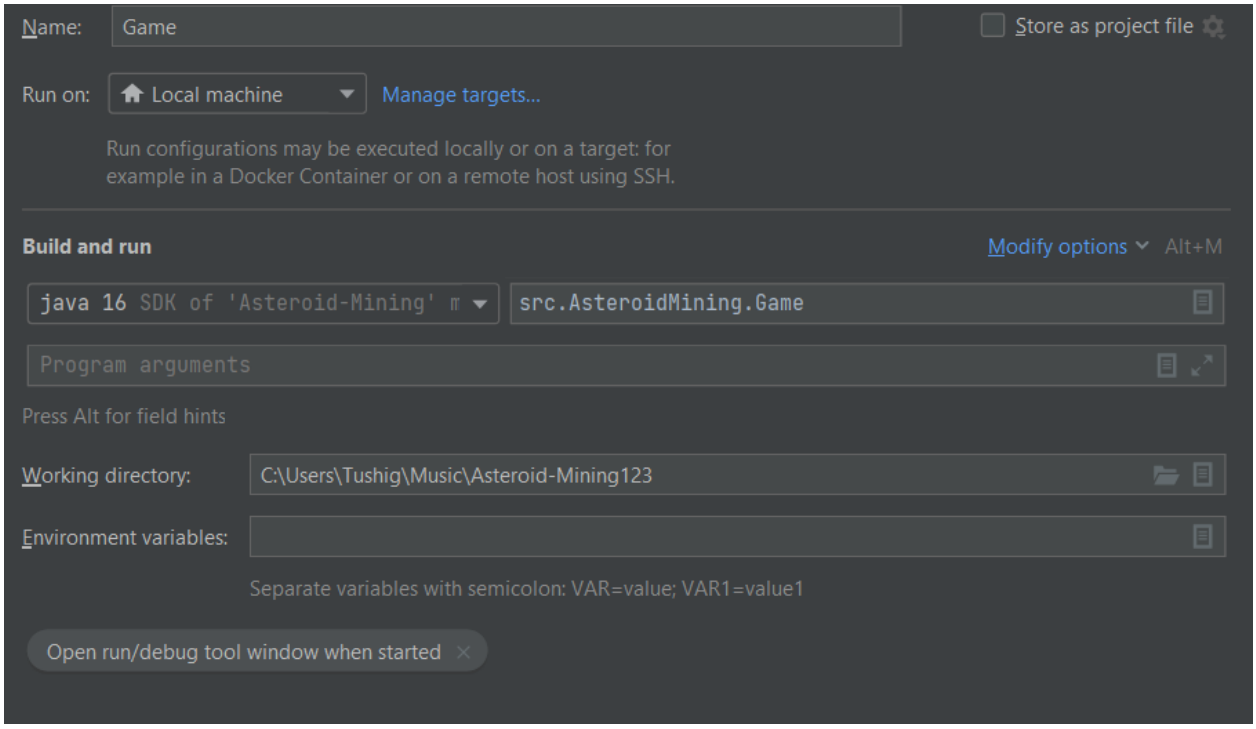
6.1.2 Compilation

In order to do the successful compilation of the code, one can clone the project or download the zip file and run it in Intellij IDE.
To execute the program we need to run the main function present in the Game.java file. Namely, the entry point is this function.
The link to github repository:
https://github.com/jackdotb/Asteroid-Mining

6.1.3 Run

There are no particular requirements except the fact that jdk should be installed in the users IDE and should preferably use intellij IDE. In order to compile the project successfully, “Game” class is needed to configured in “Run/Debug Configuration” section of the IntelliJ IDE. Especially, some of test cases are suggested that they should be checked in debugging mode with breakpoints.

Run/Debug Configuration:



6.2 Evaluation

Name of the team member	Participation (%)
Janibyek Bolatkhan	16.66%
Kasay Ito	16.66%
Chaitanya Arora	16.66%

Tushig Bat-Erdene	16.66%
Neda Radonjic	16.66%
Abdelrahman Desoki	16.66%

7. Concept of prototype

7.1 Interface definition of Prototype

7.1.1 General description

Our game program will have two choices of the input command to control the game by the player. First one and default one is from the Keyboard handler(Virtual keys) and according to the taken input key value, the application will perform corresponding actions and the game will be played through. Moreover, the second one is more likely depending on the testing side. Game application runs the whole game scenario through text file context where specific commands are already written. This type of input command usually will be used for testing the game.

7.1.2 Input language

<Mouse Click> Command 1: Start Game

Description: In the main menu, the user can choose the option of starting the game when the “Mouse_Click” is pressed.

Options: Starting the game.

<Virtual Key/Mouse Click> Command 2: Exit Game

Description: In the main menu, the user can choose the option of exiting the game when the associated key is pressed.

Options:

1. Exiting the game by pressing “Mouse_Click”.
2. Exiting the game by pressing “ESC_Key”.

<Virtual Key> Command 3: Settler traveling

Description: When the user presses the key associated with this command, the settler will travel through space.
Options: The settler can either travel between neighboring asteroids or through a teleportation gate by following input keys:

1. Settler goes up by pressing “UP_Key”.
2. Settler goes down by pressing “DOWN_Key”.
3. Settler goes right by pressing “RIGHT_Key”.
4. Settler goes left by pressing “LEFT_Key”.

<Virtual Key> Command 4: Drill

Description: When the user presses “D_Key” with this command, the settler will drill the core of an asteroid.

<Virtual Key> Command 5: Mine

Description: When the user presses the “M_Key” with this command, the settler will mine the resources from an asteroid.

Options:

1. Using “Mine” command, Settler can mine “Carbon”.
2. Using “Mine” command, Settler can mine “Water Ice”.
3. Using “Mine” command, Settler can mine “Uranium”.
4. Using “Mine” command, Settler can mine “Iron”.

<Virtual Key> Command6:Hide

Description: When the user presses the “H_Key” with this command, the settler will hide in the core of a hollow asteroid.

Options:

<Virtual Key> Command7: Build Robot

Description: When the user presses the “R_Key” with this command, the settler will build an AI robot.
Options: The user can choose which asteroid to build the robot on.

<Virtual Key> Command: Build Teleportation Gate

Description: When the user presses the “T_Key” with this command, the settler will build a teleportation gate.

Options:

<Virtual Key> Command: Fill Asteroid

Description: When the user presses the “F_Key” with this command, the settler will fill an asteroid with a unit of resource.

Options: The user can choose which of the available resources the asteroid will be filled with.

<Virtual Key> Command: Deploy Teleportation Gate

Description: When the user presses the “G_Key” with this command, the settler will build a teleportation gate.

Options: The user can choose where the gate will be deployed.

1. Deploying the first gate.
2. Deploying the second gate.

<Virtual Key> Command: Build Space Station

Description: When the user presses the “S_Key” with this command, the settler will build a space station.

<Virtual Key/Mouse Click> Command: Check Inventory

Description: When the user presses the “TAB_Key” with this command, the settler will check the inventory of the spaceship.

<Virtual Key/Mouse Click> Command: Land on the Asteroid

Description: When the user presses the “L_Key” with this command, the settler will land the spaceship in the near asteroid.

7.1.3 Output language

Game Starts:

Description: If the player starts the game, all the major elements in the game are initialized.

Game is exited:

Description: The game is terminated when the player executes the exit command of the game.

Settler travels: When the travel command is given, the settler can move in different directions

Description: The position of the settler changes, and it is assigned to a new Place as a Visitor.

Settler drills: When the drill command is given, the settler drills the core of an asteroid.

Description: The depth of the asteroid's rock mantle is decremented by 1.

Settler hides:

Description: When the hide command is executed, Settler can hide inside of the hollow asteroid.

Settler mines:

Description: When the hide command is executed, Settler can gather resources from a drilled asteroid.

1. Using "Mine" command, Settler can gather "Carbon".
2. Using "Mine" command, Settler can gather "Water Ice".
3. Using "Mine" command, Settler can gather "Uranium".
4. Using "Mine" command, Settler can gather "Iron".

Settler builds a robot:

Description: Robot will be initialized after counting sufficient resources (Iron, Carbon, and Uranium).

Settler builds a teleportation gate:

Description: Teleportation gate will be initialized after counting sufficient resources (Iron, Carbon, and Uranium).

Settler fills an asteroid:

Description: When the fill command is executed, settlers can fill hollow asteroids with resources.

Settler deploys a gate:

Description: When a user pressed the deploy command, Settler can deploy the first or second teleportation gate.

Settler builds a space station:

Description: When the build space station command is executed, Settler can build a space station if the settler has sufficient resources. Players will win the game if the space station is successfully built.

Settler checks inventory:

Description: When the check command is executed, inventory status will be demonstrated.

Settler lands the spaceship on the Asteroid:

Description: When the Land command is executed, the player controls settlers to land its spaceship to Asteroid. It is more demonstrated as attaching the spaceship object to the moving asteroid to be able to dig and mine as well as hiding**.**

Create sunstorm:

Description: The system generates random sunstorm flashes that can hit the asteroids, damaging the visitors that are not hidden.

Check explosive asteroids:

Description: System checks asteroids when they are in the perihelion if the asteroid is explosive or not. Normally the fully drilled asteroid which contains uranium is called a RadioActive asteroid and calls its method **Explode()** inside the class.

7.2 Real use-cases

Use-case name	Start Game

Short textual description	Initializing the game
Actors	Tester
Dialog, scenario	The game can be started when “start game” option is selected from the menu. All the objects on the Asteroid belt, such Asteroids, Resources, Settlers are created.

Use-case name	Travel
Short textual description	Player controls the settler to move and change its position in the asteroid belt.
Actors	Player/User
Dialog, scenario	1. The settler can travel with his spaceship from one asteroid to another asteroid. 2. But also settler can wander in the asteroid belt and pass through the teleportation gates.

Use-case name	Drill
Short textual description	Player controls the settler to drill the Asteroid

Actors	Tester
Dialog, scenario	1. Settler drills the asteroid. This action is executed by the user command input. 2. If settler fully frills the radio-active asteroid when it is in perihelion, it explodes and kills the settler

Use-case name	Hide
Short textual description	Player controls the settler to hide in the asteroid
Actors	Player
Dialog, scenario	1. Settler hides in the fully drilled hollow asteroid. 2. Hiding action is usually executed to escape from the sunstorm

Use-case name	Mine
Short textual description	Player control the settler to mine the asteroid.
Actors	Player

Dialog, scenario	1. Settler can only mine the resource from the fully drilled asteroid. 2. Settler can only mine the resource from the fully drilled asteroid. 3. Settler only can mine when it is in the aphelion.

Use-case name	Fill asteroid
Short textual description	Player controls the Setter to fill the asteroid with a unit of resource
Actors	Player
Dialog, scenario	1.Settler can only fill the asteroid if: - the asteroid is hollow - Settler can select the specific resource from its inventory. - Settler has a available resource in its spaceship's inventory

Use-case name	Build Robot

Short textual description	Player controls the settler to build the Autonomous robot for the further help of drilling the asteroid
Actors	Player
Dialog, scenario	1. Settler only can build the robot if it has enough resources. 2. If the settler does not have enough resources, the system shows a message on the screen regarding the missing resources. 3. Setter can choose one of the neighboring asteroids to place the newly created robot. 4. Newly built robot is controlled by the system autonomously.

Use-case name	Build Teleportation Gates
Short textual description	Player controls the settler to build the teleportation gate in the asteroid belt.
Actors	Player
Dialog, scenario	1. Settler only can build the teleportation gates if settler has enough resources. 1. If the settler does not have enough resources, the system shows a message on the screen regarding the missing resources. 2. Settler only can carry two gates at a time.

Use-case name	Deploy Gate
Short textual description	Player controls the settler to deploy the teleportation gate.
Actors	Player
Dialog, scenario	1. Gates are only deployed successfully if - Settler has available teleportation gate(normally settler only can carry two gates) 2. Settler deploys one gate at a time. 3. Newly deployed gates are connected with its pairs and ready for the visitors to pass through.

Use-case name	Check Inventory
Short textual description	Player checks the current inventory of settler
Actors	Player
Dialog, scenario	Player performs this action to see the current collected resources and their amount. And also the teleportation gates that the settler is carrying.

Use-case name	Land on the Asteroid

Short textual description	Player controls settler to land its spaceship to the asteroid
Actors	Player
Dialog, scenario	Player performs this action to be able to reach the asteroid and do his/her work and mission.

Use-case name	Build SpaceStation
Short textual description	Player controls the settler to build the Space Station.
Actors	Player
Dialog, scenario	1. Building Space station is the main goal of this game. 2. Settler only can build a space station if it has enough necessary resources. 3. Building the SpaceStation puts the final point of the game. And Settler wins the game.

7.3 Test plan

Name of the test-case	Showing the GUI
Goal	Interface of the game showed properly

Short description	Make sure that the GUI of the game has displayed properly on the user screen.

Name of the test-case	Starting the game D
Goal	Making sure that the game starts appropriately
Short description	Test case made to check whether the game starts properly and the user can press the buttons and start playing or not.

Name of the test-case	Build the space station ND
Goal	Testing if the space station can be built
Short description	Test case made to check if the user presses the key, is the space station created or not.

Name of the test-case	Settler traveling D
Goal	Testing if the settler can travel

Short description	Test case for checking whether the settler can travel in all directions including the asteroid belt, checking whether the settler can travel after pressing keys.

Name of the test-case	Settler Drills D
Goal	Checking what changes after the settler drills.
Short description	Test cases for parameters will be changed after the settler drills the asteroid, the depth of the hole and whether the mantle has been drilled or not.

Name of the test-case	Settler Mine D
Goal	Checking what changes after the settler Mine.
Short description	Test cases for parameters will be changed after the settler mines the asteroid, the depth of the hole, layer of mining (at aphelion) and whether the mantle has been drilled or not.

Name of the test-case	Filling Asteroid D

Goal	Check the capacity of the asteroid
Short description	Testing the amount of resources after settlers filling the asteroid and checking the depth of the hollow.

Name of the test-case	Destination of settler from Prehilion D
Goal	Check if the settler is at perihelion layer or not
Short description	Checking the place of the settler and is settled at perihelion layer or out of it.

Name of the test-case	Building the robot D
Goal	Check if the settler can build the robot
Short description	This test case was made to check whether the settlers can build an uncontrollable robot, and the amount of resources decreased after building it.

Name of the test-case	Building Teleportation gates D

Goal	Check if the settler can build the teleportation gates
Short description	This test case was made to check whether the settlers can build teleportation gates, and the amount of resources decreased after building it.

Name of the test-case	Deploying the gate D
Goal	Check if the settler can Deploy the teleportation gates
Short description	This test case was made to make sure that settlers deploy the gate , and the location of the gate will be changed with its own neighbors.

Name of the test-case	Hiding settler D
Goal	Object of settler existing or not
Short description	This test case was made to make sure if the settler is able to do any operation except becoming visible again.

Name of the test-case	Death of the settler D

Goal	Checking the health of the settler after explosion
Short description	Testing whether the settler will die if an explosion happened or not.

Name of the test-case	Decreasing in settler health
Goal	Checking the health of the settler after sunstorm
Short description	Testing whether the settler will die if a sunstorm happened or not while the settler is visible.

Name of the test-case	Visible sunstorm ()
Goal	Checking whether sunstorm is visible or not
Short description	Test case made to check the location of the settler, whether the settler is hollow or not.

Name of the test-case	Carrying Resources

Goal	Check if settler can carry resources or not
Short description	Test case was made to check whether it's safe or not to carry resources.

Name of the test-case	Checking inventory D
Goal	Gui should show the settler the amount of each resource
Short description	Test case made to check the number of resources in each state of the game.

Name of the test-case	End game
Goal	Checking whether the settler wins or loses
Short description	Test case made to check whether settlers can end the game by either building a space station or losing the game damage(explosion-sunstorm).

7.4 Support programs for testing

At the current phase of the game program, we have developed a well working skeleton program which makes users able to already test the game by selecting the actions from the console

menu and see its printed output values, such as methods and response texts as well user interface questions.

However, for the future features and complete version of the application, we are planning to make the program be able to be tested through both JUnit testing and text file. Text file contains all the input commands, and it is the expected output.

7.5 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
30/3/2022 18:30-20:30	2.5 hours	Tushig and Neda	Defining the input/output language.
30/3/2022 18:18 to 20:30	2 and half hours	Kasay and Desoki	Writing the 19 test cases with some detailed instructions.
30/3/2022 18:00 to 20:30	2 and half hours	Janibyek and Chaitanya	Completed 7.2 Real Use Cases artifact.s

8. Detailed plans

8.1 Design level plan of classes

8.1.1 Game

Responsibility

Only one GameSystem object can exist in a game. This class has the responsibility to start and end the game. These include condition checks, creating major entities and events.

Attributes

- **private Handler handler:** This is an object of the class Asteroid belt this will; be used to create an asteroid belt in the game
- **protected Settler settler:** This is an object of the class settler which will later be used in the diagram.
- **private ArrayList<Robot> robots:** Game system must know all robots since they are controlled by the system.
- **private SunStorm sunStorm:** Sun storm object which is used when a sun storm occurs.

Methods

- **public void StartGame():** This function is used to initialize all major elements in the game.
- **public void EndGame()**: this function is majorly invoked when the conditions which the settler can die become true.
- **public void createSunStorm(int time):** this function can be invoked randomly and creates an occurrence of a sunstorm in the game
- **public void determinePosition():** Determines if an asteroid is at perihelion or aphelion, based on the attribute distance from Sun. This is important for water ice asteroids and radioactive asteroids.
- **public bool checkWin():** Checks if the needed resources have been collected on an asteroid, and if so, the game is won.
- **public bool checkLose():** Checks if the settler has been killed by an explosion, and if so, the game is lost.

8.1.2 Handler

Responsibility

Handler is a container that contains all necessary instances related to the played game. A settler, asteroids, robots and teleportation gates are contained. Furthermore, this has the responsibility for checking whether the asteroid is explosive or not.

Attributes

- **public ArrayList<GameObject> objects:** all the list of the objects that currently exist on the asteroid belt, ranging from visitor to asteroid.

Methods

- **public void addObject(GameObject obj):** Adds corresponding object to the **objects** list
- **public void removeObject(GameObject obj):** removes corresponding object from the list.
- **public void checkExplosiveAsteroids():** calls explode() if the radioactive asteroid is on the perihelion.

8.1.3 Game Object

Responsibilities

GameObject is responsible for dealing with existing objects in the game. During the extension or requirement changes, we will add the functions which are commonly used among derived classes. Many classes inherit their unique ID from the Game Object class, since it is their superclass.

Attributes

- **protected string id:** stores the unique ID of objects in the game

Methods

- **public string getID():** gets the unique ID of an object

8.1.4. Visitor

Responsibility

The Visitor class is a parent class of Settler and Robot classes, since they share many attributes and features. It stores information about the asteroid the visitor is currently on (place), whether the visitor is hidden in a hollow asteroid, and if it is alive at all. It contains methods responsible for traveling, drilling, mining, and getting and setting the place of each visitor.

Attributes

- **private bool alive:** specifies if the settler is alive or not at the moment, to ensure extendability, in case of requirements change (e.g. multiple settlers), this condition will be later changed according to the events and things that take place.
- **private bool hidden**: indicates if the visitor is hidden in the core of a hollow asteroid.It is an important attribute to understand the state of the settler as it can have limitations based on them.
- **private Place place:** stores information about the place of the visitor(the asteroid that it is on)

Methods

- **public bool travel**- The settler attempts to travel to a neighboring asteroid. If the asteroid really is a neighbor and the operation is successful, the truth is returned.
- **public bool drill**- The settler drills through the mantle of an asteroid.
- **public Place getPlace -** Gets the current position of a visitor.
- **public void setPlace**- Sets the position of a visitor.
- **public bool hide** - Checks if a visitor is hidden in a hollow asteroid. If not, the action of hiding a visitor is performed, and a true is returned.
- **public bool die -** If the health of a robot reaches zero, it is killed, with no effect to the rest of the game. If a settler is not hidden in an explosion, it is killed, and the game ends.

Superclasses

- **GameObject**

8.1.5 Settler

Responsibility

The settler class inherits many attributes and methods from the visitor class. The Settler class has a few additional attributes and methods, since a Settler has some different capabilities and responsibilities compared to the other type of visitors-robots. This class stores information about the teleportation gates and spaceship belonging to a specific settler. It contains methods exclusive to settlers - such as mining, building robots and teleportation gates, storing resources and deploying gates.

Attributes

- **private List<TeleportationGate> gates:** Stores information about any teleportation gates that the settler has built and deployed. Normally, settler only carries two gates.
- **private Spaceship spaceship:** This class is used as a inventory of the settler that stores all the goods of the settler, such ranging from mined materials to teleportation gates.
- **private Robot robot:** newly created autonomous robot of the settler. This variable helps to access the robot and get signal from the robot in terms of drilling asteroid

Methods

- **public bool mine()** - If the mantle of the asteroid has been drilled through, the settler now mines the asteroid for useful resources.
- **public bool buildRobot()** - We check if there are enough resources to build a robot. If so, a robot is instantiated, and true is returned.
- **public bool buildTeleportationGates()** - We check if there are enough resources to build a teleportation gate. If so, a gate is instantiated, and a true is returned.
- **public void fillAsteroid**- The settler can fill a hollow asteroid with one unit of a resource.
- **public void deployGate(TeleportationGate g):** deploys the corresponding available teleportation gate.
- **public void checkInventory():** shows all the current goods in the spaceship's inventory.

Superclasses

- **Visitor->GameObject**

8.1.6 Robot

Responsibility

This class is responsible for managing robots, which are a type of visitors. It stores information about the health of the robot, which is a metric indicating if a robot has been affected by dangers. It also contains a method that marks a robot as damaged, decreasing its health if it is affected by a danger.

Attributes

- **private int health:** the robot survives sun storms with damage only, so this is a metric of the damage made to the robot during a storm.

Methods

- **public void getDamage(int n):** In the case of a sun storm, a robot that was not hidden inside a hollow asteroid will be marked as damaged.
- **public int getHealth():** returns the current health of the robot.

Superclasses

- **Visitor->GameObject**

8.1.7 Spaceship

Responsibility

The responsibility of this class is to store attributes regarding inventory, capacity, the resources stored in the spaceship, and the settler that the spaceship belongs to. The methods deal with inventory- adding resources, removing them, and counting them.

Attributes

- **private int currentInventory:** stores the number of units of resources that the settler has stored in the spaceship already
- **HashMap resources:** describes the resources that the settler currently has. The key of the map is the type of the resource, and the value is the amount of the resource.

- **private int capacity:** Indicates the capacity of the spaceship when it comes to storing resources.

Methods

- **public bool addResource (Resource r):** Checks if there is enough space for an additional unit of resource, and if so, adds it to the spaceship, and returns true.
- **public bool removeResource(Resource r):** Removes a unit of resource from a spaceship and returns true.
- **public int countResource(string r):** Counts the number of units of resource in a spaceship and returns an integer value.
- **public boolean checkCapacity():** checks the capacity of the inventory and returns boolean.

8.1.8 Resource

Responsibility

This class is a parent class to four different resources- uranium, iron, carbon and water ice. It contains the attribute type, that indicates the name/type, and a method to access that name and have it returned as a string.

Attributes

- **protected string type:** Stores the name of the resource name/type.

Methods

- **public string getType:** Gets the name of the individual resource and returns it as a string.

8.1.9. Uranium

Responsibility

The Uranium class is a child of the Resource class. It stores information about one type of the resources- Uranium.

Superclasses

- **Resource**

8.1.10 Water ice

Responsibility

Water ice is a child class of the Resource class. It stores information about one type of the resources- Water Ice. It also contains a method for water sublimation (disappearance).

Methods

- **public void sublime()**: When an asteroid with water ice in its core is at perihelion, this water ice sublimates(evaporates).

Superclasses

- **Resource**

8.1.11 Iron

Responsibility

Iron class is a child of the Resource class. It stores information about one type of the resources- Iron.

Superclasses

- **Resource**

8.1.12 Carbon

Responsibility

Carbon class is a child of the Resource class. It stores information about one type of the resources- Carbon.

Superclasses

- **Resource**

8.1.13 Place

Responsibility

This class is a parent class of Asteroid and TeleportationGate. These two entities are places where a Visitor can be, and the Place class deals with them. It stores information about its neighbors, and the visitors of the places. It inherits its id from the GameObject class.

Attributes

- **protected ArrayList<Place> neighbors:** All neighbors of a certain Place are stored in a list- neighbors are Asteroids or Teleportation Gates on either side of another Place.
- **protected Visitor visitor:** Stores information about which visitor is currently visiting a place.

Methods

- **public void addVisitor (Visitor v):** adds a visitor v as a current visitor to the place.
- **public Visitor getVisitor:** gets information about which visitor is currently visiting a place, and returns it.
- **public bool removeVisitor(Visitor v):** removes a current visitor when it leaves the place, and returns true.
- **public Place[] getNeighbours:** gets neighbors of a certain place-the asteroids or teleportation gates on either side of it.
- **public void addNeighbour(Place p):** adds a new neighbor to a place.

Superclasses

- **GameObject**

8.1.14 Teleportation gate

Responsibility

This class stores information about teleportation gates that were deployed by settlers. These come in pairs, so it is important for two gates of a pair to be linked with one another, which is why we need a pairGate attribute, and methods to set and get a gate's pair. This is a child class of the Place class, since it is one of the two available types of places, and through Place, it inherits its ID from Game Object.

Attributes

- **private TeleportationGate pairGate:** Stores information about what other gate is connected with this gate, since they come in pairs.

Methods

- **public void setPair(TeleportationGate t):** Assigns one gate to another, as they come in pairs.
- **public TeleportationGate getPair():** Gets and returns the pair of a gate

Superclasses

- **Place->GameObject**

8.1.15. Asteroid

Responsibility

This class is a child class of Place, and through place it inherits an ID from Game Object. It is the parent class of RadioActiveAsteroid class, since radioactive asteroids are one distinct type of asteroid. This class stores information about whether asteroids are hollow, the depth of their rock mantle, and which resource the core is made from. Also, it contains methods such as deepening the rock mantle, adding and removing resources, determining whether the asteroid is hollow, and whether or not it is at perihelion.

Attributes

- **protected bool hollow:** stores information about whether an asteroid has a hollow core or not. A core is hollow when there is no resource.
- **protected int depth:** the depth of the rock mantle varies, and this attribute stories the depth of the mantle.
- **protected Resource resource:** stores information about the type of the resource which makes up the core of an asteroid.
- **protected int distanceFromSun:** Indicates the distance between a radioactive asteroid and the Sun. This is important to be able to determine when the asteroid is at aphelion or perihelion.

Methods

- **public void deepenHole(int n):** The rock mantle is deepened by one unit. The int parameter would be useful in case of requirement change, so the mantle could be deepened by a different amount.

- **public Resource getResource():** Returns the name of the resource that makes up the asteroid core.
- **public bool addResource(Resource r):** Adds resource r to the core of an asteroid and returns true if the action is performed successfully.
- **public void removeResource():** Removes resource from the core of an asteroid.
- **public bool isHollow():** Checks if an asteroid has a hollow core.
- **public bool isPerihelion():** used to check if an asteroid is at perihelion position. Important for radioactive asteroids, and those with water ice in their core.

Superclasses

- **Place->GameObject**

8.1.16 Radioactive Asteroid

Responsibility

This is a child class of Asteroids. It is needed because one type of asteroid- those with uranium in their core are radioactive, and can explode when at perihelion. Since it's a special behaviors of a particular asteroid and it as a whole retains properties of a normal asteroid we used inheritance

Methods

- **public void explode():** When an asteroid with a radioactive core is at perihelion, it will explode, killing settlers, and displacing robots.

Superclasses

- **Asteroid->Place->GameObject**

8.1.17. Sun Storm

Responsibility

This class stores information about the sun storm- when it occurs. It also contains the methods needed for the collisions with objects of the game. The sun storm inherits its ID from the Game object.

Attributes

- **private int time:** stores a time value for setting the time of sun storm occurrence.

Methods

- **public void setTime(int t):** Sets the time t when the sun storm will occur.
- **public void collisionWith(GameObject obj):** Performs the collision of the sun storm with one of the objects in the game.

Superclasses

- **GameObject**

8.1.18. ID

Responsibility

This class is an enumeration class that consists of predefined list values Settler, Robot, Asteroid, RadioActiveAsteroid, TeleportationGate, SunStorm, Iron, Carbon, Uranium, and WaterIce.

8.1.19. Direction

Responsibility

The direction is an enum containing the movements (UP, DOWN, LEFT, and RIGHT) which are used to handle the movement of the spaceship.

8.2 Detailed plan of testing

8.2.1 Start Game

- **Description:** A test case for testing whether the game will start when the “START” is clicked or pressed.
- **Unit of functionality to be tested, possible failures:**

Function: StartGame()
Class: Game

- **Failure:** The major elements of the game are not successfully initialized.

- **Input**

In terms of console based, “start” input is entered. But, in the GUI based game, start menu is clicked by “Mouse_Click” from the menu section.

- **Output**

  Game object are initialized successfully.

  Prints out the list of object, such number of asteroids, and info about resources.

8.2.2 Settler traveling

- **Description:** A test case for testing whether the settler will move and its coordinate changes after the corresponding button is pressed.
- **Unit of functionality to be tested, possible failures**

  Function: travel()

  Class: Visitor()
  Failure: 1. If the Place the settler is attempting to travel to is not a neighbor, the settler will not be able to travel.

- **Input**

In terms of console based game, “A” and “W” input are entered to travel between asteroids. However, as a GUI based game, “UP”, “DOWN”, “RIGHT”, “LEFT” keyboard keys are expected to control the settler.

- **Output**

  Settler has traveled to a neighboring asteroid successfully.

8.2.4 Drilling

- **Description:** A test case for testing whether the settler will drill the asteroid when the corresponding virtual key is entered
- **Unit of functionality to be tested, possible failures**
  - **Function: drill()**
  - **Class: Settler**
  - **Failure:** 1. Asteroid is already fully drilled.
    - 2. Settler is not on the asteroid
    - 3. The mantle of the asteroid is not decremented by one.
- **Input**

  “D_Key”

- **Output**

  If the position of the asteroid is perihelion and it is radioactive then the game is terminated and the settler dies.

  Else, With this command. The settler will drill the core of an asteroid, therefore the size of the asteroid should decrease.

  Mantle of the asteroid is decremented by value one.

8.2.5 Mining the Asteroid

- **Description:** Settler can only mine fully drilled asteroids when it is aphelion.
- **Unit of functionality to be tested, possible failures**
  - **Function: mine()**
  - **Class: Settler**
  - **Failure:** 1. Asteroid is hollow
    - 2. Asteroid is not fully drilled
    - 3. You do not have enough inventory space to get the resource.
- **Input**

  “M_key”

- **Output**

  You have mined $resource from the asteroid successfully.

8.2.6 Build the space station

- **Description:** Testing whether the settler will be able to build the space station while the settler has the required resources or not.
- **Unit of functionality to be tested, possible failures**
  - **Function:buildSpaceStation()**
  - **Class: Settler**
  - **Failure: 1.** You do not have enough necessary resources.
- **Input**

  “S_Key”

- **Output**

  Space station has been built successfully.

  Settler has won the game.

8.2.7 Building the teleportation gate

- **Description**

Settler only can build teleportation gates if it has enough necessary resources and its inventory has available spaces since settler only carries a pair of gates at the same time.

- **Unit of functionality to be tested, possible failures**

  Function: buildTeleportationGates()

  Class: Settler

  Failure: 1**.** You do not have enough necessary resources.

  2. Not enough spaces: You already have undeployed teleportation gates.

- **Input**

  “T_Key”

- **Output**

  Teleportation gates have been built successfully.

8.2.8 Building the robot

- **Description**

Settler only can build robot if it has enough necessary resources. Build robots will be controlled by the system autonomously.

- **Unit of functionality to be tested, possible failures**

  Function: buildRobot()

  Class: Settler

  Failure: 1**.** You do not have enough necessary resources.

  2. There is no neighboring asteroid to place it.

- **Input**

  “R_Key”

- **Output**

  Robot has been built successfully.

  Robot will be controlled autonomously.

  Robot is landed on the neighboring asteroid.

8.2.8 Filling the Asteroid

- **Description**

  Settler fills the hollow and fully drilled asteroid with a unit of resource.

- **Unit of functionality to be tested, possible failures**

  Function: fillAsteroid()

  Class: Settler

  Failure: 1. Asteroid is not fully drilled.

  2. Asteroid is not hollow.

  3**.** You do not have enough resources.

- **Input**

  “F_Key”

- **Output**

  Asteroid has been filled with $resource successfully.

8.2.8 Hiding in asteroid

- **Description**

A settler can hide inside a hollow asteroid that has its mantle drilled through to stay safe from dangers.

- **Unit of functionality to be tested, possible failures**

  Function: hide()

  Class: Visitor

Failure: 1. Asteroid is not fully drilled.

2. Asteroid is not hollow.

- **Input**

  “H_Key”

- **Output**

  You are hiding the core of the asteroid.

8.2.7 Deploying the gates

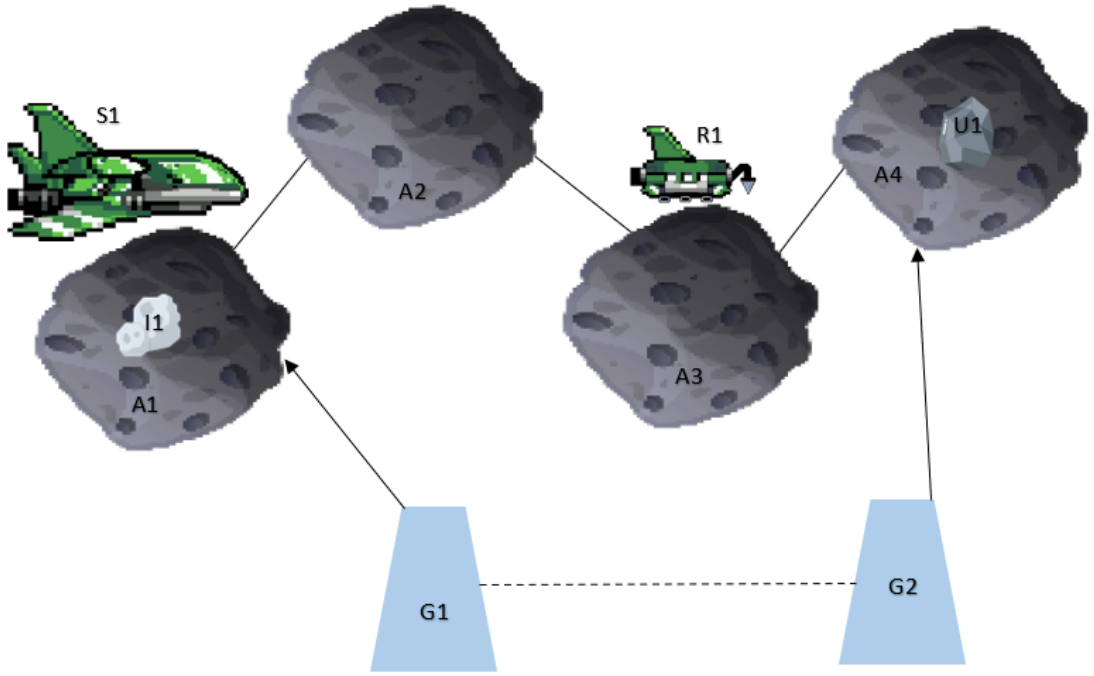
- **Description:** Testing the ability to deploy the teleportation gate. Settler only can deploy available newly built teleportation gates. Settler deploys one gate at a time at different asteroids.
- **Unit of functionality to be tested, possible failures**
  - **Function: deployGate()**
  - **Class: Settler**
  - **Failure: 1.** You do not have available teleportation gates.
    - 2. Please deploy the 2nd gate to a different place.
    - 3. The lack of neighboring asteroids.
- **Input**

  “G_Key”

- **Output**

  Gate has been deployed successfully.

8.3 Test Case Scenario



Initialization:

- A1, A2, A3, A4 asteroids are neighbors in the sequence
- Gate G1 is the neighbor of A1 and gate G2 is the neighbor of A4.
- Settler S1 is placed on the A1
- Robot R1 is placed on the A3
- A1 is drilled, contains iron I1
- A4 contains, U1 uranium

Control(commands):

- S1 mines I1
- R1 travels to A4
- S1 steps to G1 and reaches to G4
- R1 drills the A4
- A4 is perihelion

Expected continuation:

- A4 explodes
- R1 lands on A3
- G1 is destroyed and pulls G2 with it

Check of results:

- Remaining object display their states

8.4 Plans of the supporting programs

At the current phase of the game program, we have developed a well working skeleton program which makes users able to already test the game by selecting the actions from the console menu and see its printed output values, such as methods and response texts as well user interface questions.

However, for the future features and complete version of the application, we are planning to make the program be able to be tested through both JUnit testing and text file. Text file contains all the input commands, and it is the expected output.

8.5 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
8th April: 10:00am - 11:30am	1.5 Hours	All team mates	Looked over the mentors comments and decided our further steps by dividing ourselves in groups of 3 and taking 1 part each group.
10th April: 2:00pm- 4:30pm	2.5 hours	Neda, Jack and Tushig	Worked on the first part of the assignment which involved discussions on the topics pointed out by the professor and correcting them as per the feedback
11th April 5:00am - 7:30am	2.5 hours	Chaitanya and Desoki	Worked on the second part of the assignment and took points from the previous submissions and changed them as per the remarks given by the professor.

12th April 10:00am - 12:00am	2 hours	Janibyek, Neda and Kasay	Worked on the remaining artifacts and completed the whole documentation.

10.1 Deployment guide

10.1.1 List of files

Filename	Size	Date	Content
Asteriod.java	2.53KB	2022.04.24	The class contains the GUI and major attributes of the class Asteroid like depth, hollow, distancefromSun etc, and some functions required in rendering the file, getting resources, deepen hole etc which covers most of the use case related to asteroid
Carbon.java	160 Bytes	2022.04.24	Carbon class is a child class of resource
Direction.java	101 Bytes	2022.04.24	The direction is an enum containing the movements up. Down, left right, which is used to handle the movement of the spaceship

Game.java	4.85 KB	2022.04.24	The game is one of our main classes, which is initiating the game by beginning the threads when the game begins and renders all objects like asteroids, spaceships etc, which need to be displayed. It contains functions like render, tick and run which control the main structure of the game.
GameObject.java	296 Bytes	2022.04.24	Besides having the normal getter ,setter. This function is used to change the movements of the objects like asteroids and spaceship and check for collisions
Handler.java	1.26KB	2022.04.24	Handler class is for getting neighbor place, adding and removing objects, getting settler, and checking if the asteroid is explosive.
ID.java	230 Bytes	2022.04.24	Enumeration of Settler, Robot, Asteroid, RadioActiveAsteroid, TeleportationGate, SunStorm, Iron, Carbon, Uranium, and WaterIce.
Iron.java	152 Bytes	2022.04.24	Iron class is a child class of resources.

Place.java	1.22KB	2022.04.24	The place class is superclass of
RadioactiveAsteriod.java	725 Bytes	2022.04.24	RadioActiveAsteroid is a superclass of Asteroid with explode method.
Resources.java	196 Bytes	2022.04.24	Class defines the type of resources.
Settler.java	1013 Bytes	2022.03.26	The settler class is the GUI of the settler and will involve the movements and basic rendering.
Robot.java	352 Bytes	2022.04.24	The robot class is the graphical user interface of the robot which involves the movements, basic rendering, and it can get damage.
Settler.java	7.21KB	2022.04.24	The settler class is the GUI of the settler and will involve the movements and basic rendering.
Spaceship.java	1.57 KB	2022.04.24	This class is responsible for initializing spaceship with its capacity and current inventory.

Sunstorm.java	957 Bytes	2022.04.24	SunStorm class is GUI implementation sunstorm itself.
Teleportationgate.java	399 Bytes	2022.04.24	The TeleportationGate class is the graphical user interface of the gates.
Uranium.java	164 Bytes	2022.04.24	Uranium class is a child class of resource
Visitor.java	1.70KB	2022.04.24	Visitor class extends the game object and deals with the what things can be done when a settler visit the asteroid and similar functions like drill, hide etc.
WaterIce.java	307 Bytes	2022.04.24	WaterIce class is a child class of resource. It differs with sublime() method from other resources.

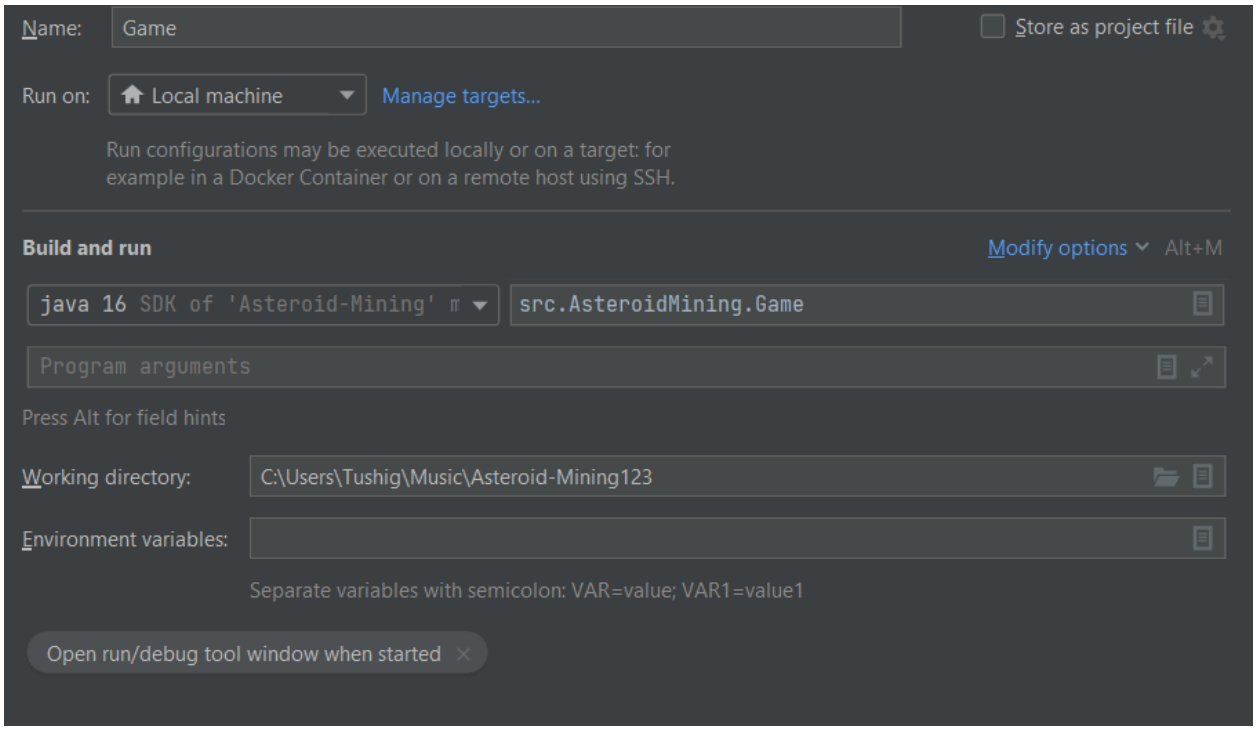
10.1.2 Compilation

In order to do the successful compilation of the code, one can clone the project or download the zip file and run it in Intellij IDE.
To execute the program we need to run the main function present in the Game.java file. Namely, the entry point is this function.
The link to github repository:
https://github.com/jackdotb/Asteroid-Mining

10.1.3 Run

There are no particular requirements except the fact that jdk should be installed in the users IDE and should preferably use intellij IDE. In order to compile the project successfully, “Game” class is needed to configured in “Run/Debug Configuration” section of the IntelliJ IDE. Especially, some of test cases are suggested that they should be checked in debugging mode with breakpoints.

Run/Debug Configuration:



10.2 Test protocols

10.2.1 Starting the game

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 22:42

10.2.2 Build a space station

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 22:45

10.2.3 Settler traveling

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 22:45

10.2.4 Settler drilling

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 22:43

10.2.5 Settler mining

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 22:43

10.2.6 Filling the asteroid

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 22:44

10.2.6 Is asteroid in perihelion (asteroid explosion)

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 23:08

10.2.7 Building the robot

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 22:52

10.2.8 Building the teleportation gate

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 22:52

10.2.9 Deploying the teleportation gate

Name of the tester	Neda Radonjic

Date & time of test	24/04/2022 23:03

10.2.10 Hiding the settler

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 23:03

10.2.12 Death of the settler

Name of the tester	Neda Radonjic
Date & time of test	24/04/2022 23:06

10.2.13. Checking inventory

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 23:12

10.2.14 End game

Name of the tester	Chaitanya Arora
Date & time of test	24/04/2022 23:15

Result (failure)	No output
Possible causes	We forgot to add a switch case for this scenario.
Correction	A switch case was added for this scenario, and the repeated test was successful.

10.3 Evaluation

Name of the team member	Participation (%)
Janibyek Bolatkhan	16.66%
Kasay Ito	16.66%
Chaitanya Arora	16.66%
Tushig Bat-Erdene	16.66%
Neda Radonjic	16.66%
Abdelrahman Desoki	16.66%

10.4 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
24/04/2022	1,5 hours	Chaitanya and Neda	Finishing the documentation and testing the code.
23/04/2022	2 hours	Janibyek	Worked on the prototype program by extending the skeleton code. Fixed bugs in the methods.
24/04/2022	1,5 hours	Desoki	Debug some methods and test the program.
24/04/2022	1 hour	Tushig	Reviewing the code and commenting methods and classes.
20/04/2022	1,5 hours	All team	Discuss the last review and check the methods that don't work, separate tasks, and schedule meetings.
23/04/2022	1 hour	Kasay	Fixing a bug

User interface specification

11.1 Graphical User Interface

Figure 1. Introduction screen of the game

Figure 1 displaying the first screen of the game when it is successfully executed or compiled. The “PLAY” button is responsible for starting the game and the “STOP” button is responsible for terminating the game.

Figure 2. Game scenario

Figure 2 is demonstrating one of the scenarios of the game. Player can see the storage from the left bottom part of the game screen which is displaying the resources that the player gathered during the game. The “Operations” button is responsible for screening the information of the important keys for actions such as drilling, mining, moving and so on.

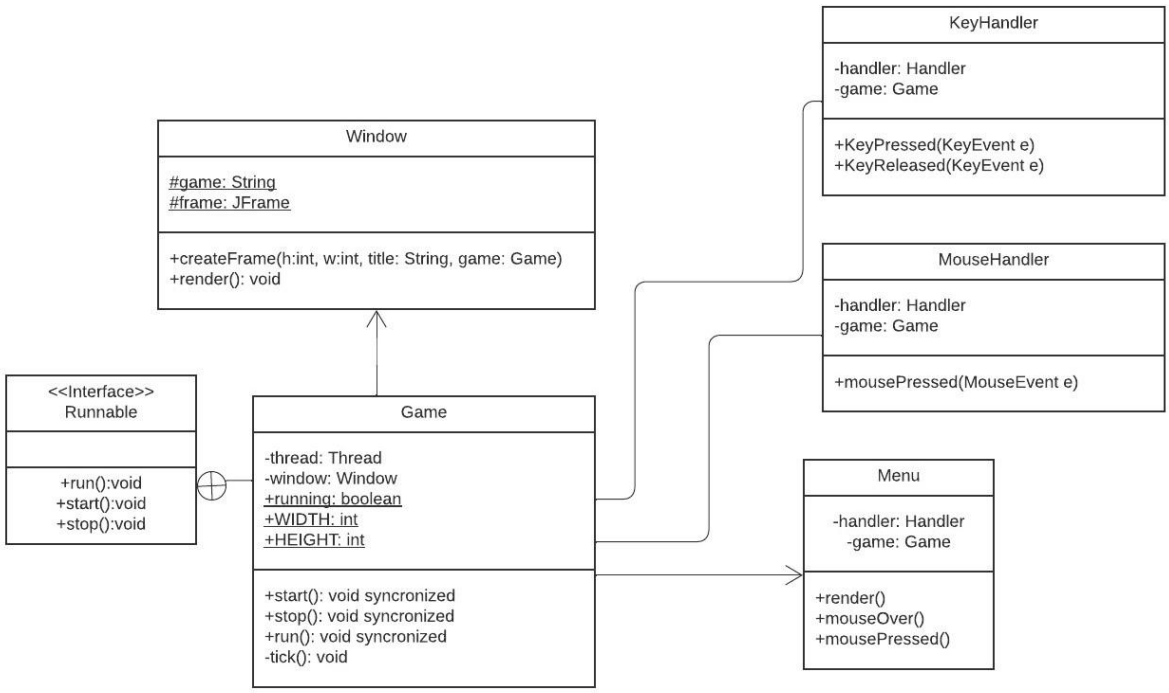
11.2 Architecture of the graphical system

11.2.1 Principles of the GUI

Our solution is based on simple Model-View-Controller (MVC) model.

- Model is considered as class Game and class Handler as abstraction, namely, these two classes can pass information from Controllers to Views although the authentic data is stored in other objects (e.g. Settelr, Asteroid).
- Controllers (i.e. KeyHandler, MouseHandler) can access Model only through class Game or class Handler.
- View is comprised of a class Window for game itself and a class Menu for menu in the game.

11.2.2 GUI Structure diagram



11.3 GUI Classes

We abbreviate some functions which come from super class or interfaces since they do not relate to necessary and important logic behind the UI or GUI specification.

11.3.1 Window

·Responsibility:

The Window class's responsibility is to create the JFrame window. This class outputs according to the computed data by Model. This is one of View in MVC model.

Attributes

- **#game: String;** distinguishes windows in case there are multiple windows
- **#frame:JFrame:** Jframe of the window class

Methods

- **public void createFrame(h:int, w:int, title: String, game: Game):** it sets the dimensions settings of the given frame with corresponding parameters
- **+render():void -** renders (repaints) the actual graphics of the game. For example, frame-per-second is determined in Model, however, this class repaints the screen actually.

11.3.2 Game

- **Responsibility:**

This class is the main class of the program. It is responsible of starting(initializing) and ending the program as well as containing important handling classes and all existing objects. For the User interface of the program, it is extending the built in class Canvas to be able to be drawn and rendered. And implementing the built in Runnable interface to be run as a multi-threaded program. Please refer back to the past class diagram for more precise responsibility. If there are corresponding events notified by Controllers or Views, this has the responsibility for defining which functions are callbacks. This is one of Model in MVC model.

- **Superclasses**

- **Game inherits from Canvas.**

- **Interfaces**

- **<<Interface>> Runnable**

- **Attributes**

- **-thread: Thread -** in order to fetch the signal and update information, it uses a thread
- **-window: Window** - the game window
- **+running:boolean -** representes the current state
- **+WIDTH:int -** indicates the width of the game
- **+HEIGHT:int -** indicates the height of the game

- **Methods**

- **+start():void synchronized -** Creates a new Thread.
- **+stop():void synchronized -** Stops the Thread. This can be used for pausing the game, therefore, we do not specify the dynamic behaviour as a sequence diagram, but we keep this remained for further extendability.
- **+run():void synchronized -** runs the program by setting frame per second as well as executing the methods render and tick method.
- **-tick():void -** it calls the tick methods of the all objects which are changeable. For example, this method is used by the Visitor class the set the location. Therefore, this function can manage time-based behaviour.

11.3.3 Menu

- **Responsibility:**

Menu page window of the game program where user can select the state of game program. Currently, menu has 3 option, such Play, instructions, exit. This class inputs and outputs

according to the computed data by Model. This is one of View-Control in MVC model. (because of superclass Canvas)

- ### Superclasses

- **Menu inherits from Canvas.**

- ### Attributes

- **-handler:Handler -** handler of the all the existing objects in the Asteroid belt. This is described in the former class diagram.
- **-game:Game:** game class of the program which is used to accessing the important attributes and objects of the game since it is main class

- ### Methods

- **+render():** renders the graphics of the menu page. For example, frame-per-second is determined in Model, however, this class repaints the screen actually.
- **+mouseOver():** checks if the mouse is on the menu setting
- **+mousePressed():** takes the mouse presses event and checks which menu setting options is pressed and thus changes the state of the game by corresponding chosen option.

11.3.4 KeyHandler

- ### Responsibility

This class has responsibility such that it must handle the event related to key inputs. The behaviour for an input depends on each key. This class notifies Model about events. This is one of Controller in MVC model.

- ### Superclasses

- **KeyHandler inherits from KeyAdapter.**

- ### Attributes

- **-handler:Handler -** handler of the all the existing objects in the Asteroid belt. This is described in the former class diagram.
- **-game:Game:** game class of the program which is used to accessing the important attributes and objects of the game since it is main class

- ### Methods

- **+KeyPressed(KeyEvent e)-** Deals with the events of Direction changing(left, right, up,down), Drilling, Hiding, and the like depending on the key pressed.
- **+KeyReleased(KeyEvent e)-** Deals with the events when the key is released from pressing state.

11.3.5 <<Interface>> Runnable

- **Responsibility:**

It makes the class with thread runnable and gives the ability to override its methods. Since this is a built-in class, we do not explain the details in this documentation.

11.3.6 MouseHandler

- **Responsibility:**

Handles the all the mouse click events of game screen. This class notifies Model about events. This is one of Controller in MVC model.

- **Superclasses**

- **KeyHandler inherits from MouseAdapter.**

- **Attributes**

- **-handler:Handler -** handler of the all the existing objects in the Asteroid belt. This is described in the former class diagram.
- **-game:Game:** game class of the program which is used to accessing the important attributes and objects of the game since it is main class

- **Methods**

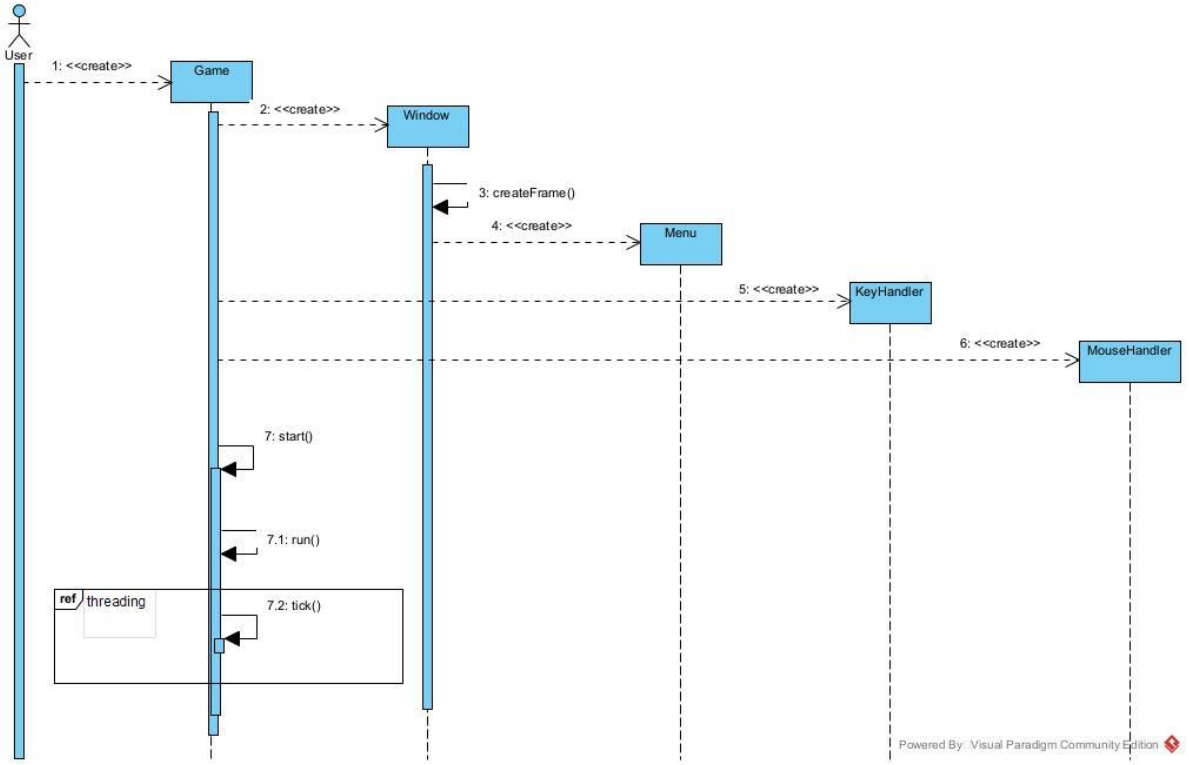
- **+mousePressed(MouseEvent e)-**Gets the coordinates of the mouse click event.

11.4 Dynamic connection between the model and the GUI

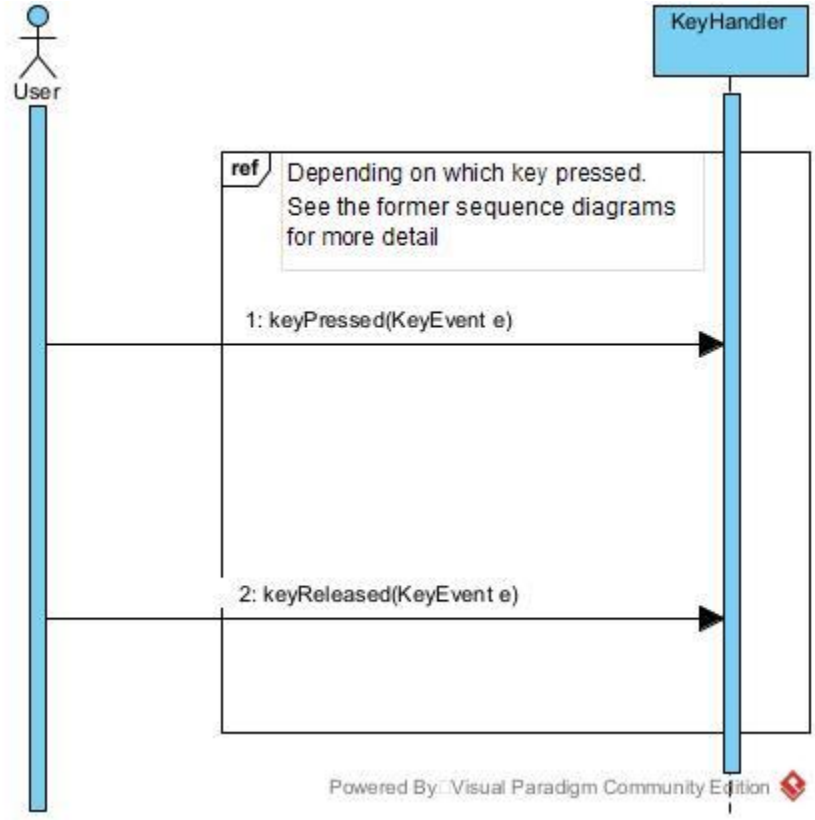
The function “stop()” in class Game is not described in any of sequence diagrams, however, the reason for that is due to the fact that the interface Runnable has stop() and pausing of the game can affect the behaviour of GUI as well.

We do not specify about pausing of a game because it is out of our scope now.

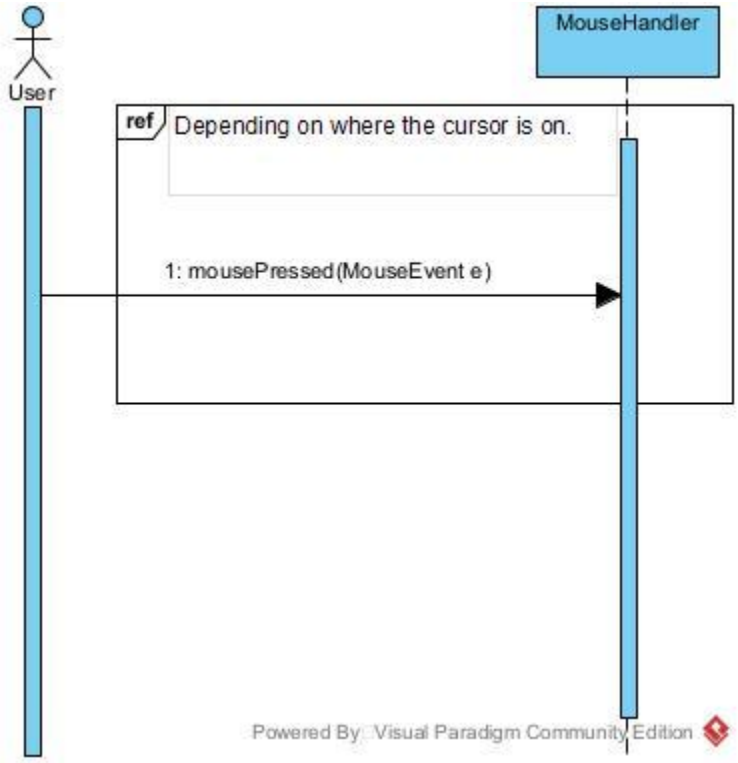
11. 4. 1. initialization



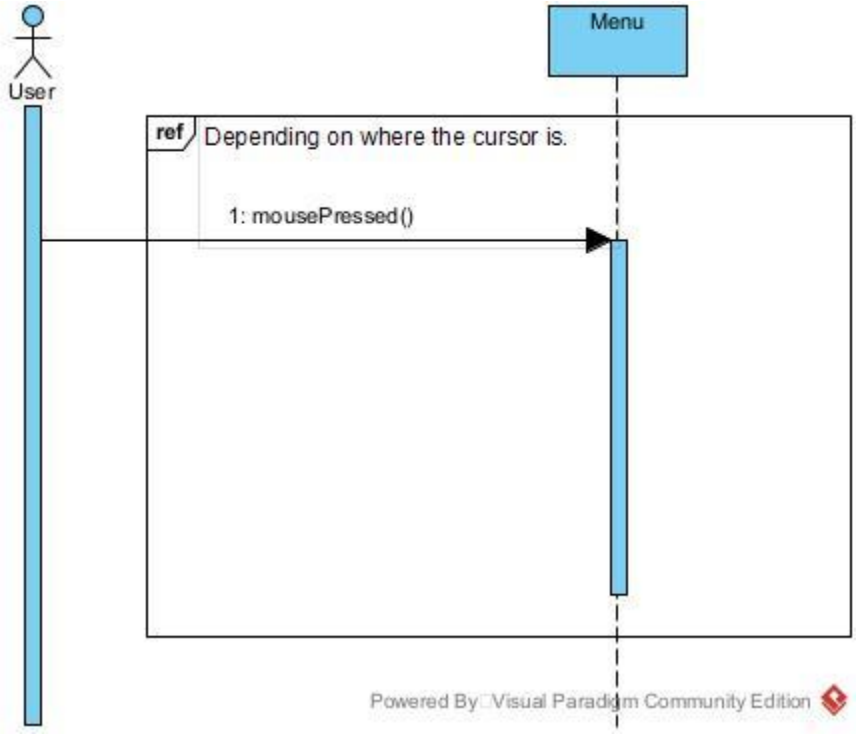
11. 4. 2. control key



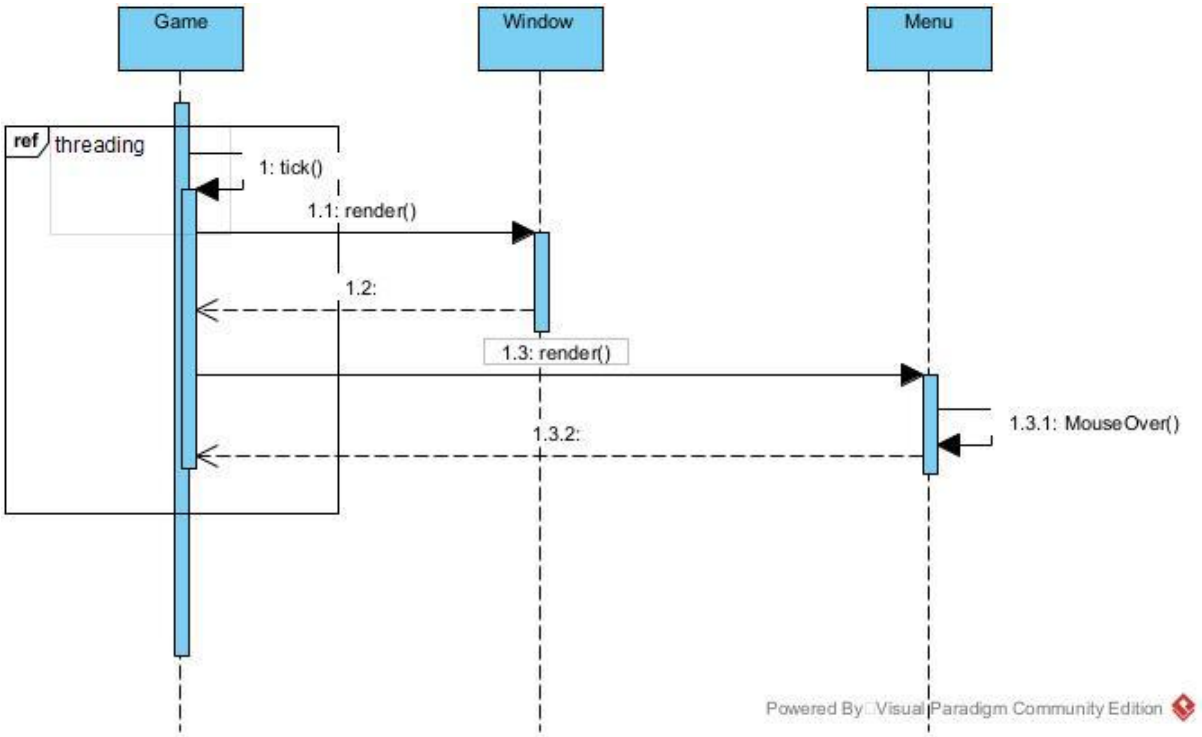
11. 4. 3. control mouse



11. 4. 4. selecting on menu



11. 4. 5. during game



11.5 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
29/04/2022	**2.5 hours**	**The whole team**	**Discussing the needed changes, creating the new class structure diagram and class descriptions, and getting started on the sequence diagrams.**
1/05/2022	3 hours	The whole team	Finalizing the GUI structure diagram, completing the class descriptions and sequence diagrams.

2/05/2022	2 hours	The whole team	Finalizing the documentation and discussing the future steps.
02/05/2022	2 hours	Kasay	really finalizing principles of GUI, class diagram, responsibility for each class and sequence diagrams

13. Complete program

13.1 Deployment guide

13.1.1 List of files

File name	Size	Date	Content
Asteriod.java	4 KB	2022.05.17	The class contains the GUI and major attributes of the class Asteroid like depth, hollow, distancefromSun etc, and some functions required in rendering the file, getting resources, deepen hole etc which covers most of the use case related to asteroid
Carbon.java	1 KB	2022.05.17	Carbon class is a child class of resource
Direction.java	1 KB	2022.05.17	The direction is an enum containing the movements up. Down, left right, which is

			used to handle the movement of the spaceship
Game.java	5 KB	2022.05.17	The game is one of our main classes, which is initiating the game by beginning the threads when the game begins and renders all objects like asteroids, spaceships etc, which need to be displayed. It contains functions like render, tick and run which control the main structure of the game.
GameObject.java	2 KB	2022.05.17	Besides having the normal getter ,setter. This function is used to change the movements of the objects like asteroids and spaceship and check for collisions
Handler.java	2 KB	2022.05.17	Handler class is for getting neighbor place, adding and removing objects, getting settler, and checking if the asteroid is explosive.
ID.java	1 KB	2022.05.17	Enumeration of Settler, Robot, Asteroid, RadioActiveAsteroid, TeleportationGate, SunStorm, Iron, Carbon, Uranium, and WaterIce.

Iron.java	1 KB	2022.05.17	Iron class is a child class of resources.
KeyHandler.java	3 KB	2022.05.17	Handles the key input events.
Place.java	2 KB	2022.05.17	The place class is superclass of
RadioactiveAsteriod .java	1 KB	2022.05.17	RadioActiveAsteroid is a superclass of Asteroid with explode method.
Resources.java	1 KB	2022.05.17	Class defines the type of resources.
Robot.java	2 KB	2022.05.17	The robot class is the graphical user interface of the robot which involves the movements, basic rendering, and it can get damage.
Settler.java	10 KB	2022.05.17	The settler class is the GUI of the settler and will involve the movements and basic rendering.
Spaceship.java	2 KB	2022.05.17	This class is responsible for initializing spaceship with its capacity and current inventory.

Sunstorm.java	2 KB	2022.05.17	SunStorm class is GUI implementation sunstorm itself.
Teleportationgate.java	1 KB	2022.05.17	The TeleportationGate class is the graphical user interface of the gates.
Uranium.java	1 KB	2022.05.17	Uranium class is a child class of resource
Visitor.java	3 KB	2022.05.17	Visitor class extends the game object and deals with the what things can be done when a settler visit the asteroid and similar functions like drill, hide etc.
WaterIce.java	1 KB	2022.05.17	WaterIce class is a child class of resource. It differs with sublime() method from other resources.
Window.java	1 KB	2022.05.17	Window class has topmost GUI functionalities in this application.

13.1.2 Compilation

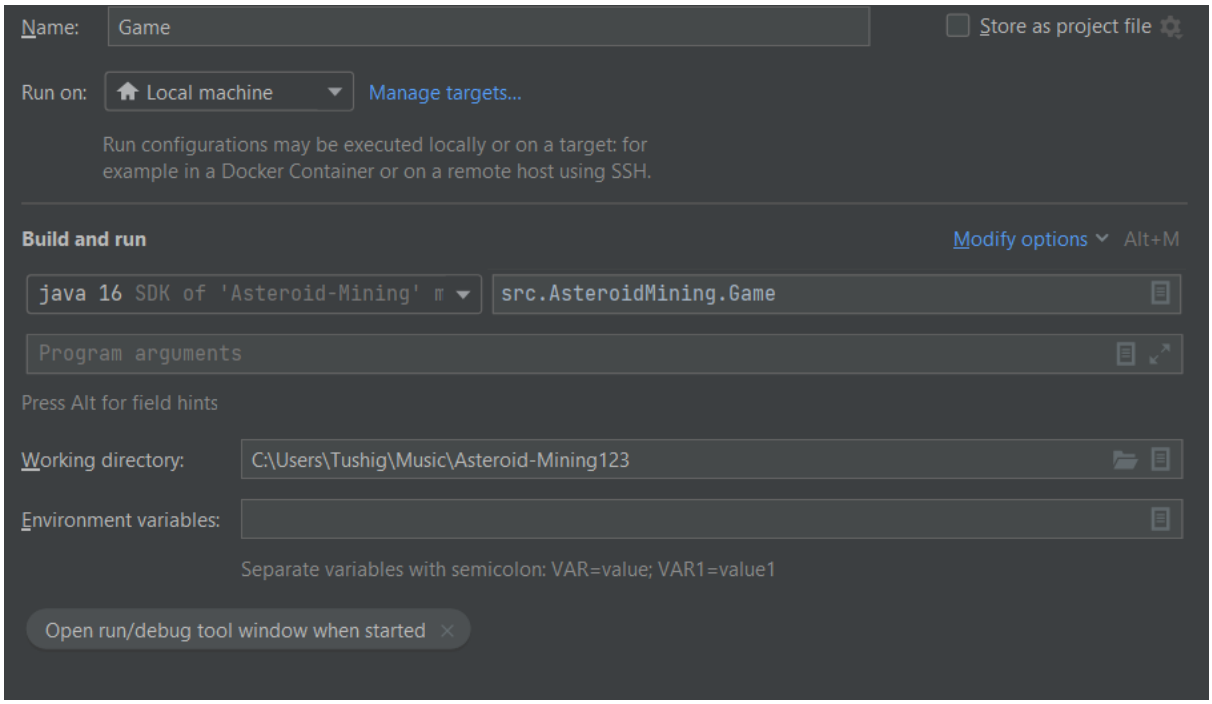
In order to do the successful compilation of the code, one can clone the project or download the zip file and run it in Intellij IDE.
To execute the program we need to run the main function present in the Game.java file. Namely, the entry point is this function.

The link to github repository:
https://github.com/jackdotb/Asteroid-Mining

13.1.3 Run

There are no particular requirements except the fact that jdk should be installed in the users IDE and should preferably use intellij IDE. In order to compile the project successfully, “Game” class is needed to configured in “Run/Debug Configuration” section of the IntelliJ IDE. Especially, some of test cases are suggested that they should be checked in debugging mode with breakpoints.

Run/Debug Configuration:



13.2 Evaluation

[Evaluation shows every team member's participation rate in the project (percents) **from the beginning of the project** to the date of evaluation. The sum of percent values must be 100.]

Name of the team member	Participation (%)

Abdelrahman Desoki	16.67%
Neda Radonjic	16.67%
Tushig Bat-Erdene	16.67%
Chaitanya Arora	16.67%
Janibyek Bolatkhan	16.67%
Kasay Ito	16.67%

13.3 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
11/5	*6-9 pm*	*Tushig, Desoki, Neda, Arora, Jack*	*Discuss the main methods that need to be updated, as well as designing of sunstorm, and the hole in the asteroid.*

12/5	1-2 pm	Jack, Desoki	Implementing hide method , and connecting the GUI with the Logic.
15/5	1 hour	Kasay	Defining algorithm for autonomous robots.
16/5		Janibyek	Implementing missing graphics for the main functions and checking the quality of code to program to be run smoothly.
15/5	1 hour	Tushig	Implementing drill GUI scene

14. Summary

14.1 Experiences of the project

What did you learn from this project (concretely and generally)?

Working in a team was a fairly new experience for most of us, and it largely differed from independent work, and gave as an insight into how it is to work on projects in real life. We also learnt the importance of the development cycle, and realized that each step is crucial in order to deliver a good result, even if it seems less important at times.

If initializing a list based on our acknowledgement, we have learnt:

- Planning the project under the specification
- Organizing the team
- Implementing the diagrams based on specification
- How analysis model and designing model works

- Developing the project based on UML diagrams
- Testing the program or application
- Understanding the main cycles of RUP
- GitHub, Git Version Control
- Decision making

What was the most difficult, easiest thing to do?

One of the most challenging parts of the project was following the software development cycle. Sticking to the process and not just jumping to code was a little difficult but it all seemed worth it in the end, as it was crucial for our deep understanding. As a team of 6, with many different ideas and points of view, the process of decision-making was sometimes slightly slow. At times, it was challenging to work around everyone's schedules and take the set timings seriously, as we worked in a pretty relaxed, informal environment.

The few of the easiest things that we did during the project was making use case diagrams and following the methodology of identifying requirements. Even though it was a trivial process, it was very important to help us understand the game in a much better way which helped us not just save time but gave us the opportunity to think about the implementation instead of jumping to coding directly. The feedbacks given by Prof. Katalin were very detailed which was very helpful for understanding our mistakes and staying on track.

Were time and points conform to the tasks?

Deadline time and points of each task were distributed well except on some occasions. We explained this in the next question.

If not, where do you think it caused difficulties?

As we experienced throughout the semester, the deadline for the several big tasks was really short, one week. We needed to schedule meetings at least 2-3 times a week, online and offline, to complete tasks perfectly, which sometimes influenced negatively on other courses' progress.

In terms of points, we think that if the point of the last assignment- Complete Program was higher, it would result in students handing out the last program more perfectly and working properly.

Have you got any suggestions to change something?

The course was designed well but there are some suggestions that we as a team will can be better:

If we were the ones designing the course we would:

- Be a little flexible with deadlines, as sometimes due to midterms and retakes it becomes difficult to finish the submissions.
- Having some of the descriptions and requirements stated clearly as we felt many times they were a little ambiguous which led us to make mistakes.
- Having examples from previous years' submissions were really helpful to understand what is needed, how long it should be, and what an ideal submission looks like. On the occasion when we were given these examples, this made us work very productively, and it would have been beneficial if we had access to something similar throughout the course.

What kind of tasks would you recommend to be a project like this?

We recommend projects like full-stack apps in Java which helps make a project not just for the course and our resume but also can be taken forward as a potential product or a business opportunity.

14.2 Protocol

Start (date & time)	Duration (hours)	Performer(s) name	Activity description
19.05.2022	1 hour	All the team members	Divided questions between us and discussed each of the questions together so that everyone is on the same page.