Asteroid Mining

Team name: Pied Pipers

Title: Detailed Plans

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

April 12, 2022

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

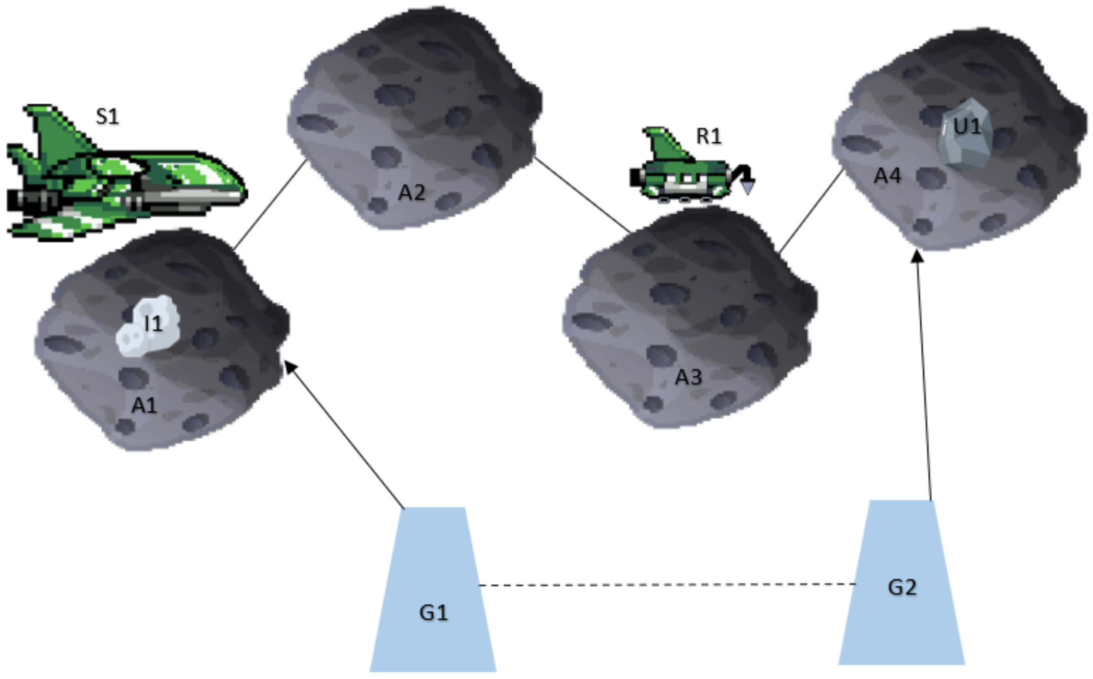
- **Description:** Testing the ability to deploy the teleportation gate. Settler only can deploy available newly built teleportation gates. Settler deploys one gate at a time at different asteroids.
- **Unit of functionality to be tested, possible failures**
  - **Function: deployGate()**
  - **Class: 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.