[8# Detailed Plan]

[5] – [SoftProLab Team]

Supervisor:
Dr. Katalin Balla

Members:

Hagverdiyev Subhan	NHL9KN	subhan.hakverdiyev@gmail.com
Madatov Ali	XVEARA	ali.madatov@hotmail.com
Shoaib Areeba Tabassum	EZXFWK	areebashoaib99@gmail.com
Salahov Kamal	IG5LSM	salahovkamal@hotmail.com
Singh Gurdeep	ERXIPV	sandhugoldy433@gmail.com

12.04.2021

8. Detailed plans

[The goal of the document is to precisely specify the classes to be implemented. This includes private attributes and methods as well.
The second part of this document lists with details all the test cases needed to fully test the prototype. Inputs and outputs should be described in the languages defined in the previous document.]

8.1 Design level plan of classes

8.1.1 Place class

- **Responsibility**

Place class is used for creating a base class for all subclasses that represent a place and can contain other smaller objects like settler,robots, minerals etc. Place class should have other neighboring places and must be able to handle a transfer of object from/to an neighboring place. Place class is public abstract class

- **Superclasses**

- **Interfaces**

- **Attributes**
  - **places**: This attribute is protected list that contains place-derived objects which represent neighbors of place-derived object. We can find any neighboring place from this list using getter method. When game is initialized list is filled with according places.
  - **travellers**: This attribute is protected list that contains traveller-derived objects which represent travellers on the surface of this place. Places may contain any number of travellers. When game is initialized list can be empty.

- **Methods**
  - **void accept(Traveller o)**: Acceptes traveller from the parameter by adding it to the end of travellers attribute. Public.
  - **void remove(Traveller o)**: Removes traveller from the parameter from the travellers attribute. Public.
  - **void setNeighbor(Place o):** Adds place object from the parameter to the end of places attribute. Public.
  - **List<Place> getNeighbor():** Returns the places attribute of the place object. Public

8.1.2 Asteroid class

- **Responsibility**

Asteroid class should serve as a place where main part of the game will happen. Actions such as drilling, mining, hiding etc happen to asteroid objects. Some functionality is derived from places class. Asteroid objects may or may not have an mineral in its core, has random depth of mantle, has a couple of states that it changes during the game. Asteroids may host traveller objects.

- **Superclasses**

Asteroid Class is derived from Place class.

- **Interfaces**

- **Attributes**
  - **state**: is a String type variable. Contains whether an asteroid is at normal/perihelion/aphelion state. All asteroids have random state at the start of the game.
  - **depth**: is a int type variable. Contains the length of the depth of mantle. If its zero it means that mantle has been drilled through, and now core can be mined if there's any. When Asteroid is created random number in range 1-7 is assigned to this attribute.
  - **hollow:** is a boolean type variable which contains whether core of an asteroid is hollow or not. Having this attribute makes it easier for determining outcome of other actions.
  - **mineralCore:** is a mineral type variable which stands for mineral mined or hidden in the core of asteroid. It's assigned randomly upon creation of asteroid if it's not hollow.
  - **mineralsSurface:** is a list containing mineral type objects. If settler drops mineral on the surface of asteroid then it's added to this list. Initially it's null.
  - **hiddenTraveller:** is a traveller object typed variable that represent traveller object hidden in the asteroid.

- **Methods**
  - **Mineral getCore()**: returns mineral in the core of the asteroid if there's any. Public.
  - **void setCore(Mineral m)**: sets mineralCore attribute to the mineral in the parameter of method. Public
  - **void setDepth(int d):** sets depth attribute to the value of parameter. Public
  - **int getDepth():** returns depth attribute of this object. Public
  - **String getState():** returns the state attribute of this object. Public
  - **void setState(String s):** sets the state attribute of this object to state given in the parameter. Public
  - **void explode():** It removes robots on the asteroid from their travellers attribute and transfers them to first neighboring asteroid.

8.1.3 Gate Class

- **Responsibility**

Gate class is responsible for representing transportation gate. Each gate has pair and travelling to one pair results in coming out of other pair. Until Deployed they remain as a unit.

- **Superclasses**

Gate class is derived from the places class.

- **Interfaces**

- **Attributes**
  - **Pair**: is a protected Gate type variable. Represents pair of this gate object. Every gate upon instantiation is attached a pair.
  - **Damaged:** is a public boolean type variable. Represents whether gate has been damaged or not. Initially value is always false.

- **Methods**
  - **void Teleport(Traveller t):** Teleports traveller object given in the parameter to the pair gate in pair attribute by removing from the list of travellers and transfering it to neighboring asteroid of the pair gate. Public.
  - **Gate getPair()**: returns the pair attribute of this object. Public
  - **void setPair(Gate g):** sets pair attribute of this object to the gate object given in the parameter. Public
  - **void hitBySunStorm():** damages given gate by changing value of damaged attribute to true

8.1.4 Carbon class

- **Responsibility**

Carbon class is used to represent carbon minerals. Although carbon class is not abstract, meaning that it can have instances, this class does not have any functionality.

- **Superclasses**

Carbon class is derived from Minerall class.

- **Interfaces**

- **Attributes**

- **Methods**

8.1.5 Traveller class

- **Responsibility**

Traveller is a superclass for Settler, Robot and Gate thus it contains common features like hiding, drilling etc. All interactions between the child classes and Asteroid are handled through Traveller class.

- **Superclasses**

None

- **Interfaces**

- **Attributes**

  currentPlace: is a protected Place type variable. Represents the current place of traveller it can either be settler or robot.

- **Methods**

  public void HitBySunStorm(): This method defines that when the object is going to get damages. It can get damaged when it gets hit by the sun storm.
  public void Drill(Asteroid a): This method defines that an object which can be either Settler or Robot, can drill the asteroid in order to find resources.It will decrease the mantle of the asteroid given in paremeter list by one unit.
  public void Travel(Place p): This method defines that the object can travel from one asteroid to another. It is carried out according to the chosen place in the parameter list.
  public void Hide():This method which is available only in the presence of hollow asteroid,defines that the object can hide inside the hollow asteroid to escape from the sunstorm. To achieve this we will check if the currentPlace variable is hallow astereoid. If it is we will call it when sunstorm happens.
  public void Destroyed(): Once the settler or robot drill a radioactive asteroid and reach to its core,the object will be destroyed. Thus, this method determines when the destruction will happen. It ca call the die function of asteroid as well.

8.1.6 Mineral class

- **Responsibility**

This class is responsible for all the resources and minerals present in a game. The minerals can be radioactive and non-radioactive. It depends on the settler which asteroid it wants to mine. The non-radioactive minerals can be used by settlers to build spaceship, robots or teleportation gates. This class will make sure that if the asteroid contains mineral such as water or ice and if that asteroid is at perihelion then the resources would melt/sublimates.

- **Superclasses**

None

- **Interfaces**

- **Attributes**

  type: is a protected String type variable. This attribute is going to tell which type of minerals are we going to
  use in the game.

- **Methods**

  public void CollectedBySettler(Settler s): This method tells that which type of resources and how many units of that resources the settler has collected. This method will contain a counter which will make sure about this. The result of this counter will be shown on the game screen which would be helpful for the player to keep in mind how many
  resources he still has to drill and mine. The more the resources, the more the counter, the more the score.

8.1.7 Robot class

- **Responsibility**

The responsibility of this class is to perform the functionalities almost similar to that of the settler. The robots can be built by settler if it has enough resources to build one. The robot class can be considered helpful for the settler as it can help the settlers to perform different functions which can lead to the winning of the game. It can make the players job a little bit easy.

- **Superclasses**

Traveller

- **Interfaces**

- **Attributes**

  None

- **Methods**

  public void HitBySunStorm(): The method defines that when the robot is going to get damages.It can get damaged when it gets hit by the sun storm. Depending on te robot’s health it can be destroyed or can survive.
  public void Destroyed(): The robot can get destroyed if it gets hit by the sun storm. This can cause serious damage to the robot and can often lead to its destruction.

8.1.8 Uranium class

- **Responsibility**

Similar to Iron class, this class is also child class of the mineral class. Uranium is considered as a dangerous resource in this game. If by mistake the settler mine the asteroid that contains uranium in peripholine, he will die.

- **Superclasses**

Mineral

- **Interfaces**

- **Attributes**

None

- **Methods**

public void explodeAsteroid(): This method determines if for instance in case a uranium is inside a asteroid and that certain asteroid is in prehelion state, it'll result it exploding. It will desturct the Asteroid also the travellers that are on it. It will call corresponding Destroyed and Die functions.

8.1.9 SunStrom

- **Responsibility**

The responsibility of this class is to make sure that sun storm occurs during the course of the game at any moment. A settler can survive the sun storm if it hides in a hollow asteroid. The sun storm can occur at any random moment and can vanish after a certain time. Sun storm can be the cause of the death of the player if it is not hiding which can lead to the failure of the game. Sun storm can destroy settlers, robots and teleportation gates

- **Methods**

- **Destroy(o: Travel):** This method describes the functionality of the sun storm i.e., it is going to destroy every object that comes in its ways. The object can be settler or robot.

  Pseudocode:
```
FOR each object of type Travel
  IF Travel Object Type is equal to Settler Object
     IF GetCurrentPosition of Settler is equal to CurrentPosition of Sunstrom
       CALL Die function of Settler
     ENDIF
  ENDIF
ELSE IF Travel Object Type is equal to Robot Object
     IF Current Position of Traveller type Robot is equal to CurrentPosition of Sunstrom
    CALL HitBySunStrom Function of Robot
     ENDIF
ENDIF
ENDFOR
```

- **Destroy(o:Gate):** This method is going to destroy all the gates built on a specific asteroid if the sun storm reaches the asteroid or gate.

  Pseudocode:
```
FOR each object of type Travel
   IF Travel Object Type is equal to Gate Object
     IF Current Position of Traveller type Gate is equal to CurrentPosition of Sunstrom
      CALL HitbySunStrom function of Gate
    ENDIF
  ENDIF
ENDFOR
```

- **GetCurrentPosition():** returns the position of sunstrom object

- **SetCurrentPosition(Sunstrom ss):** set the position of Sunstrom object at random Place.

8.1.10 Game

- **Responsibility**

This is the most essential class as it controls all the game from the player side. However, Game class doesn’t interact with all the objects. It can start the game, monitor whether the game ends and provide menu bar to the player to have a certain control over the game. At the start of the game, game class plays initializer role that is used to start the game.

- **Superclasses**

None

- **Interface**

Runnable

- **Attributes**

- **WIDTH**:

To have a fixed size of screen width at start of game.
Type: Integer,
Visibility: Private,
Initial value: 1000

- **HEIGHT**:

To have a fixed size of screen Height at start of game.
Type: Integer,
Visibility: Private,
Initial value: 1000

- **WORKING:**

To regulate the thread’s start and end. Acts like a flag.
Type: Boolean
Visibility: Private
Initial Value: FALSE

- **Time:**

Works as a time for our timer class
Type: static Ineger
Visibility: public
Intial valur: 0

- **random**

For producing random numbers from certain count
Type: Random class Object
Visibility: Private
Intial Value: None

- **SS**:

To have a private instance of an object.

Type: SunStrom Object
Visibility: Private
Intial Value: None

- **settler**:

To have a private instance of an object.
Type: Settler Object
Visibility: Private
Intial Value: None

- **Asteroid**:

To have a private instance of an object.
Type: Asteroid Object
Visibility: Private
Intial Value: None

- **Minerals**:

To have a private instance of an object.
Type: Mineral Object
Visibility: Private
Intial Value: None

- **timer**:

To have a private instance of an object.
Type: Timer Object
Visibility: Private
Intial Value: None

- **Methods**

- **void Start():** This method starts our thread

Pseudocode:

```
IF WORKING
Crate Instance of a Thread
CALL start function of class Thread
ENDIF
```

- **Void Stop():** This method stops the thread

```
SET WORKING equals to false
```

- **void StartGame():** this a method which starts the game by calling all necessary methods such as triggering tick() method.

Pseudocode:

```
IF WORKING
 CALL SetCurrentPlace of settler to set it at place
FOR 1 to 20
setAsteroids at a Place
setCore of  asteroid with random minerals
ENDFOR
ENDIF
```

- **Void EndGame():** This method ends our game. : Once the settler dies by explosion or sun storm destruction, it results in the game being over which is handled by this method. It stops all the ongoing processes and release the data to ensure that the game is in start state.
  Pseudocode:
  ```
  CALL STOP
  CALL NextRound
  ```

- **void NextLevel():** This method defines that when the player succeeds to built the spaceship and wants to continue with the next level.

- **Void Run():** This is method with the Game loop inside it to make the graphic user interface run at 60 fps

- **Void Tick():** Updates our game entities at certain rate/ticks.

- **void AddSunStorm (SunStrom ss):** This method creates a Sun Storm and add it to one of Asteroids which then will be able to destroy all objects there.
  Pseudocode:
  ```
  Intialize the Timer class object
  IF timer is less than 40 sec
     CALL SetCurrentPostion function of sunstrom to generate it at some random Place
  ENDIF
  ```

- **void RemoveSunStorm (SunStrom ss):** This method remove sunstrom object from a given position.
  Pseudocode:
  ```
  IF Timer is greater than 40 sec
        SET timer to null.
  ENDIF
  ```

8.1.11 Iron

Responsibility: Iron class is a child class of the the mineral class. Iron is considered as a resource in this game which is necessary for the user to collect to build the spaceship.

• **Super classes**: Mineral

• **Attributes** : none

• **Methods** : None

8.2 Detailed plan of testing

[The test-case defined in the chapter 7.3 should be described using the languages defined in the chapter 7.1. In this point the input data-streams - with which the functioning of program can be controlled - have to be given. For each data-stream: a definition (which parts and which functions of the program we expect to controll by executing this data-stream) and a result (what kind of results we expect) is needed. Use the syntax for both input and output as defined in the previous document.]

8.2.1 Test-case 1

- **Description**

Start game:

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

Initializing the game

- **Input**

The game gets started whenever the player chose the option from menu. The settler is created and placed in one of the asteroids

- **Output**

The game will get started

8.2.2 Test-case 2

- **Description**

Generate Starting Screen

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

Display the starting screen.

- **Input**

When the player starts the game, the first thing it sees is the starting screen in which player choses different options to start the game

- **Output**

Player will be able to see the start screen

8.2.3 Test-case 3

- **Description**

Player dies

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

Player can dies in many ways and when the player does game is finished.

- **Input**

-

- **Output**

You died

Game finished

8.2.4 Test-case 4

- **Description**

Player appearing on the screen

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

The player can be seen on the screen.

- **Input**

This happens when the game has just started nad the player is ready to play the gam**e**

- **Output**

8.2.5 Test-case 5

- **Description**

Player can be seen in space

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

When the user press start the game button on the main menu, player can see the picture of the settler on the screen

- **Input**

Player is able to move in space freely with proper input controls.

- **Output**

The picture of the settler appears

8.2.6 Test-case 6

- **Description**

Player can land on asteroid

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

Player can travel to asteroid

- **Input**

Player can easily travel to different asteroid by using proper controls.

- **Output**

Player can be seen travelling to asteroid.

8.2.7 Test-case 7

- **Description**

Player can mine

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

The goal is to mine the asteroid. Player can easily mine the asteroid and collect resources

- **Input**

Do you want to mine?
Yes, or No?

- **Output**

If yes player starts minng, if no player moves on

8.2.8 Test-case 8

- **Description**

Player can drill

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

The goal is to drill the asteroid. Player can easily drill the asteroid

- **Input**
  Do you want to Drill?
  Yes, or No?

- **Output**

If yes player starts drilling, if no player moves on

8.2.9 Test-case 9

- **Description**

Player can hide

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

Player can hide in the asteroid, if the asteroid is hollow
and fully drilled

- **Input**

Do you want to hide?
YES Or NO

- **Output**

If yes player can hide, if no player moves on. System wont allow to hide of the asterid is not drilled through

8.2.10 Test-case 10

- **Description**

Player can use minerals to build teleportation gates

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

Player can build the gates if it has enough resources
available to build one

- **Input**

Do you want to build gate?
Yes or No?

- **Output**

If yes gate is bult automatically, if no player moves on. Gate wont be allowed to built if enugh resurces are not available

8.2.11 Test-case 11

- **Description**

Player can use minerals to build robots

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

If the player has enough resource defined in the code
then the player can easily build a gate.

- **Input**
- Do you want to build robot?
- Yes or No?

- **Output**
- If yes robot is bult automatically, if no player moves on. robot wont be allowed to build if enough resources are not available

8.2.12 Test-case 12

- **Description**

Robot can hide in the hollow asteroid

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

Player can hide in the hollow asteroid if the asteroid is
fully drilled through.

Input
Described in the code, if the robot see sunstorm coming, it hides

- **Output**

The robot will hide if the asteroid is hollow

8.2.13 Test-case 13

- **Description**

Mineral’s score increases/decreases

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

It means that the player keep track of how many
different minerals its has collected so he knows how
many more does it needs to collect

- **Input**

It will be shown on the computer screen which will keep the player updated about the mierals itshas collectd

- **Output**

The counter increases or decreases

8.2.14 Test-case 14

- **Description**

Settler filling the hollow asteroid with resources

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

Settler can check if the asteroid is empty through drilling, if it is, the settler can fill it later with the resources that the settler has collected so far.

- **Input**

Do you want to fill the asteroid with resources
Yes or No?

- **Output**

If yes settler fills it, if no settler moves on. It won't be allowed the settler to fill the asteroid if the asteroid is not hollow

8.2.15 Test-case 15

- **Description**

Settler deploys the gates in the vicinity of the of the asteroid the settler is on

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

Player can deploy the gates if it has enough resources, then it can use the gate to travel from one asteroid to toher asteroid easily.

- **Input**

Do you want to deply the gate?
Yes or No?

- **Output**

If yes, the gate is deployed, If no player performs other operation

8.2.16 Test-case 16

- **Description**

Space station build successfully

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

When the player has enough resources, the player can build the space station which is the actual goal of the game

- **Input**

-

- **Output**

Congratulations, YOU WON

8.2.17 Test-case 17

- **Description**

Settler doing one operation in a single move

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

Settler can perform drilling, mining, travelling at one time only. Settlers cannot perform two operations at a single time

- **Input**

Do you want to drill, travel or mine?

- **Output**

Drill: Player drills the asteroid
Travel: Player travels to different asteroid
Mine: layer mines the asteroid

8.2.18 Test-case 18

- **Description**

Settler surviving the sun storm if hiding in the hollow asteroid

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

If there is a sun storm, the settler can first check by drilling if the asteroid is hollow, if it is then the settler can hide in it if there is a sun storm.

- **Input**

DO you want to hide?

- **Output**

Yes: player hides and survives the sun storm
No: Player doesn’t hide

8.3 Plans of the supporting programs

At this point we are only planning to use JUnit tests. JUnit testing will enforce us to validate if the code results in the expected state (state testing) or executes the expected sequence of events (behavior testing).

However, we may decide to use some program that will aid us in testing our code, but that decision will be taken later in the development of our project. A unit test targets a small unit of code and we might want to test external dependencies by replacing the dependency with a test implementation or an object created by a test framework

8.4 Protocol

Start (date & time)	**Duration (hours)**	**Performer(s) name**	**Activity description**
01.04.2021	30 minutes	All team members	Meeting to divide tasks
11.04.2021	2 hours	Salahov Kamal	8.1.1, 8.1.2, 8.1.3, 8.1.4
9.04.2021	3 hours	Gurdeep	8.1.9, 8.1.10, 8.1.11
9.04.2021	3 hours	Ali	8.1.5, 8.1.6, 8.1.9
9.04.2021	3 hours	Areeba	8.2, 8.3
9.04.2021	3 hours	Subhan	8.1.8, 8.1.7, 8.1.9