Advanced Programming

2019/20

Practical work

The assignment is the implementation in Java of the single-player game *Planet Bound*. The rules and supplementary material are available at the following address:

https://boardgamegeek.com/boardgame/298332/planet-bound

Some of that material is given in the attached file. However, you must visit that site and read the game description before reading this assignment statement. However, keep in mind that some simplifications were made to the implementation required for this assignment, as stated in this document.

General rules

The assignment must be done **individually**, without any exception.

The assignment is organized in two phases (deliverables). The delivery dates of each phase are the following:

1. First phase (model of the game using the patterns presented in class): **10th May**;
2. Second phase (functional game with GUI): **12th June**.

The deliveries corresponding to the two phases of the practical work must submitted through the **moodle** system using a compressed file in ZIP format. The name of this file must include the first name, the last name and the student number, as well as the indication of the practical class to which it belongs (ex: P3-201906104-Antonio-Ferreira.zip).

In the first phase, the ZIP file must contain at least:

- The NetBeans project, including all the source code produced;
- The report in pdf format.

In the second phase, the ZIP file must contain at least:

- The executable jar file;
- The NetBeans project, including all the source code produced;
- Any data files and auxiliary resources necessary for the execution of the program;
- The report in pdf format.

In both phases the report must include:

1. A brief description of the options and decisions taken in the implementation (maximum one page);
2. The diagram of the state machine that controls the game, and the diagrams of any other programming patterns that may have been applied at work;
3. The description of the classes used in the program (stating what they represent and their objectives);
4. The description of the relationship between classes (diagrams can be used);
5. For each feature or rule of the game, the indication of fulfilled / implemented fully or partially (specify what was effectively accomplished in this case) or not fulfilled / implemented (specify the reason). The use of a table can simplify the elaboration of this part.

Both phases are subject to presentation including the demonstration, explanation and discussion of the work presented. These may include making changes to what has been delivered.

The first and second phases of the work correspond, respectively, to **5** and **9** values out of 20.

Objectives and requirements

The objectives of the two phases of the practical work are the following:

1. **First phase**: Definition and implementation of the data model and the state machine that represents the game according to the patters given in the classes, for example, polymorphic states and object factory; **Code pertaining to user interface that may have been done in order to test the logic is not part of the evaluation of this phase, however it is advisable to include a minimum of user interaction mechanisms allowing the testing and presentation of all developed classes**;
2. **Second phase**: Complete and functional implementation of the *Planet Bound* game with a graphical user interface (GUI). This phase may be based on the elements presented and the result

(feedback) of the discussion / defense in the first phase. However, the implementation of the second phase is not conditioned by the previous presentation of the first phase. The code corresponding to the first phase can be modified in order to achieve the objectives of the second goal, in particular if it had deficiencies, and this does not mean that deliverable 1 is reassessed or its grade improved.

In the second phase, it should be possible to save the running game in an object file, as well as to load and continue a previously saved game.

For the sake of **simplification**, the following clarifications and **changes to the original rules must be** considered:

1. The game starts by choosing the type of ship to use: military or mining;
2. The game must end at the end of the turn in which the last of the five alien artifacts is found (i.e., **the Path to the Home World phase must not be implemented**);
3. Terrain cards:
    a. There are no obstacles for either drones or aliens (they are able to fly for short distances), so any movement between adjacent cells is allowed (vertically and horizontally);
    b. A 6 x 6 surface representative card is generated, in which the positions where the ship lands and where the resource is located are randomly generated, ensuring that they do not overlap;
    c. When the drone is in a cell adjacent to the alien, it is mandatory to initiate an attack, and it is not possible to escape this situation.
4. Movements of the ship through space:
    a. The Scan phase and the Space Travel cards no longer exist;
    b. In the Space Travel phase, the movement of the ship obeys the following rules: (1) between two sectors of the circle type (white or red) there is always a passage through a red dot; (2) the type of circle reached (white or red) is determined at random with the following distribution: red circles = 3/10 and white circles = 7/10.
    c. In each planetary sector the player can choose to advance or explore the planet's resources that have not yet been explored. You can only choose to explore if you have the Officer with that competence and you have the mining drone;
    d. The fact that a move was made through a wormhole is determined at random with a probability of 1/8;

e. The type of planet (black, red, blue and green) present in a sector of the red or white circle type is determined at random with equal probabilities (uniform distribution) for the four possible colors;

f. Only events 3, 4, 5, 6, 7 and 9 of the original game are considered. The new numbering is 1, 2, 3, 4, 5, 6, respectively.

When the ship reaches a red dot, in addition to the event being able to be generated randomly (rules of the game), it must also be possible to apply a specific event (e.g., there may be somewhere in the model code the methods applyEvent () and applyEvent (int idEvent)). The main objective of this mandatory addition to the rules of the game is to facilitate debugging during development and presentation.

The implementation of the work **must** obey the following requirements:

1. The patterns presented in class must be followed;

2. Anobject oriented state machine should be used in an appropriate way to concretize the logic of the game (Figure 1 illustrates, incompletely and just as an example, a possible approach, with the identification of events and actions associated with the transitions between states, considering the changes made to the original rules);

3. When finishing a game, the state machine must allow both chances: to play again, or to end the application;

4. In the Alien Attacks phase, the sequence of actions corresponding to the attack of the alien and the drone, which do not depend on any decision by the player, must be automatically triggered without any intervention from the player. The moments in the game when the player has to make decisions must correspond to states. There may be states where the player is informed about the what is happening in the game and only decides when to proceed. The existence of states in these situations depends only on the dynamics intended to the interaction with the user;

5. The application must show the information necessary to verify the proper functioning of the game (relevant data that characterize the different cards used in the game, the current turn and phase, the situations of victory and defeat, etc.). Regarding the subject of the presentation of information (user interface), attention is drawn to the fact that the screen does not have the same limitations as a physical board. This means that the interface can present the same information in a more compact or intelligent way. For example, instead of a series of squares to represent cargo hold or fuel, a numerical indicator or a pointer like that of fuel in automobiles will accomplish the same goal in a more interesting and more visually pleasing way;

6. The game model (state machine) must, through an internal log ("record") (no interaction with the user is allowed), generate and make available (note that "making available" does not mean "showing" but rather “Make accessible”) detailed information about the internal state and the result of the various actions. For example, it should be possible for the user interface to obtain

the details of what happened automatically (e.g., during the Alien Attacks phase) and present them to the user (e.g., the result of dice rolls and its consequences);

7. In both phases of the practical work, the separation between Model and View pattern studied in classes should be used.

Architecture

Figure 1 shows a possible definition of some states, missing the remaining ones that should be proposed (the states already presented can be replaced by others, as long as they make sense). The names of the actions identified in the state machine transitions in the diagram must be the same as the corresponding methods in the model. For example, based on the diagram in Figure 1, there should be the *start*, *selectShip*, *move* and *nextTurn* methods. As in this example, these names must be suggestive of the actions they represent.

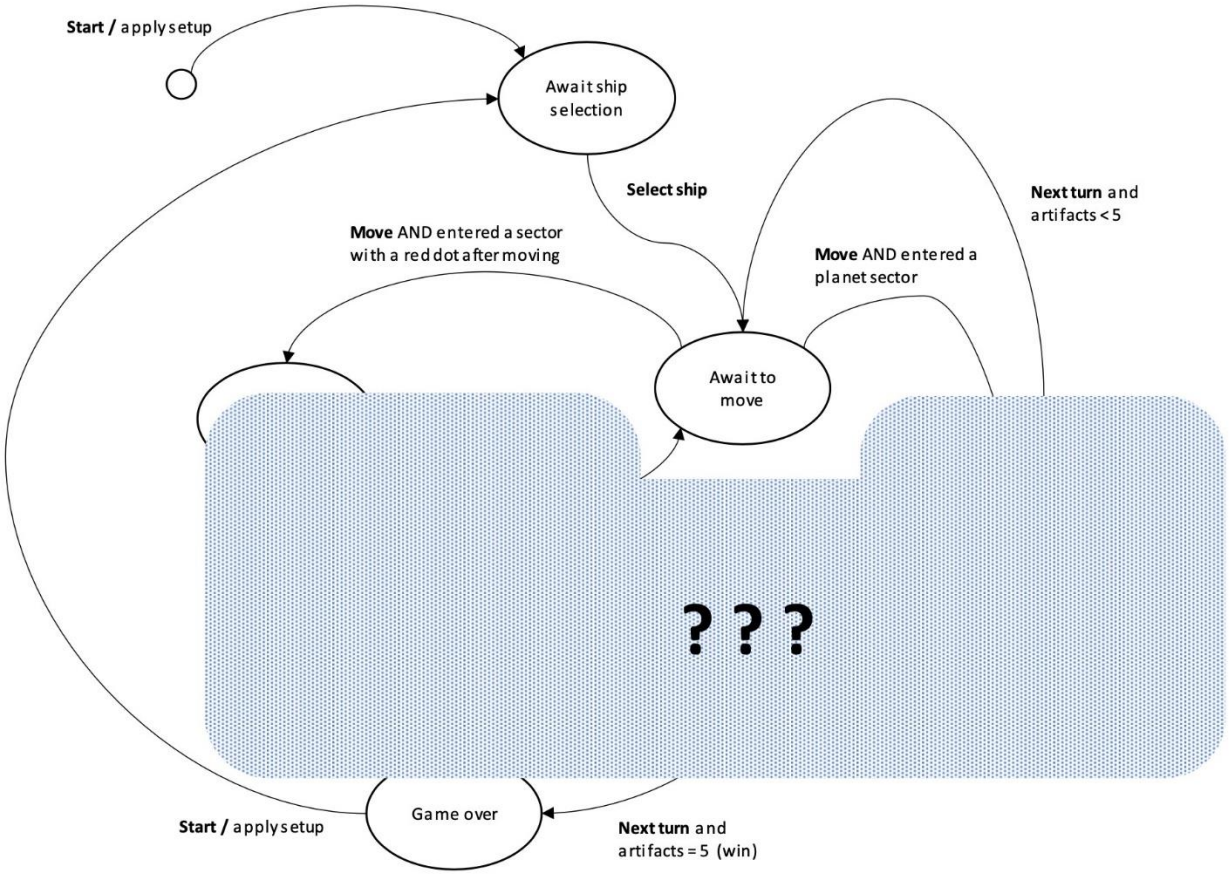

Figure 1 - Incomplete version of a possible state machine

The application must be organized in (at least) four packages: **logic**, **logic. states**, **logic.data** and **ui.gui**. The logical package corresponds to the implementation of the model / logic. The **logic.states** and **logic.data** packages will contain, respectively, the hierarchy of states according to the approach studied in class and the game data. The **logic**, **logic.states** and **logic**.**data** packages are the elements to be presented in the first phase of the practical work. The **ui.gui** package corresponds to the implementation of the user interface in JavaFx, which is presented in the second phase of the practical work. In the classes of the **ui.gui** package, there should be no logic associated with the game and its rules. On the other hand, the game logic cannot include code that performs, for example, keyboard reading or writing operations on the screen.