Project

JAVA Fundamentals

Dice Wars

---

1) General instructions

This project is to be carried out entirely in Java language.

Documentation

You can use:

- Course materials;

- Resources available on the Internet, in particular on the Oracle site or on the Scholarvox library available on myEffrei.

Modules, APIs and concepts

The realization of this project implies the use of:

- General concepts of Java (inheritance, polymorphism, encapsulation, overload, methods ...)

- Constructors, interfaces, ...

- Collections

- Threads

- Input/Output

- Persistence management

- Graphical interface: It is possible to use a graphical development tool.

Planning

- Project publication : **December 10.**
- Code review : the week of **January 04.**
    - A presentation of 10 minutes of each team progress as well as the perspectives drawn up will be panned before the submission of the final project. A preliminary score will be given.

Organizational instructions

- The project is to be carried out in teams of 3 members.
- The list of teams is to be provided to your respective teacher by **December 18** at the latest. A mallus will be applied for any delay.
- The project score will count for 30% of the final grade for the course "Java Fundamentals".

Expected results

- The projects must be returned by **January 11** at the latest.
- The deliverables must include a **.zip** or **.jar** file (be careful that the .java classes are) and a report.
- The deliverables will be submitted in dedicated deposit on Moodle.
- The report must summarize the technical design of your project as well as your justified choice of solutions.

2) Specifications

It is necessary to read the rules of the game carefully; it will help you in the design and the implementation of your program.

Game rules

The goal of this project is to provide a simplified version of the Dice Wars game. It is a strategy game in which you have to conquer all the territories of the opposing players. The general rules are:

- It is played on a map containing several territories.
- Each territory has a “strength” represented by a number of dice (6-sided).
- A territory cannot have more than 8.
- Each territory belongs to a single player.
- Each player has one or more territory (s).

Process

- Initially (at the start of the game):
    - A number of players is designated;

- The territories are allocated randomly to the different players;
- All players have the same number of dice distributed randomly on their territories.

• On each game turn, a player can make an attack. The first player to play (at the first game turn) is randomly selected.

• During his turn, the player chooses among his territories the one he wants to extend then a neighboring territory to attack.

• Territories with a single die cannot be expanded.

• The attacking player then rolls his dice (all the dice in the territory to be extended)

• The opposing player (owner of the attacked territory) on his side rolls his dice (all the dice of the attacked territory).

• The battle is done by adding the points of the dice in attack (sum of the points on the faces) and comparing them to the sum obtained from the dice in defense.

• If the attacker scores more points, he moves his dice to the conquered territory, except one that remains on the starting territory and the defeated dice of the opponent disappears.

• If the attacker loses, he only keeps one die in his territory and the attacked territory remains unchanged.

• During his turn, a player can attack as many times as he wants. His turn ends:

- If he decides to stop it himself (Ex: using an "End turn" button)
- If he has lost all his territories.

• At the end of his turn, if the player has not yet lost, he receives as a bonus as many dice as the greatest number of contiguous territories of his color.

• These reinforcement dice are distributed randomly over its territories. The image below shows a game in progress with 6 players represented by different colors.

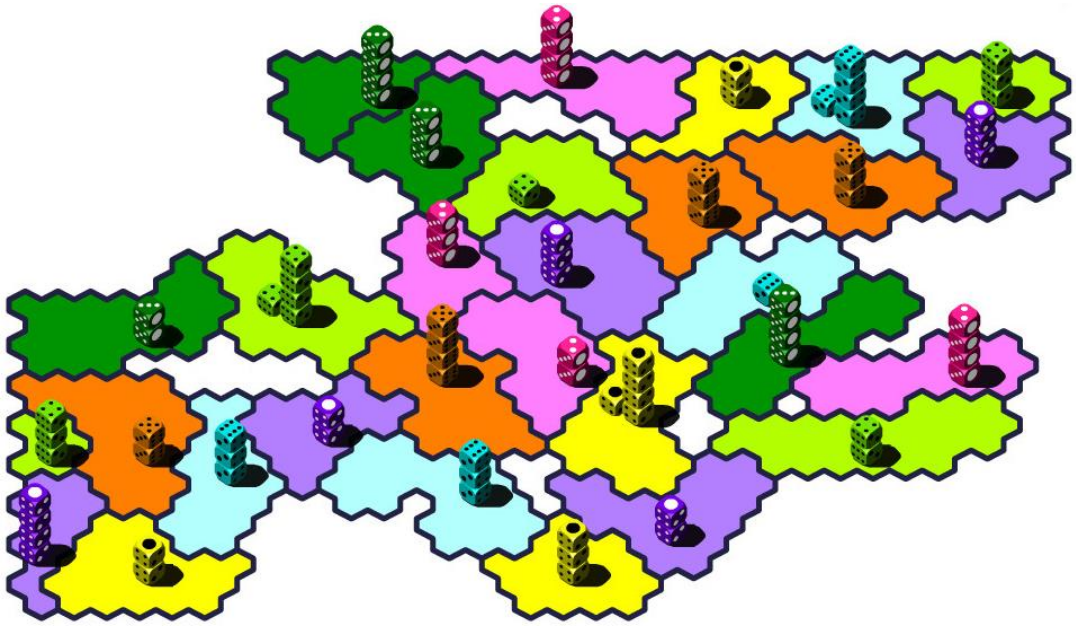

Figure 1. Example of a game in progress with 6 players

Implementation methodology

Version 1: Without graphical interface

> **Note**
>
> As a first version, we offer here a non-exhaustive list of classes and methods to implement. It is necessary to go "to the end" of this first part in order to be able to move on to the next steps. This outline is given as a general structure of your program to guide you through the process. However, you can define other classes and methods as needed.

1. Create a *`Player`* class representing a player of a game defined by an *`ID`* attribute (unique) and the *`list of his territories`* (conquered). In addition to a constructor, the player class should contain at least the following methods:

   a. *`attackTerritory()`* which is called when a player performs an attack. It must make it possible to enter the number of the attacking territory and that of the attacked territory separated by a space.

   b. *`endTurn()`* is a method the player calls to stop his attacks and pass the hand to another player.

2. Create a *Territory* class defined by a (unique) *ID*, *PlayerID* of the player who owns it, *strength* represented by the number of dice it contains and finally the *list of IDs of its neighboring territories*.

3. Create a *Map* class containing territories that can be modeled by a two-dimensional array. Provide in this class two types of constructors:
   a. A constructor for initializing the map from a CSV file.
   b. A constructor allowing the map to be initialized randomly (respecting the rules described above).

4. Create a *Game* class defined by *NB* players and which initializes a game and manages the scheduling of the game. It must be able to:
   a. Create a map and initialize it;
   b. Order the roles of the players during the game.
   c. Display the map after each turn as a matrix where each box contains the ID of the player who owns it and the number of its dice.
   d. Keep the game going as long as there is no winning player.

5. In addition, the program must also manage the following actions:
   a. The update of the dice number at the level of each territory.
   b. Updating the ID of each territory owner if necessary after each game turn.
   c. Calculation and updating of the neighbors list if necessary after each game turn.
   d. Dice roll which is an action made by the computer on a given territory dice.
   e. Calculation of scores (sum of dice) after each roll.
   f. Guarantee the uniqueness of IDs in the different classes.
   g. Make sure the number of territories is a multiple of the number of players.
   h. Allocation of reinforcement dice at the end of each turn.
   i. Random distribution of reinforcement dice on a given player's territories.

6. Use Exceptions to manage forbidden actions during a game turn, such as:
   a. Select an attacking territory that does not belong to the player.
   b. Attack with a territory containing only one die.
   c. Attack a territory that is not part of the neighborhood of the attacking territory.
   d. Attacking a territory belonging to the same attacking player.
   e. Etc..

Version 2: It's more fun with a mouse

1. In this version, you will improve your program to allow graphical display. The program must allow the interaction of several players with a game board where:
    a. Each player is identified by a color.
    b. The map will always keep the same representation: matrix of territories.
    c. Each territory is represented by a button of the player's color and displaying the number of dice.
    d. The player selects the territories by clicking on the corresponding buttons and ends his turn by clicking on the “end of turn” button
    e. The display must be updated after each attack and each turn end

2. Modify your program by adding the modules necessary for window managing, components and actions. Your design must respect the Model-View-Controller (MVC) design pattern. Your program should also handle forbidden actions.

Version 3 : Don't forget to save

In this part, you will add persistence to your application: a player must be able to stop the game at any time, save it on a persistent medium, a file in this case, then it must be possible to resume it by restoring all the objects, with all the values of their attributes but also all the relations between the objects.

1) Modify your program to enable this saving feature

2) Test this version on different scenarios

Bonus features

Face the IA

Improve your program to include simple AI controlled players with different difficulty levels.

Play online

Developed a version allowing two players to play remotely (one player per machine) using a TCP client and server. To simplify, one player will be associated with the server and the other with the client (asymmetric).