CMPT 125/126,
Spring 2009, SFU Burnaby
Instructor: Diana Cukierman

Project #2: Magni rocks in outer space game – Final description

This assignment is meant as team of up to three people work. No sharing of code should be done between different teams, no specific solution details should be shared with people from other teams or anyone outside the team except the teaching staff. If in doubt about academic honesty policies please check the course website and/or ask the instructor.

Submission deadlines:
Intermediate submission *March 30, 12:00 noon, whatever you have so far.*
Final submission: *Monday April 6, 3:00 pm via the submission server. No late submissions are possible as this is the last day of classes and a solution will be discussed then.*

A. GENERAL DESCRIPTION of the project and the game

See initial description, previously posted. The description applies completely.

B. GRADUAL IMPLEMENTATION and features to include.

1. You are recommended to start doing some planning in draft diagrams and then implementing the game with few features and then add new features in subsequent stages, as you test it. If at the end you do not submit all the features you will not receive all the points, but it will be better to have a partial solution running than code that does not execute at all. Keep backups. Save versions that run well before introducing a major change. Some features will receive bonus points, as indicated.

2. **Main features (regular points):**
   a. Deal with two worlds/boards. (You may want to start testing everything with one board only for example)
   b. Deal with at least two different types of players: kind and aggressive
   c. Implement the variation of players types with inheritance (for example creating additional classes: KindPlayer and AggressivePlayer inheriting a Player class)
   d. Include robbing of magni rocks from the other player's chest if the aggressive player reaches a cell where the other player is
   e. deal with celestial catastrophes
   f. deal with wormholes one way
   g. deal with wormholes two ways (which kills the player)

3. **Other features (bonus points)**
   a. Include a graphical display of the game (while this is optional it is highly recommended)
   b. Include the treatment of exceptions for one of the values asked to the user [this will be seen in class shortly]:
      i. try/cath
      ii. Throwing an exception

c. Have additional strategies (or types) of players. Different types of players could for example: suffer risks differently, collect magni rocks differently, kill other player, get a chance to roll the die again if previewing a non-desirable outcome, defend itself from a robbery, etc.

C. INTERACTION WITH THE USER (what to ask, what to show)

1. All the input should be done using Scanner methods

2. At the very beginning ask the user if he/she wants to play a game, allowing playing multiple games, one after the other.

3. At the beginning of each game ask the user :
   a. Initial size of boards
   b. Minimum and maximum values for magni rocks and for risk levels (the concrete values in each cell will be generated randomly within those min and max values). Validate that the minimums are not larger than the maximums.
   c. Number of wormholes to include in random positions in each board. Validate that there are no more wormholes than cells in the board (the wormholes will be positioned randomly in the boards)
   d. Data for each of the two players
      i. Name
      ii. Type (kind or aggressive)
      iii. Number of lives at the start
   e. Maximum number of turns to play (unless a player dies before that, ending the game). (We count one turn per player)
   f. You can assume that the user types in numbers when so required.
   g. You can assume a maximum and minimum value for the die (for example -6 to 6)
   h. Define a constant MAXLIVES and initialize with some value and then make sure that the number of lives is not greater than such value.

4. To ease your testing you can also have a method to initialize all the values you would ask the user and you can execute that method after you have tested the user data entry.

5. It is required to inform the user how the game is evolving, by showing to the user, after each player had a turn:
   a. how many turns already took place and how many are left,
   b. what the rolled die value was in each board, (if no die was rolled in the board indicate it with a value -100)
   c. how the boards look (their contents in each cell: magni rocks, risk level, wormholes) (you can show the board before or after a turn, indicate what the case is),
   d. any major event that took place: a player being robbed, a player entering a wormhole, a player getting trapped between two wormholes,
   e. what the state of the two players is, including:
      i. name and type (kind or aggressive)
      ii. the cell number they are currently in
      iii. the magni rocks in their chests so far,

iv. the remaining lives (or life points),
v. the path that they have traveled so far,
vi. the path should also include the history of catastrophes and the new revised position of the player if such changed due to the catastrophe

6. At the end of the game the program
   a. should show the final state of the players
   b. who the winner is

7. It is not required but it is recommended to implement some graphical display illustrating the evolution of the game only as output and additionally to the text output. If you develop such, use the DrawingPanel.java class and get inspired in the examples posted in the Examples section (see also Lab #10).

8. See the sample runs posted (follow the Assignments link) and demo in class and/or office hours. You do not have to repeat the exact words nor the exact graphical looks in the user interface, but you have to include the information shown there.

D. HINTS, SUGGESTIONS, CLARIFICATIONS

1. **(Suggestion):** Consider at least the classes Game, Player, Board, Cell (where the board has an array list composed of many cells). You can also have a class TUI: (Text User Interface) in charge of all the exchange of information with the user and using the class Game.

2. **(Suggestion)** You may want to create the new players and boards in the TUI class and pass them as parameters to the game.

3. **(Clarification)** Following the initial description, cells in the boards are numbered from #0 to #n. You can ask the user exactly the number #n (in which case the user is asked 'the highest possible cell number') or you can ask the user the 'size of the boards', in which case you are really asking the user the number (n+1) (and the last cell will be #n). As long as your program is consistent with what you ask to the user, it is ok.

4. **(Clarification)** Players collect the magni rocks and suffer risks from the cell where they land; they do not 'touch' the magni rocks nor are affected by risks in cells which are in between their previous cell and their current cell. Similarly when traversing a wormhole they only collect rocks and suffer risks from the cell where they land.

5. (Clarification) Imagine that a player is in position wx4, a catastrophe takes place in cell #1, (so the player is not affected) , but then, we need to correct his/her position; after the catastrophe the player will no longer be in wx4, but rather wx3 (it's the same cell, but different number, because the cell #1 was eliminated. so the path will be: [wx4,***,wx3] and the player did not move yet from the cell (originally numbered wx4 now wx3)

D. WHAT TO SUBMIT (electronically via submission server only). Submit one submission per team but every team member should keep a copy of all the files submitted

Intermediate submission

1. Submit whatever you have so far (scan notes if needed, or submit code) AND ALSO text file with names and student numbers of team members by March 30, at noon.

Final Submission

2. Text file with
    a. personal information OF ALL the team members
    b. Time dedicated to work adding all the time that all members spent, even if working together. You may want to distinguish the time understanding/analyzing the problem, the time implementing and debugging and the time documenting.
    c. Number of files submitted including this text file.

3. Simplified UML diagram:
    a. Each class will be represented by a box with only the class name (and the indication of whether it is an abstract class or not). The areas for instance variables and methods should be included but should be left empty in the diagram
    b. The relations between the classes should be included in this 'simplified' UML diagram

4. Documents (text or word file) with details of instance variables and methods in each class:
    c. Copy/paste the variable declarations and the method headers in a document file. You can have different files for each class or all the information in one file (with clear titles indicating the classes).
    d. This is the same information that you would include in the UML diagram, but you are asked to submit the details separately from the UML diagram and using the Java syntax (so you can easily copy/paste and have a cleaner UML diagram).

5. Top level flowchart diagram/s. (When using methods from a different class refer to that as a subroutine call in the flowchart diagram and include some indication of which class the method is defined).

6. Brief description with assumptions made (a list in point format is ok). If in doubt consult with the teaching staff.

7. Brief description (2 paragraphs or so) describing how the team work was done, who was responsible of what and how you got organized.

8. Code (with the author/s name/s, date and comments included).

9. All team members should keep a copy of all the files submitted. Keep the email that the submission server sends you confirming your submission and forward to all the team members.

E. ADDITIONAL ASSUMPTIONS

Additional assumptions can be made as long as they do not alter the description of the application. Include a description of any assumptions you make. Any major assumption should be justified, be well documented and consulted with the instructor.

End of Project #2