Concordia University

Department of Computer Science and Software Engineering
Advanced program design with C++

COMP 345 --- F2019 (Section N)
Assignment #3

Deadline: Sat. 16, 2019 by 11:55PM

Type: this is a team assignment (4 members max.)

Evaluation: 8% of the final mark

Submission: Electronic Submission through your Moodle course homepage. Create an appropriate zip archive including the entire C++ *source* files and all related deliverables.

Problem statement

This is a team assignment. It is divided into four distinct parts. Each part is about the development of a part of the topic presented as the team project. Even though it is about the development of a part of your team project, each assignment is to be developed/presented/tested separately. The description of each part portrays what are the features that the part should implement, and what you should demonstrate. Note that the following descriptions describe the baseline of the assignment, and are related to the project description. See the course web page for a full description of the team project, as well as links to the details of the game rules to be implemented.

Design Requirements (for all the four parts)
All the code developed in assignment 3 must stay in the same files as specified in assignment #1 and #2.

Part 1: Player Strategy Pattern

Using the Strategy design pattern, implement different kinds of players that make different decisions when selecting the card that gives a good and an action {`PlaceNewArmies, MoveArmies(); MoveOverLand() and MoveOverWater; BuildCity(),DestroyArmy(), AndOrAction(),Ignore(), etc.`}. The kinds of players are: (1) human player that requires user interaction to make decisions, (2) greedy computer player that focuses on building cities or destroying opponents, (3) a moderate computer player that control a region in which it just needs to occupy it with more armies than the opponents.

You must deliver a driver that demonstrates that (1) different players can be assigned different strategies that lead to different behavior using the strategy pattern; (2) the strategy adopted by a player can be changed dynamically during play, (3) the human player makes decisions according to user interaction, and computer players make decisions automatically, which are both implemented using the strategy pattern. The code for the Strategy class and its ConcreteStrategies must be implemented in a new `PlayerStrategies.cpp/PlayerStrategies.h` file duo.

Part 2: Phase Observer

Using the Observer design pattern, implement a view that displays information happening in the current turn. It should first display a header showing what player and what is currently being played, e.g. "Player 2: select the third card from the left, he would pay one coin" or "Player 3: move armies the amount of movement the card gives him " Then it should display important information related to what is happening in this turn, which should be different depending on which player turn. This should dynamically be updated as the game goes through different players and be visible at all times during game play. The Observer and Observable classes code must be implemented in a new `GameObservers.cpp/GameObservers.h` file duo (same as for Part 3).

Part 3: Game statistics Observer

Using the Observer design pattern, implement a view that displays some useful statistics about the game, the minimum being a "player #' location, and his procession (e.g. victory points, cities, continents, etc.) view" that shows using some kind of bar graph depicting what city/continent is currently being controlled by each player. This should dynamically be updated as the map state changes and be visible at all times during game play.
You must deliver a driver that demonstrates that (1) the game statistics view updates itself every time a city has been conquered by a player; (2) the game statistics updates itself when a player for example has been removed from any location by an opponent, or has built a city on the view; (3) as soon as a player owns a required number of cards, the game statistics view updates itself and displays a celebratory message with each player scoring..
The Observer and Observable classes code must be implemented in a new `GameObservers.cpp/GameObservers.h` file duo (same as for Part 2).

Part 4: Game Map Singleton

Using the Singleton design pattern, implement the structure and operation of a map for the Eight-Minute Empire game. You must deliver a driver that demonstrated that the game map contains only one instance of the class shared by game players objects. The Singleton class class code must be in the already existing `Map.cpp/Map.h` file duo, as specified in assignment #1, part #1.

Assignment submission requirements and procedure
You are expected to submit a group of C++ files implementing a solution to all the problems stated above (Part 1, 2, 3, 4). Your code must include a *driver* (i.e. a main function or a free function called by the main function) for each part that allows the marker to observe the execution of each part during the lab demonstration. Each driver should simply create the components described above and demonstrate that they behave as mentioned above.
You have to submit your assignment before midnight on the due date using through Moodle "A#3 submission Box". Late assignments are not accepted. The file submitted must be a .zip file containing all your C++ code. Do not submit other files such as the project file from your IDE. You are allowed to use any C++ programming environment as long as you can demonstrate your assignment in the labs.

Evaluation Criteria

Knowledge/correctness of game rules:	2 pts (indicator 4.1)
Compliance of solution with stated problem (see description above):	10 pts (indicator 4.4)
Modularity/simplicity/clarity of the solution:	2 pts (indicator 4.3)
Mastery of language/tools/libraries:	4 pts (indicator 5.1)
Code readability: naming conventions, clarity of code, use of comments:	2 pts (indicator 7.3)

Total 20 pts (indicator 6.4)

Reference

[1] "Game rules." Red Raven, game Design: Ryan Laukat Illustration: Ryan Laukat, Alex Davis, John Breckenridge, Malorie Laukat ©2012 Red Raven Games. https://redravengames.squarespace.com/eightminute-empire