Concordia University

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

COMP 345 --- W2018
Assignment #3

Deadline: Mar 30, 2018 by 11:55PM

Type: Individual or team of 2 students 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 an individual or a team (max 2 students) assignment. It is divided into 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 describes 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 course term project, as well as links to the details of the game rules to be implemented.

Part 1: Phase Observer

Using the *Observer* design pattern, implement a view that displays information happening in the current step. It should first display a header showing what player and what phase is currently being played, e.g. "Player 4: Conquers some regions" or "Player 1: Scores some victory coins" Then it should display important information related to what is happening in this turn, which should be different depending on what step is being played. This should dynamically be updated as the game goes through different players/steps/turns and be visible at all times during game play.

Part 2: Game Statistics Observer

Using the *Observer* design pattern, implement a view that displays some useful statistics about the game, the minimum being a "players *SmallWorld* domination view" that shows using some kind of bar graph depicting what percentage of the world 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.

Part 3: Player Strategy Pattern

Using the *Strategy* design pattern, implement different kinds of players that make different decisions during the turn for (a) picks a race and special power combo; (b) Conquers some regions; and (c) Scores some victory coins. The kinds of players are:

(1) An *aggressive* player who expand his/her empire and then hold onto everything he/she got until the end of the game, that is expand to as many regions as quickly as he/she can, the player don't fear enemy races counter attacking and destroying his/her gains.

(2) A *defensive* player constantly looking to dismantle the opponent(s) empire, that is wants a slow, well-defended advance.

(3) A *moderate* player who know when to place the race in decline, which likely ends its hold on the board, but paves the way for his/her control of another race to renew his/her conquests. (e.g. unless the player first-pick the Ghouls, whose race ability allows him/her to control them as if they were active even though they're in decline and should go in decline on the second turn, he/she'll want to spend two or three turns with him/her first race.

(4) A *random* player whose strategy reinforces random region conquers and random placing of race in decline.

Part 4: Game Statistics Observer Decorator

Using the *Observer* design pattern, implement a view that displays some useful statistics about the game as it is being played. This should dynamically be updated as various aspect of the game state changes and be visible at all times during game play. Using the *Decorator* pattern, provide different decorators that add more information to be displayed on the Basic (undecorated) Game Statistics Observer of part 2:

1. Basic (undecorated) Game Statistics Observer: Display the turn number; update the view every time a new turn starts.
2. Player Domination Observer Decorator: Display the percentage of regions owned/lost for each player, update the view when any player conquers/loses a region.
3. Player Hands Observer Decorator: Display the cards owned by every player, update the view when any player's hand is changing.
4. Victory Coins Observer Decorator: Display what player victory coins he/she scores, update the view when any more coins earned by a player.

You must be able to demonstrate that the Basic (undecorated) Game Statistics Observer is available at all times during play and updates the view when the turn number increases. Before the beginning of every turn, the user should be given the opportunity to add one or more Decorators (2, 3, or 4) above, which should result in the Observer to show more information and be notified as instructed above for each Decorator. Each Decorator (2, 3, and 4) should not be used more than once. The user should also be allowed to remove Decorators. The user should be allowed to specify that it does not want to add/remove Decorators in the future, after which the user shall not be prompted anymore at the beginning of every turn.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to each of the separate problems stated above (Part 1, 2, 3, and 4). Your code must include a *driver* (i.e. a *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 via Moodle. Late assignments are not accepted. The file submitted must be a .zip file containing all your code. 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):	12 pts (indicator 4.4)
Modularity/simplicity/clarity of the solution:	2 pts (indicator 4.3)
Proper use of language/tools/libraries:	2 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)