Concordia University
Department of Computer Science
and Software Engineering

Advanced program design with C++
COMP 345 --- Fall 2019

Team project assignment #4

Deadline:	November 30th, 2019
Evaluation:	8% of final mark
Late submission:	not accepted
Teams:	this is a team assignment

Problem statement

This is a team 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 team project, as well as links to the details of the game rules to be implemented. All the code developed in assignment 4 must stay in the same files as specified in assignment #1, #2 and #3:

- Map implementation: **`Map.cpp/Map.h`**.
- Map loader implementation: **`MapLoader.cpp/MapLoader.h`**.
- Dice rolling facility implementation: **`Dice.cpp/Dice.h`**.
- Player implementation: **`Player.cpp/Player.h`**.
- Card hand and deck implementation: **`Cards.cpp/Cards.h`**.

Part 1: New Strategies

(Extension of Part1 of Assignment 3) Add 2 more computer strategies (you should then have a total of 4 different computer strategies). These strategies must be designed so that they have a chance to win the game (e.g. the "benevolent" strategy in assignment 3 cannot win a game).

- A Random Computer Player Strategy that reinforces random a random country, attacks a random number of times a random country, and fortifies a random country, all following the standard rules for each phase,
- A Cheater Computer Player Strategy whose `reinforce()` method doubles the number of armies on all its countries, whose `attack()` method automatically conquers all the neighbors of all its countries, and whose `fortify()` method doubles the number of armies on its countries that have neighbors that belong to other players.

You must deliver a driver that demonstrates that (1) different players can be assigned different strategies that lead to different behavior for the reinforcement, attack, and fortification phases using the strategy pattern, as specified above; (2) the strategy adopted by a player can be changed dynamically during play; (3) all the computer player strategies require no user interaction, i.e. a game with only computer players runs without any interruption once the game is started. The code for the Strategy class and its ConcreteStrategies must be implemented in the **`PlayerStrategies.cpp/PlayerStrategies.h`** file duo developed in assignment 3.

Part 2: Tournament

A tournament starts with the user choosing M = 1 to 5 different maps, P = 2 to 4 different computer players strategies, G = 1 to 5 games to be played on each map, D = 10 to 50 maximum number of turns for each game. A tournament is then automatically played by playing G games on each of the M different maps between the chosen computer player strategies. In order to minimize run completion time, each game should be declared a draw after D turns. Once started, the tournament plays all the games automatically without user interaction. At the end of the tournament, a report of the results should be displayed, e.g.

M: Map1, Map2, Map3
P: Aggressive, Benevolent, Random, Cheater.
G: 4
D: 30

	Game 1	Game 2	Game 3	Game 4
Map 1	Aggressive	Random	Cheater	Cheater
Map 2	Cheater	Draw	Cheater	Aggressive
Map 3	Cheater	Aggressive	Cheater	Draw

You must deliver a driver that demonstrates that (1) when the game starts, the user is given the choice between single game mode and tournament mode; (2) when the tournament mode is chosen, the user is asked to select (2a) from 1 to 5 maps; (2b) from 2 to 4 players which can be any of the computer player strategies (no human computer should be allowed in tournament mode); (2c) the number of games (from 1 to 5) to be played on each map; (2d) the number of turns after which each game is to be considered a draw (3 to 50 turns); (3) after being started, the tournament runs without any user interaction; (4) upon completion, the results of the tournament are displayed as depicted above. This must be implemented in the **`GameEngine.cpp/GameEngine.h`** file duo introduced in assignment 2.

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). 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 using the ENCS Electronic Assignment Submission system under the category “programming assignment 4”. 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: *Mark deductions: during the presentation or code review it is found that the implementation does not follow the rules of the game of Risk.*	**2 pts (indicator 4.1)**
Compliance of solution with stated problem (see description above): *Mark deductions: during the presentation or code review, it is found that the code does not do some of which is asked in the above description. The implementation of the Observer, Adapter and Strategy patterns is not made according the pattern as described in class.*	**10 pts (indicator 4.4)**
Modularity/simplicity/clarity of the solution: *Mark deductions: some of the data members are not of pointer type; or the above indications are not followed regarding the files needed for each part.*	**2 pts (indicator 4.3)**
Mastery of language/tools/libraries: *Mark deductions: constructors, destructor, copy constructor, assignment operators not implemented or not implemented correctly; the program crashes during the presentation and the presenter is not able to right away correctly explain why.*	**4 pts (indicator 5.1)**
Code readability: naming conventions, clarity of code, use of comments: *Mark deductions: some names are meaningless, code is hard to understand, comments are absent, presence of commented-out code.*	**2 pts (indicator 7.3)**
Total	**20 pts (indicator 6.4)**