Concordia University

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

COMP 345 --- W2018
Assignment #2

Deadline: Mar 6, 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: Game start

Provide a group of C++ classes that implement a user interaction mechanism to start the game by allowing the player to:
1) Select a map from a list of map files as stored in a directory.
2) Select the number of players in the game (2-5 players).
The code should then use the map loader to load the selected and appropriate map, create all the players. Then, the code should load all the game pieces place them in the wells designed for each type of piece. Some types fit in the removable storage tray. Other components fit inside their respective wells in the main vacuum tray.

You must deliver a driver that demonstrates that 1) different valid maps can be loaded and their validity is verified (i.e. it is a connected graph, etc.), and invalid maps are rejected without the program crashing; 2) the right number of players is created, and all the game pieces are created and put in their designated area.

Part 2: Game play: startup phase

Provide a group of C++ classes that implements the startup phase following the rules of the SMALL WORLD game. This phase is composed of the following sequence:
1. Placing the Game Turn marker on the first spot of the map's Game Turn track. Where the track is used to monitor the game's progress.
2. Shuffling all the Race banners; draw five at random and lay them face up (i.e. colored side visible) in a single column.
3. Placing the remaining banners face up, in a single stack, at the bottom of the column. Do the same with the Special Power badges, shuffling them and placing one to the left of each Race banner, its round edge fitting snuggly into the banner's round opening. Stack the rest of the badges in a pile to the

left of the Race banners stack. A total of 6 Race banner and Special Power badge combos should be visible face up on the playing area, including the one on top of the stacks.

4. Placing a Lost Tribe token on each Region of the map featuring a Lost Tribe symbol. The Lost Tribes are remnants of long-forgotten civilizations that have fallen into decline but still populate some Regions at game start.
5. Put a Mountain token on each Region of the map featuring a Mountain.

Each player is then Given five "1" Victory coins, then place all remaining coins, including all "3"s, "5"sand "10"s, in a Victory stash next to the board. These coins will serve as currency during the game, and help determine the winner at the end. The game ends at the end of the turn in which the Game Turn marker reaches the last position on the track (8^{th}, 9^{th} or 10^{th} turn, depending on the map played).

You must deliver a driver that demonstrates that 1) all the game pieces have been set either on the map or on the play area according to the game set up; 2) The order of play of the players in the game is determined.

Part 3: Game play: main game loop

Provide a group of C++ classes that implements the main game loop following the rules of the SMALL WORLD game. During the main game loop, the player whose ears are the most pointed starts the game, and takes his first turn. The game then proceeds clockwise, from player to player. Once all players have had a turn, a new turn begins. The First Player moves the Game Turn marker forward one spot on the Game Turn Track, and proceeds with his next turn, followed by the others.
When the Game Turn marker reaches the last spot on the Game Turn Track, one final turn is played by all and the game ends. The player who has amassed the most Victory coins is declared the winner of the game.

- During the **First Turn** of the game, each player:

1. Picks a Race and Special Power combo
2. Conquers some Regions
3. Scores some Victory coins

- In **Following turns**, the first player moves the Game Turn marker up one spot on the track and the game continues clockwise. During his turn, each player must now either:

- Expand the reach of his race through new conquests
  or
- Put his race In Decline to select a new one.
  The player then scores Victory coins again

You must deliver a driver that demonstrates that 1) every player gets turns and that their methods `picks_race()`, `conquers()`, `scores()` are called; 2) the game ***First turn*** and **Following turns** are performed; and 3) the game ends as indicated in the game rules.
(The driver should explicitly "hard code" empty behaviors of the above methods, i.e. no real code for these methods needs to be executed in this part).

Part 4: Main game loop: Picks up Race and Special Power Combo

Provide a group of C++ classes that implement the Picks up Race and Special Power Combo, following the official rules the SMALL WORLD game.

You must deliver a driver that demonstrates that 1) The player selects one Race and Special Power combo, from among the six that are visible on the play area (including the combo made of the Race banner and Special Power badge sitting on top of the stacks at the bottom of the column). (Showing different cases); 2) the player replenishes the column of combos available to others.

Part 5: Main game loop: Conquers some regions

Provide a group of C++ classes that implement the Conquers some regions, following the official rules the SMALL WORLD game, and this is through: a) First conquest, b) Conquering a region, c) Enemy Losses & Withdrawals, d) Following Conquests; and e) Final Conquest attempt/Reinforcement Die Roll.

You must deliver a driver that demonstrates that 1) only valid conquering regions can be performed; 2) only valid reinforcement Die roll can be chosen; 3) given the sum of the die rolled, combined with the Race token(s) left in player's possession, if it is high enough to conquer the Region, the player deploys his remaining Race token(s) there. Otherwise, he deploys his remaining token(s) in one of the Regions he already occupied prior. Either way, his conquests for the turn end immediately thereafter; 4) Once a player's conquests for the turn have ended, he may freely redeploy the Race tokens he has on the board, moving them from one Region to any other Region occupied by his race (not necessarily just an adjacent or contiguous Region), provided that *at least* one Race token remains in each Region under his control.

Part 6: Main game loop: Scoring Victory Coins

Provide a group of C++ classes that implement the Scoring Victory Coins following the official rules the SMALL WORLD game.
You must deliver a driver that demonstrates that 1) the player receives 1 coin from the Victory stash for each Region his Race tokens occupy on the map. The player may also collect additional Victory coins as a result of his Race and/or Special Power benefit; 2) Also a player will likely have some tokens from another race on the map. These tokens are the remnants of an earlier race he chose to put In Decline previously.

Part 7: Main game loop: Following turns

Provide a group of C++ classes that implement the **Following turns**, according to the official rules of the SMALL WORLD game.
You must deliver a driver that demonstrates that 1) expending through new conquests with: a) ready your troops, b) conquer, and c) abandoning region; and 2) entering in decline to select new one.

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, 4, 5, 6, and 7). 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)