# Concordia University

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

## COMP 345 --- W2018
## Assignment #1

**Deadline:** Feb~~. 13~~ **16** 2018 by 11:55PM **(strict)**

**Type:** Individual Assignment

**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 five 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 the features that the part should implement are, and what you should demonstrate. Note that the following descriptions describe the baseline of the assignment, and are related to the project description [1]. 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.
Along with your submitted code, you have to explain your design. This can be, for example, as Doxygen [2] generated documentation, regular code comments, or a simple diagram. The use of test cases is not mandatory but encouraged. You are also responsible to give proper compilation and usage instructions in a README file to be included in the zip file.
A demo for about 10 minutes will take place with the marker. The demo times will be determined and announced by the markers, and students must reserve (arrange directly with the markers) a particular time slot for the demo. No demo means a zero mark for your assignment.

**Part 1: Maps**

Provide a group of C++ classes that implement a map for the Small World game. The map class must be implemented as a connected graph, where each node represents a region. Edges between nodes represent adjacency between regions. Each region can have any number of adjacent regions. Each region is owned by a player and contain a number of tokens. The map class can be used to represent any map configuration (i.e. not only the "Small World" map). You must deliver a driver that creates a map and demonstrates that the map class implements the following verifications: 1) the map is a connected graph, 2) regions are connected subgraphs and 3) each region is a node. The driver must provide test cases for various valid/invalid maps.

**Part 2: Map loader**

Provide a group of C++ classes that reads and loads a map file in the .map text file format as found in the "Small World Maps" resource, available online. The map loader must be able to read any such map. The map loader should store the map as a graph data structure (see
Part 1). The map loader should be able to read any text file (even ones that do not constitute a valid map). You must deliver a driver that reads various files and successfully creates a map object for valid map files, and rejects invalid map files.

**Part 3: Reinforcements Die**

Provide a group of C++ classes that implements a dice rolling facility to be used during the final conquest attempt. The dice rolling facility should enable the player object to decide if the sum of the die rolled, combined with the Race token(s) left in the player's possession, 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.
You must deliver a driver that creates ~~at least 2~~ a dice rolling facility objects, tests that 1) one can request ~~from 1 to 3~~ a dice being rolled, 2) that the container returns the right number of values, 3) that no value outside of the ~~1 6~~ 0-3 range is ever returned (The game contains 1 dice with three empty sides, and remaining sides are 1, 2, and 3 dots) that, 4) that every dice rolling facility object maintains a percentage of all individual values rolled up to now.

**Part 4: Player**

Provide a group of C++ classes that implement a Small World player using the following design: A player owns a collection of regions (see Part 1). A player owns the different game tokens, budges, summary sheet, fantasy race banners (see Part 5). A player has his own dice rolling facility object (see Part 3). A player must implement the following methods, which are eventually going to get called by the game driver: `picks_race(), conquers(), scores()`. You must deliver a driver that creates player objects and demonstrates that the player objects indeed have the above-mentioned features.

**Part 5: Tokens/ Fantasy Race Banners/ Badges/ pieces**

Provide a group of C++ classes that implements the game different type of tokens and coins ( e.g.: matching race tokens, victory coins); the fantasy race banners; the badges, and the game pieces( e.g. Fortresses, Mountains, Dragon…).You must deliver a driver that creates a deck of the Small World game, demonstrates that it is composed of all its elements. The driver must also include the methods that set up the game pieces.

**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, and 5). 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 Moodle under *A#1_SubmissionBox*. 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)** |

**Reference**

[1] *SMALL WORD* , Philippe Keyaerts, available on the Moodle Course homepage, or under https://ncdn1.daysofwonder.com/smallworld/en/img/sw_rules_2015_en.pdf
[2] *Doxygen* - Generate documentation from source code, available at: http://www.stack.nl/~dimitri/doxygen/.