COMP 345 - Winter 2018

Advanced Program Design with C++

Assignment 4 Report

(Part 3 and Part 4)

Rameen Rastan-Vadiveloo (27191863)

Uzair Nami (27206240)

Part 3: UML and brief design decisions

a)

As the class diagram is quite big, it is located in a separate image file in the root folder of this project (SmallWorldClassDiagram.jpg) or can be viewed online here: https://imgur.com/BGTOg9i

Note: In order to keep the diagram clean, most attributes and methods were not included in the classes, only some key methods and attributes (such as the ones used in the implementation of our design patterns), were included.

b)

Our GameLoop class can be viewed as the "body" of our program. It contains the logic that handles the game session (initializing the game as well as the main game loop). As such, it holds our main game objects (the players, the game map and the game deck) as attributes. It is for this reason that, when implementing our observer pattern, the GameLoop is our subject (and thus extends the base subject class), and anytime the GameLoop changes state, its observers are notified. Our observers (phase observer and game statistics observer), of course, extend the base Observer class, and our game statistics decorator extends our game statistics observer (and also contains the game statistics observer in order to decorate it). Our concrete decorators extend the base decorator class. For the strategy pattern, our Player class holds a Strategy object, and our abstract Strategy class is extended by our various concrete strategies (Aggressive, Defensive, Moderate and Random), as they define the implementation of the execute() method, which does not have a definition on its own in the Strategy class. Players can thus be assigned concrete strategies at runtime, which will change the behavior of the player's executeStrategy() method, based on the type of Strategy it holds. In order to keep high cohesion and low coupling, we ensured that each class in our program only has a single responsibility, and all the methods in a class are related. Also, our design patterns utilize inhertance and polymorphism by overriding the behavior of methods in order to utilize their own implementation.

Part 4 : Game Key C++ Concepts and Libraries
Note: Some of these concepts were used extensively throughout the program (such as pointers and vectors), therefore we only pointed out classes/functions where they were of critical importance, instead of listing every example of their use.

Concepts Used	**Brief Definition/Description**	**Class/Function/File Name**
pointers/smart pointers	Variables that hold a memory address as their value.	GameLoop.h GameLoop.cpp Map.cpp (also used in othr files throughout the project)
Memory management	A way to manage computer memory utilized by the C++ program (as C++ does not do this automatically via garbage collection unlike Java, it must be handled by the developer).	initializeGame() and addDecorator() in GameLoop.cpp loadMap() in MapLoader.cpp
Vectors	A sequence container class provided by STL. An indexed data structure similar to arrays, however the size is dynamic (can increase/decrease).	Map.h dfsTraversal() in Map.cpp generateRaceCombo() in GameLoop.cpp (also used in other files throughout the project)
Data structure	A structure used to store, organize and operate on a given set of data.	Map.h Map.cpp Region.h Region.cpp
Operator Overloading	Allows operators that exist natively within C++ (such as + or -) to work with objects or data types that are defined by users.	N/A
File I/O	Allows a C++ program to read and write to and from an external file. Provided natively by the <iostream> library in C++.	loadMap() in MapLoader.cpp

Exception Handling	A mechanism used to respond to and recover from exceptions (an anomaly in the program). More simply put, it is a way to handle and recover from runtime errors.	Smallworld.cpp GameLoop.cpp initializeGame() in GameLoop.cpp
Templates	A way to allow functions and classes to operate on generic types.	N/A
Any library including GUI	A library is a collection of external source code containing classes and functions that developers can import and utilize in their own program.	<iostream>, <string>, <algorithm>, <fstream>, <vector> used throughout the project