|  |
| --- |
| Mark Chilev (40031684) & Melanie Damilig (40032420)  4-11-2018 |

|  |
| --- |
|  |
| Project Description |
| COMP 345 |

# **Part 3: UML and Brief Design Decisions**

## **UML Diagram**

Refer to ClassDiagram.png

## **Design decisions**

Most methods in our application are public to allow other classes to use them. Methods that are not intended to be used by other classes and class variables are made private to lower coupling. This will allow changes to be more easily made to a class. Instead, they can be accessed and manipulated using getter and setter methods.

FantasyRaceBanner and Badge are generic classes representing a race and super power respectively. There are several subclasses (e.g. the Amazons race) that are based on them to differentiate them in terms of name, token value, and skill, but still maintain a common structure.

Similarly, Observer and Strategy are superclasses to different types of observers and strategies. Polymorphism is used by creating virtual methods in the superclass and overriding them in the subclasses.

The application has created high cohesion by breaking down classes into separate classes as much as possible. For instance, while the tournament class runs the whole game, the player class executes different turn actions such as picking a race and conquering a region.

The classes have a low interdependence, aside from where it can’t be avoided, such as knowing which player owns which region. Most interactions between modules is handled by the main driver and facilitated by abstraction.

# **Part 4: Game Key C++ Concepts and Libraries**

|  |  |  |
| --- | --- | --- |
| **Concepts used** | **Brief definition/description** | **Give: class/ function/file name including extension** |
| Pointers/smart pointers | Generic pointers used to reference objects such as a Player or an observer Subject. | Tournament in Tournament.cpp |
| Memory management | To free up memory, pointers are deleted when no longer in use. | Tournament.run() in Tournament.cpp  ~Player() in Player.cpp  ~Deck() in Deck.cpp |
| Vectors | Game pieces, players, map regions, and additional information are held in vectors. | Deck in Deck.h  Tournament.updateObserver() in Tournament.cpp  Map in Map.h |
| Data structure | Map is implemented as an interconnected graph of regions with edges, allowing for depth first traversal to check edges. | Map in Map.h  Map.linked() in Map.cpp  Map.dft() in Map.cpp |
| Operator overloading | << operator overloaded as a way to display the map as a string. | Map.<< in Map.h |
| File I/O | IO used to read map files and display list of map files. | MapReader.makeMap() in MapReader.cpp  Tournament.dir() in Tournament.cpp |
| Exception handling | Exceptions are caught and handled when users try to provide illegal input and when map files are incorrect. | Tournament.processInput() in Tournament.cpp  MapReader.makeMap() in MapReader.cpp |
| Any library including GUI | Imported the tinydir.h library as a way to quickly navigate a windows directory, to display all available maps. | Tournament.dir() in Tournament.cpp |

# **Bonus**

* Special abilities granted by races and badges are implemented.
* Maps adhere to the official board game’s maps.
* Map regions include types (e.g. forest, farmland, caverns) and tokens (e.g. mountains, troll lairs, encampments)