General Implementation

1. main: if a file for floor map is provided, that file will be read in otherwise the default floor map will be read in and generate a floor map on the screen. For the start of the game race info will be printed and the file info will be passed to Cell::play.
2. For the beginning of the game, player will be asked to chose whatever they want. And the race will be generated correspond to the choice of the player and passed to Game::init where the gameboard is initiated. After the initiation of the gameboard, the floor map with poisons, enemies and hero generated through Game::generate. Each time a valid movement happened, the board will be printed through gameNotification to controller to print.
3. Controller will read in command and make corresponding action and effects on the player. When hero tries to attack the enemy, functions like fight, check NeighbourEnemy, fight, fightEnemy will be call for a combat. And at the end, moveEnemy will be called for the purpose of attacking hero from enemys and movement on the map. Use of potion and treasure will call corresponding functions like effect. Player will be moved to next floor of the game when player approaches the stair.
4. At the end , the winning state of the game will be checked. Promote will be printed out. If the player wants to play again, a new game will be generated and procedure from 2-4 will be repeated.

Design Pattern Used

MVC:

Controller, Cell, Game, Map, Chamber

This is the basic pattern we use to design the entire project. Controller is the main class that take actions, game is the place that make actions and Cell and chamber are used to store info and make actions. Map is used to print the floor map and receive notify from controller, and this map class has a method called print, which is the actual map that the user will see.

Visitor Pattern

Since there are lots of enemy and races, it is a good practice to use visitor pattern here for the purpose of readability and maintainability. i.e. In enemy class, there is a method called move, which dealt with attacking the player and moving the enemy. We used visitor pattern to double dispatch hero and enemy, in order to act special functionalities for particular enemies on specific hero's race.

Decorator Pattern

A poison is created by decorating it use effects. There is a virtual method in potion class, which each particular potion will inherit and modify from.

Template method:
Enemy, Hero, Cell, Chamber.
The template method is extremely useful when generating enemies, poison, and treasures. (We do not want the user only generate hero or only generate stairs without enemies and items, once the game is initialized, all of the generation should happen together, means we want the user to generate everything.

Final Questions:

1. What lesson did this project teach you about developing software in teams?

   It is a hard project since the workload is pretty heavy. So we started really early and spent all our weekends on the project. After working on the project, we found that it is very important to make the functionality of the method clear and useful. With a clear documentation, i.e. the UML and pattern design, helped us a lot in coding time. However, since the habits of coding is really different between ours, we tries to code ourselves for a bit and let others play the game afterwards. For teamwork, the clearness of the code is really important. It took us a lot of time on matching our code together. However, by meeting together and cleared documentation, we solved this thing together really quickly.

2. What would you have done differently if you had the chance to start over?

   a. Think more about the design of the code before coding, try to figure out ways to make the method more useful and make high cohesion of the classes and methods.
   b. We will carefully write our documentations, because sometimes we got really confused about some functionality of methods. Good documentation really helps avoid this.
   c. We need to ensure the folder we are modifying is the same one as before. Sometimes we messed up the folder that we need to modify, since we have created several folders with the progress of coding.
   d. Creating the skeleton of the game first before adding really specific features in the game, because sometimes the features will not work if we changed the structure of the game.