Final Design Document for WATAN

Team member: c28su

Introduction

WATAN is a board game, which is similar to Settler of Catan, played by four players. It is now offering a command-line version on Linux, implemented by C++. Other versions, macOS, Windows included, are coming soon.

Overview

WATAN is constructed by four different classes: Board, Player, Game, and Subject. Game class controls the whole game and interacts with Board and Player. Player class stores everything about that player and help the Game class to keep the game going. Board class stores anything about the board and provides the Game class many functions to control the board as a whole. Subject class contains a small data point on the Board and it provides the base for Board class to control every details in the board. More details will be shown on the following paragraphs.

Game

Game has one center function called start(), which is the center function of the whole program as well. Every private function of Game is served for start(). Game class mainly store the game information except for those about board and player. Because the information about board and player is stored in different classes. The only way they can interact with each other is through Board class.

Board

Board class handles all the details in the board. Because there is only one board in this game, Board class is being object for only once. It did not store any detail information about specific points in the board, but it can access them through the public function provided by Subject class. It also provides public functions to Game class in order to let Game class deal with board data as a whole but not as separate points.

Subject

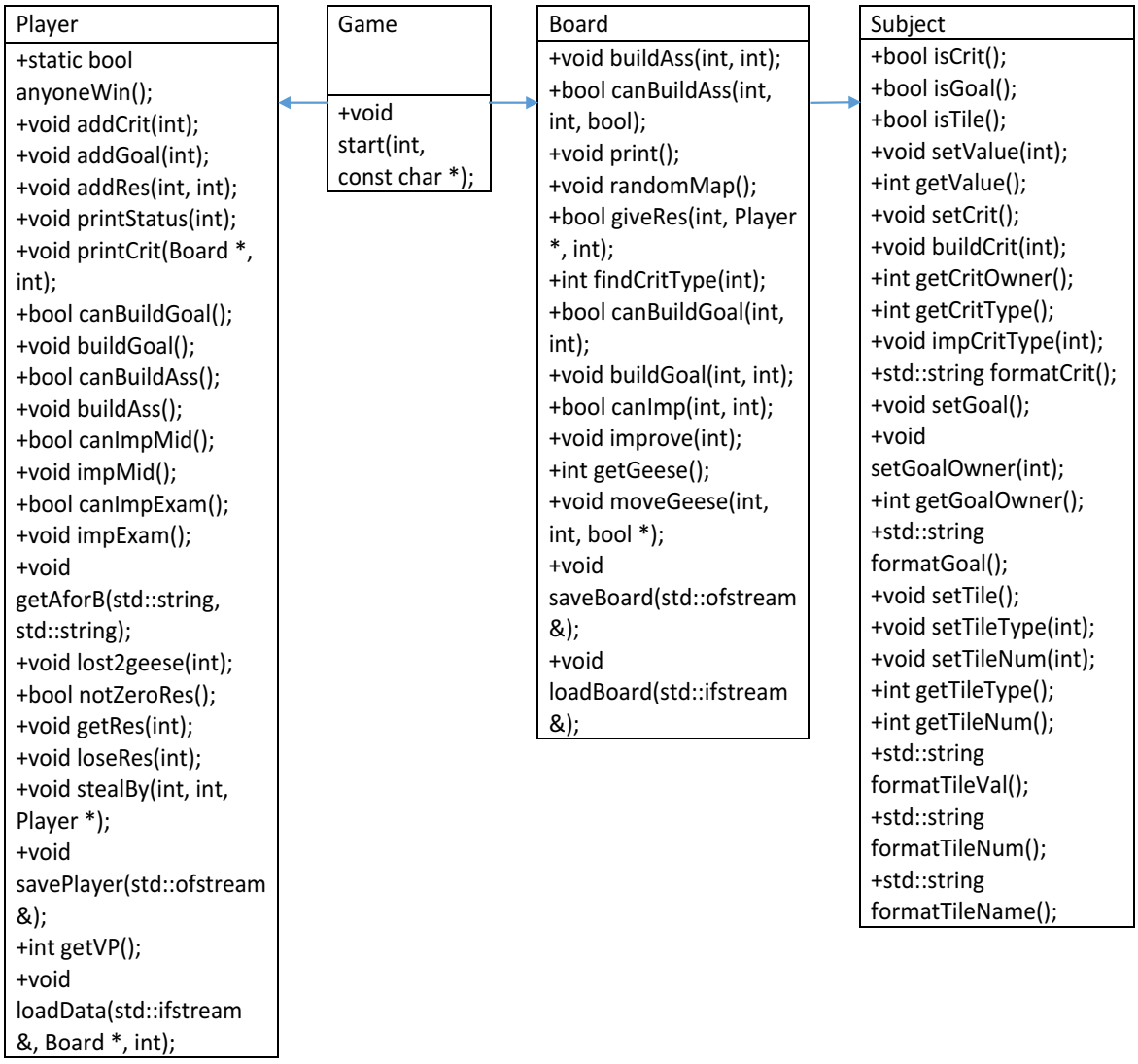
Subject class is the most basis class in the whole program. Each object of this class stores the data for one point in the board. It provides lot of public functions to Board class, resulting that Board class can deal with data in every point easily.

Player

Player class separates the data for different players. Each object of this class stores the data of exactly one player in the game. It provides some convenient public functions to Game class when those request is mainly on one player. It also provides some public functions to Game class to get and set the data about this player.

Updated UML

Final UML for WATAN



Design

I used a design pattern, which is similar to factory pattern, in Subject class to deal with 3 different types of points on the board: which is crit(criteria), goal, and tile. I considered them all as subjects, then add special feature for them in the same class, and add three bool public function to control the type when using them.

Regular hexagon is extremely hard to deal with when writing this program, especially the number of criteria, goal and tile as required by the instruction. I solved this problem in a special design of my Board class. I considered the regular hexagon as a 2-dimension array with subjects that is different type. After observation, I found that there is some mathematical relationship between each tile, goal and criteria. Much more detail is written on the constructor of Board class.

Resilience to Change

In my plan of attack, I assume that I would solve this problem basically in an observer design pattern. Why did I give it up finally? Because the separate number for tile, goal and criteria are hard to deal with. And then when I found the 2-d array can solve this regular hexagon problem, I found the i and j in the array can indicate everything I need. So I didn't use observer pattern.

In my first UML, I wrote I would use a class called Turn. But when I actually wrote that. It seems that it is useless, because Turn can not store the information about board and player, which is obviously not changed by turn.

Answers to Questions

1. I could use decorator pattern to implement, but I did not. Because the feature can be implemented by setting up two different function, which is more efficient.

2. I could Singleton Pattern for only one dice instance, and there are two functions in the dice class. I did it because I won't be worried about two dice instances if I didn't use this design pattern.

3. I would consider using observer pattern, to make all goals and criterion observe the tiles. Then even if it is square titles or any sized board, it would be easy to follow the process.

5. I would use decorator pattern to allow change of computer players. Because decorator pattern will dynamically add feature to the computer players to let them be different style computer players.

6. I would recommend factory pattern to build tiles. Because factory patterns only generalize the main structure of tiles and allow different tiles to have different specific structure.

Extra Credit Features

I checked the every player's victory points and indicated who is the winning in the end of the game.

Final Questions

1. Start as early as possible! Don't be like this one! Construct the UML carefully before started.

2. Try to find new way to deal with the regular hexagon.

Conclusion

Whatever, this is a nice software, the Windows and macOS versions will be developed by myself as well, and will be coming soon. Start early next time!!!