Rationale

Core design decisions:

I decide to have the following classes:
Board: a class to represent the game board. I decide to use a 2-D array of tiles to represent the board. Although most of the array space is not used, it will make implementation easier. Also, considering the memory size of a modern PC, the memory cost can be ignored. Also, I set the size of the board = 2*total tile number+1, make sure it will be enough space. The init tile will be placed at the middle

Tile and **TileDeck**(to store unused tile. Tile will be moved to board and removed from the deck once it is placed)

Player and **Meeple** classes will represent the two objects. The Player class has a field of the score, which will record the score.

I created a **Segment** interface (a small piece of land on the tile. Each tile should have exactly 5 segments) and several classes (Road, Field, Monastery, and City) implement the interface. This will make future extension easier.

I also created a **Feature** abstract class, with some of the methods implemented, to represent a game feature. It has 3 subclasses, road feature, game feature, and Monastery feature. The template pattern is used here. It will be easier to add new features in the future.

The design of Feature and Segment classes make feature and segment processing much easier, using polymorphism.

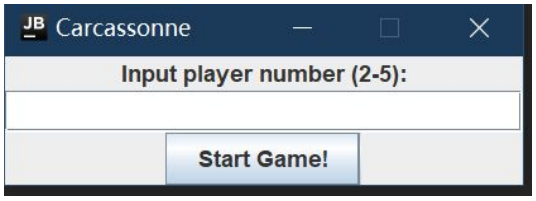
Game class, the main class and the controller of the game. I use **Game** class to control all the classes and establish communication between classes for the purpose of **low coupling** and reduce the chances of classes operating each other directly. One advantage is that the user does not need to know how the components are implemented. All users need to do is interact with the game and it will be easier to add new features (and GUI)!
Command pattern is used for feature processing and score calculating.

One major question I thought about is: who should process all the rounds and who should own the features in-game? I decided to let the game class do it. The board class should not have features since the board is only for placing the card in a location. Board class should have simple functions related to the board itself such as validate if a tile is legit at a location. The feature created does not belong to the board. Feature is a status of the game so I decide to let the game class process features

<u>GUI design decisions:</u>

In my implementation, the core does not dependent on GUI.

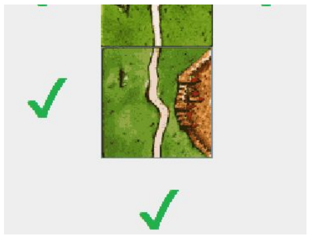
When designing the GUI, first I thought about is that the user should be able to choose the number of players from 2-5, so I decide to use a simple dialog at first to ask the user to input. The game validates the input and checks all the configuration files in the meantime (**Initwindow class**).



For the main part of the game, I choose to separate the JFrame into two panels:
A **board panel** to represent the board and a **Control panel** for user control commands and scoreboard. It a bit difficult to communicate between panels so the observer pattern is used here for GUI update and both of the panels implements the observer interface:**GameChangeListener.** This can solve the problem of interaction between panels and the core will not dependent on GUI.

The GameChangeListener has five functions, to represent the different status of a game. newRound, tilePlaced, meeplePlaced, featureFinished and gameEnds. These five can cover all events.
Both panels implement the functions and will react to the event.

One major decision is about how to set up the board to deal with user click events. First I try to get the x and y coordinates of the click events and convert to the board location. I found several problems: the click event is not accurate and the users may have no idea which area to click. Users may not sure if a tile can be placed at a location. Then I decided to make the board area a grid of buttons and make them visible when needed, like this:



Both the tile and the checkmark are buttons. I will calculate all possible locations the current tile can be placed and make that location a checkmark so the user can click on it. When a tile is placed, the appearance of the button will become the tile. It is more accurate for the users to perform click events.

The only drawback is that when the game started, it takes about 1-3 seconds to load, but it acceptable for a clicking better experience. I wrote my own TileButton class extends from JButton for better functions.

I also store a copy of the tile in the buffer so that when a meeple is removed, the tile can be recovered.

For better interaction, all clickable items in the game will provide feedback using a dialog box.

One challenge I met is the size of the panel. I decide to set the size of the panels related to the screen resolution instead of using constant values (the tile size is still constant value). Since the tile board can exceed the area, I use a scroll panel to contain the board area so the user can move around. The scroll bar will be moved to the initial tile when the game starts.

The overall result is satisfying and works well.