Rationale (Updated)

My design goals for this homework are fulfilled functional correctness of Carcassonne game, strong flexibility of my implementation and the proper reusability on different platform. Also, to make it more efficiency, I specially use data structures that not occupy too much memory.

In the design principles, I mainly focus on the information hiding, code reuse, polymorphism and extensibility. I make almost all the fields and method to be private or package private, implement many getter and setter method, also mind the defensive copy to guarantee the encapsulation idea and the unchangeability of original data. I integrate so much code into helper methods to avoid much duplicated code, and also use abstract methods to realize the code reuse. For some classes, I designed interface, like the ImageOperation class, to use polymorphism when the concrete instance initialized. I have an Enum class for four type segments and apply check completion method and score method in an abstract way, which is much easier to override —— apply different algorithms and strategies, so it is also very convenient to extend the characteristic, like adding more different type of segments.

As you can see, I designed a new way to implementing the strategy pattern, by using the Enum. To be specific, Segment class is an enum class with four types, including CITY, MONASTERY, ROAD and FIELD. Also, there would be abstract methods, including checking completion and scoring methods, whichis very easy to override by implementing different strategies, looks concise but make sense. Also, in my implementation idea, the way to check road and city is exactly the same, just by checking the edges of tiles that abut with vacancies. One of the key advantages of Strategy pattern is its extensibility, like introducing a new Strategy is as easy as writing a new class and implementing Strategy interface. But with Enum, you can create a different Enum instance without creating a different class, which means a smaller number of classes and the same benefit of strategy pattern as using abstract class or interface.