Breakdown of project:

My Straights project uses the Template Method design pattern and the Strategy Method design pattern.

In a game of Straights, we know that each round follows the same algorithm. For example, each round starts by shuffling the deck, handling out cards, and displaying who should go first. Then, while no one has ran out of cards yet, we iterate over each player, and each player makes an action and that action is handled according to the game rules. If the game rules were changed, for example, including the Joker cards, then the player's action can be different and the way that the action is handled can be different. Since the steps are the same but their implementations can change, this matches the requirements for the Template Method design pattern.

In a game of Straights, a player's strategy can change dynamically. For example, in the first couple of rounds, a player might play more conservative, then, in the later rounds, they might play more aggressive. This is reflected in implementing the Strategy design pattern where the ComputerPlayer has choice of which strategy they would like to use to execute their action on their turn.

Plan of attack:

First, I will create all of the required classes and make sure that they can communicate with each other. For example, I will create the "Display" class and its child class "ConsoleDisplay." Then, in the "Game" class, I will create an instance of "ConsoleDisplay" and make sure I can call its methods.

Once I have finished implementing this main structure and relationships between classes, I will start implementing the main functionality of the classes. I will first start with the classes that I can most easily test.

In the "ConsoleDisplay" class, I will implement each of the methods such as "displayHorCPrompt()," "displayBegOfRoundMsg()," etc. I will test that these work as I go by creating instance of these classes in "main()" and calling their methods to make sure that it is printing the correct message.

In the "Card" class, I will add class attributes to represent the card rank and suite. I will test that the game can compare if cards are equal using "operator==" by creating instances of this "Card" class in "main()".

For the "Strategy" class, I will implement "executeStrategy()" method in the child class "NormalStrategy." I will test that, given a deck of cards, this will always play the first valid move, or discard a card if there are no valid moves.

For the "GamePlayer" class, I will create its two child classes "HumanPlayer" and "ComputerPlayer." For the "getAction()" method in "HumanPlayer," I will check that it correctly prints and prompts the user for input. For the "getAction()" method in "ComputerPlayer," I will check that it correctly calls "executeStrategy()" from "NormalStrategy."

For the “Game” class, I will test that it can store both “HumanPlayer” and “ComputerPlayer” types as “GamePlayer” pointers in a vector. I will implement the “handleAction()” method in “NormalGame.”

I will first test that all the functionality of Straights has been implemented correctly. I will test this by playing as the “HumanPlayer” and by letting the algorithm play as the “ComputerPlayer.” I will test that the program does not leak memory using Valgrind. If I have extra time, I will implement a new game mode that involves the Joker cards. I will do this by adding a new child class to “Game” called “JokerGame” and adding a new child class to “Strategy” called “JokerGameStrategy.” In “JokerGame,” I will implement “handleAction” to handle the behaviour of a Joker being played.

Estimated completion dates:

- Nov 28, 2021: Create all of the required classes and make sure that they can communicate with each other
- Nov 29-Dec 3, 2021: Implement the main functionality of each class separately and test them. Integrate classes together. Test that the functionality of each class still works when they are tested together instead of separately.
- Dec 4-Dec 10, 2021: Extensively test that all the functionality of Straights has been covered and edge cases are handled.
- Dec 10-2021: More testing and potential bug fixes.

Answers to project questions:

Question: What sort of class design or design pattern should you use to structure your game classes so that changing the user interface from text-based to graphical, or changing the game rules, would have as little impact on the code as possible? Explain how your classes fit this framework

- To change the user interface from text-based to graphical, a new child class of “Display” is required. It needs to implement all of the abstract methods such as “displayHorCPrompt(),” “displayBegOfRoundMsg(),” etc.
- To change the game rules, for example, including the Joker cards, a new child class of “Strategy” and a new child class of “Game” are required. The new child class of “Strategy” needs to implement a new strategy for the “ComputerPlayer” according to these new rules. The new child class of “Game” needs to implement “handleAction” to handle moves according to the new rules.

Question: Consider that different types of computer players might also have differing play strategies, and that strategies might change as the game progresses i.e. dynamically during the play of the game. How would that affect your class structures?

- To allow the computer to change strategies as the game progresses, a new child class of "Strategy" is required. For example, one can add "PassiveStrategy" or "AggressiveStrategy" child classes. These new child classes need to implement the "executeStrategy" method to decide what moves the "ComputerPlayer" should make based on the strategy.

Question: How would your design change, if at all, if the two Jokers in a deck were added to the game as wildcards i.e. the player in possession of a Joker could choose it to take the place of any card in the game except the 7S?

- To handle the new behaviour of including the Joker cards, a new child class of "Strategy" and a new child class of "Game" are required. The new child class of "Strategy" needs to implement a new strategy for the "ComputerPlayer" according to the rules for playing the Joker. The new child class of "Game" needs to implement "handleAction" to handle the Joker card being played.