In order to change our program to play the game war, we must extend a few of our existing classes. This game is only going to run with 2 players, so we don't have to worry about a single-player AI or a 3 or 4 player game. When war is selected from the menu, our program will jump straight to the game. Once two players join a deal button will pop up giving each of the players their own deck of 52 shuffled cards. The players can click their decks to show their cards and whoever has the higher card gets a point. The winning card must be pressed for the point to go through. If the cards are the same number then we have a war and each player selects 3 cards. The players then select one of the 3 cards to face up. Whoever has the higher card gets the point. The game ends when both players are out of cards. After the game ends, the player that won shows up on the screen. Then, the users are prompted to start a new game.

When it comes to creating the menu for this game, it'll be very similar, if not the same, to what we did to create our menus and buttons in iterations 2 and 3. It should be easy to create a GameSelectMenu and edit the GameController to implement the new War game. As for the MatchController, we must use the Table and Rules interface to manipulate the game before it actually starts. So, we created a WarTable and WarRules which are classes that implement the interfaces table and rules. By creating this we are extending our existing code to create the game that we want. We can do this with other games other than War. Because the rules and table will be changed, the mainloop will run properly for the game War.

The skeleton of the new gamemode, War, consists of a deal button, along with a title change. Adding the deal button consists of extending the button class and calling the super classes constructor giving it the new buttons ID. This is identical to the pickup 52's deal button. To add the deal button to the screen we create a new menu with an apply method which adds the button to the screen. This new apply method will also adjust the screen title to say "War."

Pressing the new, War, deal button should result in the deck being evenly split into two piles, randomly. This is accomplished by adding a new class called "WarRules" and adding an apply method which will randomly split the deck in half using the Pile class.

When it is a players turn there is a boolean value, which is modified and checked using a local function, that verifies that it is a players turn to select a card from the top of their pile. If it is not their turn then their actions are ignored and no modifications are made. When a player takes a card from the top of their pile it is then put in the center pile, this is done by extending the WarTable class. There is also a pile checker that verifies that the player draws from their pile. If a user selects from a pile that is not theirs then no changes are made to the center pile, or the player pile.

When a user wins a round the PileTransfer class will allow the player to select the winning card, and then the pile in the WarTable class will add the cards to the bottom of the user's pile and clear the WarTable pile. If a player selects a winning card but is not the winner, no changes are made to either pile.

When the players show their cards for the turn, if any of the cards match it will invoke the WarTieRule class which implements the Move interface. This will cause each player that is now in the "War" tie to show the next three cards from their piles and if one player has the highest card amongst the rest it will then give them the cards. If there is another tie it will repeat this process with the players in this new tie.

When a player makes a move it will update the score depending on how many cards are in their pile. Then the score will be displayed to the user using the ShowPlayerScore class. This is similar to how the 52 game handles it's score within the P52CardMove class.

When a player runs out of cards, this will trigger table.matchOver to be true, and then will set the game title to Player X wins. This will be handled with an event that will tell the playController that there is no more moves that can be done and that player X has ran out of cards. From there, the game will be ended and the players should not be able to click on cards or buttons.

If a player clicks the cards in the middle out of turn, the cards will automatically go to the other player. To do this, there will be an event that will pass in which player should have won

that hand and which player clicked on the cards, and if they dont match up we can set the cards to go to the player that won.