**How to Play:**

To play Arimaa, there are two teams: gold and silver. To set up the game, gold player places their 16 pieces on the bottom two rows of the board in whatever way they wish. Then silver does the same on the top two rows of the board. Each player is given one elephant, one camel, two horses, two dogs, two cats, and eight rabbits. This is the order in which you will set up your pieces. The game will start with gold making the first moves. Each player gets up to four moves. A valid move is made by move in an adjacent (left, right, up, or down) direction by one placement. Rabbits are the only pieces not allowed to move backwards. Once gold’s moves are up the turn switches to silver and so forth until someone wins. There are four traps on the board. If you place a piece onto this spot or your piece is pushed/pulled onto it and you don’t have a friendly piece (a piece of the same team) adjacent to the trap, that piece will be removed. If there is an opponent’s piece adjacent to yours and that piece is stronger than yours (order of strength: rabbits, cats, dogs, horses, camels, elephants -> weakest-strongest), and there are no friendly pieces adjacent to your piece, then that piece is frozen and can’t be moved. Players can push or pull opponents pieces if they do so with a stronger piece (a cat can push/pull a rabbit, an dog can push/pull a cat, etc.). When you push an opponent, you take the spot their piece was on and push them to an adjacent spot. When you pull an opponent you make your move and pull their piece to the spot you piece originally was on. To win the game, you need to either get one of your rabbits to the opponents side, or trap all of their rabbits.

| * The enum Player holds the two types of players you can have: Gold and Silver. * The enum Piece holds the seven types of pieces your piece can have: Rabbit, cat, dog, horse, camel, and elephant. Within this, the strength of the piece is assigned based on the type of piece you have. * The enum GameState determines if the game is being set up by gold, set up by silver, the gold’s turn, or the silver’s turn |  |
| --- | --- |
| * BoardButton class extends JButton and based on what button is clicked on the board assigns it a player, piece value, the background color of the button, and x/y coordinate values. Also holds the copy, reset, setPlayer, and trapPiece methods. * The reset method removes the image, background color, player, and piece status so that it is a blank button again. * The copy method copies one button’s information onto another button. * The setPlayer method changes the color of the button depending on whose turn it is and sets the player to whomever’s turn it is. * In the trapPiece method, a given board button is set to convert its player status to none, its background to the trap color (red), its piece status to none, and its icon to nothing. |  |
| In performanceSetUp, the player and the button clicked on are passed through. If you are in the setup status, you will only be able to click on buttons within the first two rows or last two rows based on whose turn it is. The first click places the elephant, the second the camel, third the horses, fifth the dogs, seventh the cats, ninth rabbits. |  |
| In isInFreezablePosition, the board selected is checked to see if it is on the edge and check the adjacent positions to see if there is an enemy piece on it. If there is, it will set the boolean to true and continue to check the others for a friendly piece. If there is no friendly piece, the method will return true. If there is a friendly piece it will return false. |  |
| The handleFreezing method changes the status of the selected board button to frozen if the isInFreezablePosition returns true. |  |
| The isValidMove passes the player whose turn it is, the button that has been selected to be moved, and the button where that player would like to place their piece. If the piece is a rabbit, it won’t let the player move backwards. Also makes sure that the move between the selected piece and the desired direction is only one space adjacent from the initial play the piece was in. Also makes sure that the place where the player wants the piece to go is not occupied by any other pieces. |  |
| In the handleTurnSwap method, the player’s turn is switched to the next player if the number of moves they’ve made equals four or if the player clicks the “no” button when asked if they would like to make another move. When the player changes to the other, the moves are reset with the resetMoveCount method. |  |
| In the resetMoveCount method, the variable representing the moves made is reset to zero and the JLabel keeping track of the moves left is set to four. |  |
| In executeValidMove, the button spot a player wants to move is copied from the button that the piece was originally on so that it appears that the piece has moved on the game board. It also adds a point to the number of moves a player has made in their turn by calling the incrementMoveCount method. |  |
| In the incrementMoveCount method, the number of moves is increased by one and resets the text on the above JLable keeping track of the number of moves left to accurately reflect their moves left. |  |
| In the performMove method, the player and the button that is selected when clicked on. If the piece selected to be chosen is not selected (this is represented by the from BoardButton) the button selected will become that button (from == selectedSpot). Else if the their is already a piece selected, the selectedSpot button is indicating the boardButton that represents “to” or the place the player wants to move that piece. This then executes the handleFreezing method, executeValidMove method, the handleTurnSwap method, the trap method, and the recolorTrap method. However, if the piece clicked to move to is occupied by the opponent’s piece, it will trigger the pushing sequence. This starts by checking if it is possible for the piece clicked on to be pushed. If it is then it will store the opponents piece values in the pushed BoardButton and then fire the highlightPushSpot passing in the pushed BoardButton. Then the GameState is set to the pushing gold/silver (depends on whose turn it is) status indicating that a piece is getting pushed. |  |
| In the trap method, each trap is checked to see if there is a piece on it. If there are, it will check if there are any friends adjacent to it with the checkForFriends method. If there aren’t it will remove that piece with the trapPiece method. |  |
| In the checkForFriends method, a board button is passed through and checks its adjacent buttons to see if there is a piece of the same team. If there are not friendly pieces adjacent to the piece passed through it returns false. |  |
| In the recolorTrap method, if there are no pieces on one of the trap spots, it will set the trap board button to that spot on the board and execute the trap method passing through the trap board button. |  |
| In the setGameStatus method, the new gameState is passed through. If the status is the gold player’s setup, the gameStatus JLabel will say that it is gold’s turn to set up their pieces. If it is on the silver player’s setup, the gameStatus JLabel will say that it is silver’s turn to set up their pieces. If it is gold’s turn the gameStatus JLabel will say that it is golds turn (same for if it were on silvers turn). If gold wins the game, the JLabel will say that gold wins (same for silver if it were set to the silver wins status). |  |
| In the actionPerformed method, the program checks which piece was clicked on and converts the button into a BoardButton called selectedSpot which is then passed through the performSetup or performMove or performPush method depending on the state of the game. |  |
| In the handleWins method, the program checks if there is a winning setup on the board. First it will go through each button and count the number of silver and gold rabbits. If one of those counts are less than or equal to zero (there are none on the board), the other player will win. If that is not the case and there are rabbits of both team still on the board, it will check if there is a rabbit on the opponents side (a gold rabbit on silver’s first row or a silver rabbit on gold’s first row). If there is a win in any of these cases, it will set the status to gold or silver wins triggering a JLabel to display who won. Then it will call the disableButtons method. |  |
| In the disableButtons method, the program sets all the buttons on the board to be disabled through the enable(false) function. This way no players can click or move pieces anymore. |  |
| In the performPush method the player whose turn it is and the BoardButton clicked indicating where the piece is getting pushed is passed through. Then it checks if that button is an eligible position to push the piece onto by checking if the checkHighlight method is true. If so it changes the GameStatus to whichever team pushed the piece and resumes their turn. |  |
| In the removeHighlight method, all the buttons on the board are checked and if they have a blue background, it is removed. |  |
| In the checkHighlight method the BoardButton clicked to move the pushed piece to is check to see if it has a blue background (which indicates if that spot is a valid pushing spot). |  |
| In the highlightPushSpots method, the BoardButton of where a player is trying to move and push another piece is passed through. The adjacent buttons are highlighted blue if it is empty and a valid spot to get pushed to. |  |
| In the canBePushed method, the piece a player is trying to move to and push another piece off of is passed through. If there is an empty board spot adjacent to this spot and the opponent’s piece getting pushed is weaker than the player’s it will return true (the piece can be pushed). |  |