Instructions Manual/Read Me:

Run the jar file:

- Open terminal/command prompt
- type “java -jar Blokus-Game.jar”
- Hit enter

Start a game:

When launching the application you are brought to the main menu from there click “New Game” to start a new match. Choose the number of human players and computer players with the drop down menus.

To play the game, you will have to choose at least 1 human player, and one computer player. You can choose up to a total of 4 players, at least one of those must be human player. Total number of players cannot be more than 4. Once you are in the game, the first players are always the human player. For example, if you selected , 2 human players and one CPU player. Players 1 and 2 will be human players. Player 3 will be the computer.

Then you choose either easy, medium, or hard (currently easy and medium are the same difficulty in game) as the level of difficulty and hit “Start game”. This brings up the game screen.

Changing Settings:

From the Home page, click on Settings. Here you can either enable or disable hints. Hints is enabled by default. For each players turns, once a player has selected a piece, hints shows the positions on the board that the player can place the selected piece on in a dark grey color.

You can also change the theme, that determines the color of the pieces of each player. It also changes the look of some of the menus and buttons. The themes “Earth” and “rainbow” have been designed for the visually impaired and they have higher contrast between pieces in the colorblind spectrum. To go back to the main menu hit “Go back to return to the main page”.

Playing the game:

The game starts with Player 1. Clicking on the “rotate” button rotates all the pieces. Clicking on the “flip” button flips all the pieces. If hints is enabled, once the player has selected a piece, all the valid board positions are shown on the board.

Whenever a player makes a move, that player’s score is updated. If one player runs out of moves, that player’s turn is skipped. So, that player’s pieces won’t show up anymore.

If a player wants to skip a turn, they can click on “pass Turn” which lets you pass a turn.

If no player can find a move but the game is continuing (there are still valid moves available the player just doesn't see them) then once everyone gives up a user can click end game to call the game completion/win screen.

When all the players run out of moves, the game automatically ends and a pop-up window occurs that tells the name of the winner and gives the user the option to play a new game or exit the program.

At any point in the game, you can save the game with the save button. After saving the current game state will be saved and the program can be closed. After opening the program again click on the “Resume” button on the homepage to start the resume the game.

Explanation of the Code:

The program has 6 classes, namely Button, Game, home, Piece, Player, and Driver. The driver class class contains the main method that creates a Home object.

- **The Home Class:**

The Home class is an extended Jframe that has 3 Panels on top of each other. Only one panel is visible at a time. The panels are home panel, the settings panel, and the start game panel. Initially, only the home panel is visible. The home panel has the “New Game”, “Resume”, and “Settings” buttons. Clicking the “new Game” button makes the home panel invisible and the start game panel visible. The start game panel has a “Start game” button. Clicking the start game button creates a Game object.

- **The Game Class:**

A Game object is a Jframe that has three visible panels at one time. The left and the center panel are always visible, and the right panel changes depending which players turn it is.

The left panel is the score panel that displays the score of each player, the players potential moves and a “Save Game”, “Pass Turn”, and “End Game” buttons.

Whenever a player makes a turn, the score label for that player is updated through label.setText() method. Pass turn and End game call functions to change the game state.

The center panel is the game grid panel that contains Jbuttons. The Jbuttons are added using Flow layout.

There are four left panels on top of each other. The four right panels are stored in the Players[] array. Each right Panel is a Player object. A player object is an extended Jpanel Class. Each player panel contains the pieces for each player. Hence, different player panel contains different colored pieces. The Player class is explained in detail at a later stage. Only one player panel is visible at a time. When a player makes a turn, the changePlayer() method is called that makes the current player panel invisible, and the next player panel visible. The changePlayer() method then checks if the new current player panel is a CPU player by calling isComputerPlayer(int index) method with the index of the current player panel . It then calls ComputerPlays() if isComputerPlayer(int index) returns true. The ComputerPlays() method selects a valid piece and a valid placement position on the board. It sets the class's pieceSelected and BoardButtonSelected to the piece and board button that it has selected and calls the movePieceOnBoard() method. movePieceOnBoard() places a piece on the board. The movePieceOnBoard() places a piece on the board (the center panel with JButtons) by changing the colors of the corresponding JButtons of the center panel to the color of the piece.

The saving and resuming implementation for our program consists of 4 parts. The first part is saving the board. The way we placed a piece was changing the color of the buttons and so saving the board loops over all the buttons, gets the background of the current button and saves the color values for red green and blue in a text file. Resuming it reads the color line by line and sets the color back to what it was. The second part is saving the pieces and the panel where the pieces are shown to the player a used. We implemented this by looping over all the panels and writing the return of isVisible() because once the player has placed a piece the panel of that piece is set visible false.

Resuming this is also reading the file line by line and setting the panels visibility to whatever it was when saved if it was true the the piece has not been placed and gives that piece as an option if it was false then it was placed and is set back to false. The last two parts are saving the players index when saved and the difficulty and if hints were turned on or not. Resuming sets the variable back to whatever was saved. Games can be stored in one theme and loaded with another active if the user wants to change the theme mid game.

- **The Piece Class:**

he piece class has a getAllShapes() method. The method builds a 3D array. Each array element is a 7x7 2D array of numbers. The numbers stored are 0-3. Each of these 2D array represents a shape. 3 represents a colored button, 2 represents a

button that is touching a colored button sideways, 1 represents a button that's touching a colored button diagonally. 0 represents the rest of the buttons the rest. We ended up checking valid moves based on tile color so we are only using the 3 to signify a square with color.

- **The Player class:**

The player class creates 21 Piece objects which are extended JPanels and adds it to itself. It retrieves the 3-D array using the Piece class's getAllShapes() method. For building each piece, the Player class uses a for loop that loops 21 times. Each loop creates one Piece object (which is an extended JPanel), adds 21 buttons to the Piece object, access the corresponding 2D array of numbers to determine which buttons to color and colors the corresponding buttons with the color that gets passed into its constructor. It then adds the Piece Object (extended JPanel) to itself.

- **The Button class:**

The button class is an extended JButton that has x and y coordinates as instance variables. The x and y coord are set via the setXY method in the game class when the buttons are added to the center panel that form the game grid.

- **Quirks, bugs and other weirdness.**

Overall our program works pretty well throughout but we do have some bugs and a few other logical functionality errors. The game has checks in most places to stop errors, such as starting a game with too many players. But there are a few spots where errors persist and we didn't have time to work them out. In the Game class the action listener creates an error when you click on any button not in the game grid window, this doesn't stop any ingame functionality as everything still runs properly, it just prints some red errors to the consol. After saving and loading a game the "player moves remaining" tip doesn't show the current player number properly. After a players turn if the next player clicks rotate before they click any pieces then the gameboard will show hints for the previous players piece they just placed until the active player clicks a new piece. On the logical side the check for checking if a player has valid moves only checks the current orientation of pieces, not all rotations or flips. Because of this a game or turn might end prematurely when a player doesn't have any moves they can place right now but would if they rotated the piece. Currently the game allows any player to start in any corner rather than a dedicated corner per player. This works fine for four player games as each player can pick their favorite starting corner. But for less than four players it does lead to the game not functioning completely properly, after a player has claimed a

corner of the map they can also click another corner to start another branch of pieces and thus have two corners claimed. Overall we feel our code runs pretty well but we have also planned how to keep making the code even better with more time.