Introduction:

The final project is a type of Chinese chess called dark chess. It is an expansion of the midterm project, where a simplified version of Chinese chess is implemented on a 4x4 grid. This time, we implemented the complete version of Chinese chess on an 4x8 grid.

Instructions:

To play the game, you need to open the program through terminal by typing in “python” and the file name. When you open the program. You will see a 4x8 grid. There will be 32 pieces faced down on the grid colored as grey. The player will need to click on the piece to flip the piece over. The revealed color will be the player’s color. There will be a colored circle on the piece. The thicker the colored circle means the higher the hierarchy the piece is. Each turn, the player can choose to either click on a revealed piece and click on an adjacent piece of lower hierarchy to capture it, click on a revealed piece and then an empty square to move it, or to flip another unrevealed piece. To win the game, the player will need to capture all enemy pieces.

Features:

In the midterm project, we originally had the features of shuffling face-down pieces, capture pieces of different color, and moving pieces to empty spaces. The game was built in Snap. For the final project, we decided to improve the game by implementing three more features. We included an extended board from 4x4 grid to 4x8 grid, capturing pieces of different colors and according to hierarchy, and a scorekeeper that keeps count of your pieces. Also, we translated our code from Snap to Python. The board is extended from a list of 16 items to a 2D list. A hierarchy is implemented in the capture feature so that only pieces of lower hierarchy can be captured. The scorekeeper displays the number of pieces remaining for each player in the terminal and determines the winner by the end of the game.

Complexity:

An algorithm first converts the square that the player has clicked from a set of coordinates on the screen to an array that shows the row and column representing the square’s position. Then, another algorithm that incorporates multiple helper functions check if the clicked square can either be flipped over or movable by checking if the adjacent squares are blank or include pieces of lower hierarchy. When the player clicks on the piece that he or she would like to capture, another algorithm checks whether the piece is adjacent and is of lower hierarchy to be captured. All of these algorithms are composed of multiple helper functions which requires multiple weeks of work to complete.

Lists and local variables:
Each row of the board is a list. The board is list of 4 lists (2D list). The rows and the board can be found in lines 33-41.
Multiple local variables are implemented in the helper functions. In the helper function “adjacent” on line 68, local variables of x and y are implemented as they are the conversions of mouse coordinates to a position (row and column).
The helper function “neighbors” on line 85 contains a local variable called “NEIGHBOR” which is a list of all squares adjacent to the piece clicked.
The function called “Movable” on line 102 includes a variable called “MOVABLE” which is a Boolean. It is changes to true or false to see if the clicked piece can be moved.
In the function called “Endgame” on line 157, a variable called “remain” is a list containing the pieces remaining on the board.

Table:

Function	Domain	Range	Behavior
switchplayer	No argument	string	Changes the string which indicates current player
find_centercoord	List of 2 items	List of 2 items	Returns coordinates that are center of square
find_square	List of 2 items	List of 2 items	Converts mouse coordinates to position by row and column on board

square_value	List of 2 items	string	Returns value of item in row and column
piecehidden	List of 2 items	boolean	Returns whether the piece is facedown
adjacent	2 Lists of 2 items	boolean	Returns whether the desired capture piece is next to the first clicked piece
neighbors	2 lists of 2 items	1 list of 2-4 items	Return a list of neighboring pieces
movable	List of 2 items	boolean	Returns whether the piece is movable or not
legal_click1	List of 2 items	boolean	Checks if the first clicked piece during the player’s turn is a possible move
legal_click2	List of 2 items	boolean	Checks if the second clicked piece during the player’s turn is a possible move
color	List of 2 items	tuple	Returns a tuple that is identified as a color in the pygame library
thickness	List of 2 items	integer	Returns a number according to the hierarchy of the piece that is used

			to determine the thickness of the circle drawn on the piece
opponent	string	string	Returns a string that represents the opposing color
endgame	2D list	boolean	Determine whether the game has ended or not

Python:
We used python 2.7.10 for our project. We imported the library pygames for implementing graphics. We also imported a library called random to randomize the items in the list.