MSCC - NET Project : Chess

Brandon Rozario
Student no: 2955533

Divina Thomas
Student no: 2955532

Introduction :

The aim of the project is to make an android application to implement the game of chess. The board itself is contained in a custom view and has various UI elements surrounding it.

Done by Brandon :

1) onDraw()
2) onTouch()
3) onMeasure()
4) check_if_check()
5) check_before_movement()
6) check_min_number()
7) Feature : SOUND EFFECTS

Done by Divina :

8) init ()
9) reset()
10) movement()
11) check_after_movement()
12) stalemate() (Includes checkmate as well)
13) Feature : UNDO

Parts done by both Divina and Brandon:
1) UI
2) Data Structures.

Features Implemented :
1) Sound Effects
2) Undo Last Move Button

SECTION 1 : Contribution of Brandon

1. onDraw() method :

The onDraw() method is called immediately after the board is initialized.

It sets the size of each block as :

size of each block = (Width of screen) /8

It then sets the height of each block = width of each block so that each block is a square. It also sets the coordinates to draw each selected tile to approximately 90% the block size.

It then uses a two for loops to traverse the 2-dimensional String array ‘a’. This positions of text in this array correspond to the positions of coins on the board. The for-loops traverse each position of this array. If it finds a coin (for example “white_knight” or “black_queen”), it draws the appropriate coin on that particular tile.

It also draws light and dark tiles alternately, and if a tile is supposed to be selected, it draws that tile in bright green to show it is selected.

The onDraw() method is called every time a selection or change occurs on the board.

2. onTouch() method

This method is used to handle touch events on the custom view. Since the 8x8 board occupies the whole custom view, any touch in the custom view corresponds to a touch on one of the tiles on the board.

- The method first accepts the touch_down event as :
  event.getActionMasked() == MotionEvent.ACTION_DOWN
- If the mode is ‘SELECT” that means a coin is to be selected in order to be moved.
- When a coin is touched, it is checked if it it that player’s turn.
- If it is, the available moves of that coin are saved in an array.
- Those available tiles are drawn in green

- If the mode is “MOVE”, that means a coin has been already been selected, and the possible tiles it can move to are highlighted. If any of these highlighted tiles are selected, then the coin is moved to the new location.

- The ‘move’ mode also contains provisions for **promotion** (i.e. when a player’s pawn crosses the entire board and reaches the last row on the other side). The pawn is then promoted to a higher coin such as a queen, a rook, a bishop, or a knight.
- Each time a coin is moved, the game is checked to see for end conditions like checkmate or stalemate to decide if the game should carry on further or not.

TURN :

- The string variable ‘ turn’ is used to keep track of which player’s turn it is. Each time a player moves a coin, the value of turn is changed from “black” to “white” or vice-versa.
- This helps keep track of turns.
- It is important for turn to be a string variable since it helps determine if a player is moving the correct coin.
- This is done by checking the name of the coin (example white_king or black_pawn) with the current value or the variable “turn”

The onTouch method stops accepting user input from the user once the stopping condition is reached. It instead, uses a toast prompting the user to reset the game.

3.onMeasure()

This function restricts the size of the board to a square. It reads the size of the width and height of the screen and finds which is smaller of the two. It then sets the dimensions of the custom view to this smaller dimension.

4. check_if_check() method

This method accepts a two-dimensional String[][] array containing the state of the board as input. It returns true if the king in this current state of board in under check or false if the king is not under check. It does this by :

- Accepting the board as parameters
- Finding position of the king of the opposite player(not the player who just played)

- Finding the nearest neighbour of the king along the eight cardinal directions - top, top-right, right, bottom-right, bottom, bottom-left, left, and top-left. If any of these coins are of the opposite color, then they may pose a threat.
- The movement() method is called to get the possible movements of this hostile coin.
- If the king lies on the possible movements of this coin, then the king is under check.
- An Important consideration is traverse the board once again and search for knights as they do not require line-of-sight to attack.

This method will then return either ‘true’ or ‘false’.

5.check_before_movement():

This function is used to inform the player if his king is in check i.e if the black player moved a piece which put the white player’s king in check, then this function will inform the white player before he starts playing that his king is in check.
This function is called after a player moves his piece to check if the opponent’s king is in check or not.
It will pass the array containing the state of the game to the check_if_check() method, which returns a boolean value indicating whether the king is in check or not.
If the king is in check, then it sets the ‘gamestatus’ textview accordingly
Else, it will sets the gamestatus textview as ‘normal game’

6. check_min_number():

This function checks for the minimum number of pieces for a player. It stops the game if neither player has minimum number of pieces. It will scan through the board to find if the player whose turn it is to play has the minimum no of pieces. Each player should have :

1) A queen
2) A rook
3) A knight and a bishop on the dark tile or A knight and a bishop on the light tile
4) 2 knights
5) A bishop on the dark tile and a bishop on the light tile
6) A pawn or more

The function will return true indicating that the player does not have minimum pieces. You check this for both the players and increment a counter. If the counter>1, then both the players do not have minimum pieces and the state of the game will be STALEMATE and will be stopped.

FEATURE :

Sound EFFECTS:

This feature adds sounds to the game. It plays the following sounds depending on certain game events :

- a knocking sound every time a coin is moved.
- a cheer if a coin is promoted.
- a warning sound if a player's king is in check.
- applause when someone wins by a checkmate, or if stalemate occurs.

Even though this feature is not technically complex, it was added because it enhances the user experience immensely, and pretty much brings the game to life.

SECTION 2 : Contribution of Divina

1. init() :

This method sets the initial state of the game.

- It sets the stopping condition of the game to false.
- It initializes the Paint elements used in the game such as the colors : green , red, blue, black, brown, etc
- It initializes the position of coins in the the two dimensional String[][] array 'a'.
- In this array, the position of a coin in this array corresponds to the position of the coins on the board.
- For example: the white king is placed in position a[1][4] in the array since it appears in the first row, 4th column on the board.

2.reset() :

This method is used to re-initialize all variable, flags and counters whenever the game is to be reset. The game can be reset any time during the game to start over, but once a checkmate or stalemate occurs, the game stops accepting input, and the reset() method will have to be called to start the game again.To call this method, the player will have to click the button 'reset'.

After resetting all variables, it calls the init() method to reset the positions of the coins on the board.

3.public int[][] movement(int row, int col, String[][] position) :

This method will return all the possible moves for a selected coin in an array. The parameters of this method is row i.e the ith position of the selected coin and column i.e the jth position of the selected coin and string[][] position i.e it will accept the entire board as an array. You will know the position of the selected coin with the help of int row and int col.

I initialized a new array every time this function is called because it's been called by many other functions for movement of different coins. The new keyword will allocate a new memory location for the array every time you call the movement() so it does not overwrite will the old values used by other functions.

I have initialized the selected tiles to 0 so, for every call the array does not hold values of the previous piece.

After this, based on which piece is present in position[row][col], you call the switch case.
The first thing that happens when you enter a particular case, is the selected piece is highlighted at all times even if it does not have any legal moves. I have done this so the user knows which piece he has selected. Then based on the piece you do the following processing for each piece : (I have written different case for white_pawn and black_pawn as they both move in the opposite direction whereas, the rest of the coins have the same movement)

1)white_pawn : As mentioned earlier, the selected piece is highlighted. Since, the piece is white_pawn you need to keep you will have the following conditions :

Checking for the cells ahead

- If the row!=8 (If it's not the last row of the board because any movement in the last row will lead to promotion) : If true then,

a) Check if it's the 2nd row i.e (row==2), if it is then the possible moves are the next tile ahead and the next to next tile ahead. But it can only move to those tiles if it's empty. So check if the two tiles ahead is empty, if it is , then those two are the possible moves. If the first tile is empty and the second is not, then only the first tile will be the possible move. But, if the first tile is not empty and has another piece but the second tile is empty then the pawn has no possible moves in forward direction. This will be done using a for loop which will first check for the 1st tile ahead and break if it's not empty and if it is then add a value '1' to that position in the array indicating a possible move and also setting the value of can_move to true which indicates that the user has

chosen a coin and the next tile he chooses will be the tile he will want to move the piece to.

b) If it's not the 2nd row, then the only possible move in forward direction is one tile ahead. Check if that tile is empty, if it is add a value '1' to that position the array indicating a possible move and also setting the value of can_move to true which indicates that the user has chosen a coin and the next tile he chooses will be the tile he will want to move the piece to. If it has another piece, then break.

Check for opponent's piece diagonally :

It has 3 conditions :

a) If the column lies between 1 and 8 :

You check the next row and the two columns i.e one to the right and the one to the left, if it has a piece of the opponent i.e black, then the tile is a possible move and highlight it

b) If the column value is 1

You check the next row and the column to the right(as there is no column to the left), if it has a piece of the opponent i.e black, then the tile is a possible move and highlight it

c) If the column value is 8

You check the next row and the column to the left(as there is no column to the right), if it has a piece of the opponent i.e black, then the tile is a possible move and highlight it

Finally you check, if can_move is true, if it is then set the mode to 'move'.

2) black_pawn : As mentioned earlier, the selected piece is highlighted. Since, the piece is black_pawn you need to keep you will have the following conditions :

Checking for the cells ahead

a) Check if it's the 7th row i.e (row==7), if it is then the possible moves are the next tile ahead and the next to next tile ahead. But it can only move to those tiles if it's empty. So check if the two tiles ahead of the tile of the piece is empty if it is , then those two are the possible moves. If the first tile is empty and the second is not, then only the first tile will be the possible move. But, if the first tile is not empty and has another piece but the second tile is empty then the pawn has not possible moves in forward direction. This will be done using a for loop which will first check for the 1st tile ahead and break if it's not empty and if it is then add a value '1' to that position in the array indicating a possible move and also setting the value of can_move to true which indicates

that the user has chosen a coin and the next tile he chooses will be the tile he will want to move the piece to.

b) Check if it's the 1st row(row!=1)(as any movement to the 1st row will trigger promotion), then the only possible move in forward direction is one tile ahead. Check if that tile is empty, if it is add a value '1' to that position the array indicating a possible move and also setting the value of can_move to true which indicates that the user has chosen a coin and the next tile he chooses will be the tile he will want to move the piece to. If it has another piece, then break.

Check for opponent's piece diagonally :

It has 3 conditions :

a) If the column lies between 1 and 8 :

You check the previous row(as the black piece moves from 7th row to the 1st) and the two columns i.e one to the right and one to the left, if it has a piece of the opponent i.e white, then the tile is a possible move and highlight it

b) If the column value is 1 You check the previous row(as the black piece moves from 7th row to the 1st) and the column to the right(as there is no column to the left), if it has a piece of the opponent i.e white, then the tile is a possible move and highlight it

c) If the column value is 8, you check the previous row(as the black piece moves from 7th row to the 1st) and the column to the left(as there is no column to the right), if it has a piece of the opponent i.e white,then the tile is a possible move and highlight it.

3) "white_rook":

"Black_rook" : As both the white and the black rook have the same movements, I haven't made two different cases for them. Rook moves forward, backward and sidewards.

As mentioned earlier, the selected piece is highlighted. Since, the piece is elephant you need to keep you will have the following conditions :

Downward direction

Start the row iteration for 1 to 8,

a) check if you find an empty("") string in the tile ahead of it i.e in downward direction, if you do then it is a possible move hence it will be highlighted

b) Check if the selected piece is white and if it encounters the opponent's piece, then it is a possible move and will be highlighted (Vice versa)

c) If it's the same row(checking itself), then break

Right side direction

Start the col iteration for 1 to 8,

a) check if you find an empty(“”) string in the tile beside the selected tile, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same col (checking itself), then break

Forward direction

Start the row iteration for 8 to 1,

a) check if you find an empty(“”) string in the tile ahead of it i.e in forward direction, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same row(checking itself), then break

Left side direction

Start the col iteration for 1 to 8,

a) check if you find an empty(“”) string in the tile beside the selected tile, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same col (checking itself), then break

4) white_knight :

Black_knight : As both the white and the black knight have the same movements, I haven’t made two different cases for them.

As mentioned earlier, the selected piece is highlighted. Since, the piece is knight you need to keep you will have the following conditions :

If the absolute difference between one of the coordinates (either row or column) of the position of the knight with the loop traversing the board is 1 and the other difference between the other coordinate and the loop is 2, that position is a movable position for the knight, unless another of the same player’s coin already exists there.

5) "white_bishop_light":
"white_bishop_dark":
"black_bishop_light":
"black_bishop_dark": As both the white and the black bishop have the same movements, I haven't made two different cases for them. Bishops moves in diagonal direction depends on the tile it is in.
As mentioned earlier, the selected piece is highlighted. Since, the piece is bishop you need to keep you will have the following conditions :
int i = row; (row position of the selected bishop)
boolean stop_left = true, stop_right = true;
int jminus = col, jplus = col; (So it goes to the right and left)

a) Start iterating from row to 8 i.e downward direction

Increment jpus(So it goes to the next right tile) and decrement jminus(So it goes to the previous left tile). Start iterating for the left tile

- check if you find an empty("") string at that position, if you do then it is a possible move hence it will be highlighted
- Check if the selected piece is white and if it encounters the opponent's piece, then it is a possible move and will be highlighted (Vice versa)
- Check if the selected piece is white and if it encounters its own piece, then it is not a possible move and the loop should break by setting stop_left to false.

The same logic is used to iterate to the right tile in downward direction.
Initialize all the i to col and all the other variables and repeat the same logic to iterate to the left and right tile in upward direction.

6) "white_king":
"Black_king": As both the white and the black king have the same movements, I haven't made two different cases for them. The King can move in any direction but only 1 tile.
As mentioned earlier, the selected piece is highlighted. Since, the piece is king you need to keep you will have the following conditions :

a) Check if the king is not present in the corners i.e if (col != 1 && col != 8 && row != 1 && row != 8, if true, then check in all direction only once and break:

- check if you find an empty("") string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent's piece, then it is a possible move and will be highlighted (Vice versa)

b) Check if the king is present in the 1st row and 1st col i.e if (row == 1 && col == 1), if true then check in 3 direction i.e(1st row-2nd col & 2nd row-1st col & 2nd row-2nd col)only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

c) Check if the king is present in the 1st row and 8th col i.e if (row == 1 && col == 8), if true then check in in 3 direction i.e(1st row-7th col & 2nd row-7th col & 2nd row-8th col) only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)

d) Check if the king is present in the 1st row and between 1st and 8th col i.e if (row == 1 && col > 1 && col < 8), if true then check in 5 direction only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

e) Check if the king is present in the 8th row and the 1st col i.e if (row == 8 && col == 1), if true then check in in 3 direction i.e(8th row-2nd col & 7th row-1st col & 7th row-2nd col) only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

f) Check if the king is present in the 8th row and the 8th col i.e if (row == 8 && col == 8), if true then check in in 3 direction i.e(8th row-7th col & 7th row-7th col & 7th row-8th col) only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

g) Check if the king is present in the 1st col and between 1st and 8th row i.e if (row > 1 && row < 8 && col == 1), if true then check in 5 direction only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

h) Check if the king is present in the 8th col and between 1st and 8th row i.e if (row > 1 && row < 8 && col == 8), if true then check in 5 direction only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

i) Check if the king is present in the 8th row and between 1st and 8th col i.e if (row == 8 && col > 1 && col < 8), if true then check in 5 direction only once and break:

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted.
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)

7) white_queen :

Black_queen : As both the white and the black queen have the same movements, I haven’t made two different cases for them. The queen can move in any direction. As mentioned earlier, the selected piece is highlighted. The forward, backward and sidewards movement are same as that of the rook and the diagonal movement are same as that of the bishop. Since, the piece is queen you need to keep you will have the following conditions :

<u>Downward direction</u>

Start the row iteration for 1 to 8,

a) check if you find an empty(“”) string in the tile ahead of it i.e in downward direction, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same row(checking itself), then break

<u>Right side direction</u>

Start the col iteration for 1 to 8,

a) check if you find an empty(“”) string in the tile beside the selected tile, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponen’ts piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same col (checking itself), then break

Forward direction

Start the row iteration for 8 to 1,

a) check if you find an empty(“”) string in the tile ahead of it i.e in forward direction, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponent's piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same row(checking itself), then break

Left side direction

Start the col iteration for 1 to 8,

a) check if you find an empty(“”) string in the tile beside the selected tile, if you do then it is a possible move hence it will be highlighted
b) Check if the selected piece is white and if it encounters the opponents piece, then it is a possible move and will be highlighted (Vice versa)
c) If it’s the same col (checking itself), then break

Diagonal

int i = row; (row position of the selected bishop)
boolean stop_left = true, stop_right = true;
int jminus = col, jplus = col; (So it goes to the right and left)

a) Start iterating from row to 8 i.e downward direction

Increment jpus(So it goes to the next right tile) and decrement jminus(So it goes to the previous left tile). Start iterating for the left tile

- check if you find an empty(“”) string at that position, if you do then it is a possible move hence it will be highlighted
- Check if the selected piece is white and if it encounters the opponent’s piece, then it is a possible move and will be highlighted (Vice versa)
- Check if the selected piece is white and if it encounters its own piece, then it is not a possible move and the loop should break by setting stop_left to false.

The same logic if used to iterate to the right tile in downward direction.
Initialize all the i to col and all the other variables and repeat the same logic to iterate to the left and right tile in upward direction.

Finally, return the selected array, which has all the possible movements of the selected piece.

If the any possible path is found you add a value ‘1’ to that position in the array indicating a possible move and also setting the value of can_move to true which indicates that the user has chosen a coin and the next tile he chooses will be the tile he will want to move the piece to.Finally you check, if can_move is true, if it is then set the mode to ‘move’.This function will return the array consisting of all the possible moves of the selected coin.

4.check_after_movement(int cell_i,int cell_j) :

This function is responsible to remove the invalid moves from the available moves in the following manner :
The parameters of the function is the position of the piece you want to check.
The temp_selected_array will have all the available movements of the selected piece.
I placed the piece to first the available position, passed the array to check_if_check(), which will return a boolean value stating if the king is in check or not. If the king is in check, then it is not a legal and valid move, hence remove that position from temp_selected_array. If the king is not in check which means the move is valid hence, I moved the piece to its original position.
I repeated this for all the available moves and finally, the temp_selected_array will have only legal moves in it removing the invalid moves.

5.stalemate() :

This function will change the state of the game if it is in stalemate or checkmate. You iterate through all the coins i.e if its white’s turn then iterate through the entire board for all the white coins. Once you find a white piece, pass the position of the white coin to movement() which will then return all the possible moves of that piece. After you get the possible moves, pass the position of the white piece to check_after_movement()(which will then call check_if_check(),which will check for the legal moves and remove all the moves that will compromise the king i.e put it in check). After this you check the temp_selected_tiles if there are any possible legal move, if there are then increment the counter.

- If the counter==0 && answer is false(the king is not in check), then there are no legal moves, hence its a stalemate.
- If the counter==0 && answer is true(the king is in check), then there are no legal moves, hence its a checkmate.

6. FEATURE : UNDO LAST MOVE BUTTON :

This feature allows the player whose turn it is to move, to undo the move they just made.

For example : if it is white's turn to move, and he moves a pawn forward, by clicking the undo button at the bottom on the screen, the pawn moves back to its previous position. In order to maintain fairness the undo button has the following two characteristics:

(a) only the last move made can be undone
(b) once a coin is moved, if any other tile is touched, the undo button is disabled. i.e. the undo button will work only if a player moves to a new position and immediately after that presses the undo button. If the opposite player starts playing, or any other tile is touched (even if it is blank), the undo button is disabled.
The undo button works by saving the current state of the board in the application's shared preferences for persistent storage. When the UNDO LAST MOVE button is clicked, getSP() method is called and the shared preferences from the last move are read back into the String[][] 'a' and the old board is redrawn.

It consists of two methods :

sp() method :
This method is called every time a movement is to be made.
It writes the current state of the board to persistent memory.
Writing is done using the .putString(key, value) method where 'key' is the unique key for each entry in the sharedpreference.

getSP() method :
This method is called when the UNDO LAST MOVE button is clicked. However, it works only when the player has made a move AND not touched any other tile. This gives the player only one chance to go back, and not once the other player has started to play.
Each value is read out using the .getString(key) method. The key is specified, and the corresponding value is read back into the String array 'a' that represents the coins on the board.

Before UNDO :White King is in CHECK After UNDO:Queen is in previous position

SECTION 3: PARTS DONE BY BOTH DIVINA AND BRANDON:

1. UI :

The UI is based on the previous assignment : Minesweeper. The board has been modified to remove the white spaces between tiles. Tiles are alternately light and dark beginning with a light colored tiled in the top-left corner and ending with a light colored tile in the bottom-right corner.
The colors used for the buttons and textview are the primary colors for the application to maintain a theme throughout the application.

The application begins with two text-views. The first one displays the status of the game. It displays ‘Normal Game’ indicating the game commences under normal circumstances. It changes to display the following :

- ‘Check’ - if a player’s king is under check
- ‘Checkmate’ - if a player has won
- Stalemate - if stalemate has occurred
- Normal Game - if the game is continuing normally

The second textview displays who’s turn it is. It displays either “WHITE’S TURN” or “BLACK’S TURN”. It always starts with WHITE’S TURN.

Below the textviews sits the board. The board an 8x8 grid where the color of each tile alternates between light and dark color. The board is forced to be a square to keep the size of each tile the same. Hence the height of the board equals the screen width. Each player has a set of 16 coins each. The coins are :

- 8 pawns
- 2 rooks
- 2 knights
- 2 bishops
- A King
- A Queen

Each coin has a set of rules governing their movement.
Below the board are two buttons. They are both kept at the bottom of the screen under the assumption that both players are going to be sitting beside each other as opposed to across each other since this is how we usually play sitting at college desks. It will be easy for both players to reach the ‘UNDO LAST MOVE’ button and the ‘RESET’ button.

The application is forced to stay in portrait orientation to maintain the best user experience as the board tends to get a lot smaller in landscape mode on a phone with a 16:9 aspect ratio.

2. DATA STRUCTURES :

BOOLEAN :
These variables have only 2 states and are hence stored in boolean variables to save memory

bombed	True : endgame condition is satisfied. onTouch() stops accepting input False: game has not ended yet. Players can continue playing.
answer	TRUE if the king is is check. False : King is not in check
gamestarted	TRUE : a player has moved a coin, and he can undo the move. FALSE: if a player can no longer UNDO his move.
color	TRUE: draw a light tile FALSE : draw a dark tile.

INTEGER : Basically, used when the value is not critical and also for counters.

pop_counter	It will check if a player has moved his piece and if it has, then change the turn. It is re-initialized to the previous value during undo and reset
width	Sets the width of the custom view

height	Sets the height of the custom view
cell_i	Corresponds to the row of the cell touched by the user
cell_j	Corresponds to the column of the cell touched by the user

Double Dimensional Array :
They are used to save the entire board so you can iterate through it and perform some actions on a particular value at a particular position.

int [][] temp_selected_tiles	Consists of all the available moves of the selected piece. It stores the moves by inserting a value ‘1’ in the position of the available path.
String[][] a	It stores the entire board after every movement. String is used as it stores the coins of the chess
String[][] temp_a	It stores the entire board and is used for processing.

String :
The text in these variables are important to the functioning of the game. They occupy the largest size, and are hence sparingly used.

String turn	Used to mark which player’s turn it is to play. It contains “white” if it is the white player’s turn, or “black” if it is the black player’s turn. It is also useful to check if a player selects his coins and not the opponents by running the check : if (selected_coin.contains(turn)) Where selected coin is a string such as “white_knight” or “black_knight”
String mode	Contains either : “select” (mode where a player selects a coin he wants to move) or “Move” (mode where a player moves the selected coin to a new location)

Acknowledgements:

*The images for the coins are taken from the from Wikimedia Commons, the free media repository available online at :
https://commons.wikimedia.org/wiki/Category:PNG_chess_pieces/Standard_transparent

*Sounds for promotion and end-game checkmate is taken from soundbible available online at: *http://soundbible.com/1823-Winning-Triumphal-Fanfare.html*
http://soundbible.com/2087-Audience-Applause.html

*Sound for coin placement is a recording of me knocking hard on my laptop.

*Sound for check is a Windows error message.