Peel District School Board

Stephen Lewis Secondary School | 3675 Thomas Street, Mississauga, ON

Culminating Project – ChessBasic

**BY:** Roshan Munjal

**TEACHER:** Ms. Megson

**Course:** ICS 3U0

Timeline: ChessBasic Program

|  |  |
| --- | --- |
| Day(s) | Task(s) Completed |
| May 16th, 2016 | * The problem definition and proposal section of the project was completed. * It included information such as: name of the program, reasons for creating the program, a description of the game, rules of the modified version of chess, checklist of features of the program, and a rough timeline for when tasks would be completed. |
| May 17th, 2016 | * The problem definition and proposal section was handed in. * The problem analysis and design section was started. * On this day, the screen images for the output screen were drawn to show how each screen in the final product would look. * Also, a description of each output screen was written. |
| May 18th, 2016 | * Now, the major components of code were written – these include the chess program logic, in the form of a flow chart, chess piece logic for each individual piece, a draft of code regarding a move checker, etc. |
| May 19th, 2016 | * The problem analysis and design section was handed in and programming the game began. |
| May 20th, 2016 | * The problem analysis and design section was approved and the coding for the program could begin. * Firstly, I imported the images of all the pieces needed for my specific variation of chess (i.e. white king, white pawn, white rook, black king, black pawn, and black rook). * Then, I started to draw the chess board itself, with all the squares. |
| May 21st, 2016 | * The drawing of the chess board was completed and pictures were properly placed on the screen. |
| May 23rd, 2016 | * The programming for the logic began. I started by creating a 2 dimensional array to represent the starting position of all the pieces on the board. This gave me a way to represent each piece on the board. * Also, I was thinking about how each piece on the board would be linked to that array and moved each time the player clicked on a piece. |
| May 24th, 2016 | * In the paint method of my program, I now changed it to paint each piece depending on its location on the board. Instead of just drawing the original setup, now the chess board screen changed as the 2D array did. * I also began to create a mouse click feature that would identify the location where it was clicked. |
| May 25th, 2016 | * Now, I started to refine the mouse click and added a method to identify where the mouse was clicked in terms of a location on the board. This would allow me to reference it to my main 2D array regarding the chess board. * I began to start programming how pieces would move depending on the locations that were clicked. |
| May 26th, 2016 | * A certain integer variable, called mouseWasClicked, helped me determine if nothing had been clicked yet (-1), if a piece was selected (0), or if a piece had been selected and dropped (1). * Also, I began to create a movePiece method, which did not turn out to be useful because the logic of moving the piece could easily have been put into the mouse click method. * I faced many bugs at this point in the program, as the location of the piece returned was not necessarily the correct location, and was getting difficult to fix. * However, with great help, I managed to create a program that allowed a user to pick and place a piece on the board. |
| May 27th, 2016 | * I started to make a method to check the available moves of a certain piece in particular situations. * This was a general method that linked to the valid move checkers of different pieces depending on which piece was clicked in the first place. * I started to make a method to check the possible moves of the white pawn in any situation. Since then, it started to highlight the moves that were possible, but did not force the user to go to a highlighted square. * NOTE: Error in pawn movement found – when pawn is on the other side of the board, it cannot be clicked on. |
| May 28th, 2016 | * The pawn movements were improved on and move checkers for the rooks and partially for the kings were developed. * They are able to move, but there is no forced sequence that players must follow in the games – i.e. black can move first, any player can move more than once, the pieces can move anywhere, etc. * NOTE: Error in pawn movement still present. |
| May 29th, 2016 | * Documentation for the code was updated, and code was formatted to make it easier to read and understand. It was also separated into 2 sections: the applet section and the logic section. This segregation provided ease for me in particular, helping me navigate my code. * The logic behind the king’s available moves was further developed, but has not been completely finished. * NOTE: Error in pawn movement still present. |
| May 30th, 2016 | * Code was more neatly organized – lines of code were reduced and were made more efficient. * NOTE: Error in pawn movement still present. |
| May 31st, 2016 | * Finished the movements for the king’s valid moves, so that it could move in the diagonal positions as well. * NOTE: Error in pawn movement still present. |
| June 1st, 2016 | * Fixed the pawn movement so that when a pawn reaches the opposite end, it automatically turns into a rook of that colour. This was because the only major piece (except the king, of course) that it was able to turn into was the rook. * I also made it so that in the chess game, the players had turns – in the first turn, only the white pieces can move; in the second turn, only the black pieces can move; etc. Also, if the piece is clicked on and dropped in the same position, then it is still the same player’s move. * Furthermore, I documented all of the code that needed to be documented and explained my logic, all of the methods, etc. |
| June 2nd, 2016 | * I improved upon the game so that the title (ChessBasic) was displayed on the home screen. * Also, I made it so that there was a square on the side that indicated which player’s turn it was to move. * Furthermore, at this point in the timeline, I decided that it would be too difficult to produce an algorithm to determine whether the king was in check at any point in time. This would have taken lots of time to complete, and due to time constraints, it was decided against after discussion with the teacher. * An algorithm to determine check and checkmate is very difficult to create because the algorithm has to check for all moves that can attack the king in any way. Also, if the piece checking the king can be captured by another piece to remove the check, then those possibilities must also be calculated. And checkmate is a combination of looking for check in all the possible moves that the king has in its current position. |
| June 3rd, 2016 | * Added buttons to the side of the chess board to make it so that there were three options – Main Menu, Rules of the Game, and Results. * I will later develop the game so that these options can be displayed as separate screens. * I also altered the game so that when a player wins, then, underneath the chess board, it will display “BLACK WINS!” or “WHITE WINS!” Additionally, when a player wins, then the game cannot continue from that point – no pieces can be selected. Eventually, to restart the game, I will make it so that the user has to go back to the main menu and click begin (again) to start the game over. |
| June 4th, 2016 | * At this point, I have started to make some of the screens for the chess game. * For instance, the main menu has been made. Also, some of the buttons needed for future use and current use have been made as to put some graphics on the screen. * This involved a lot of guess and check with placing text and some images on the screen, and testing to see which colours would look best on the screens. |
| June 5th, 2016 | * Some more screens are starting to be developed. * The mouseClicked method and paint method were heavily improved upon to introduce the capability of changing screens. |
| June 6th, 2016 | * I started to create the results page and more buttons for use in the pages. The rules of the game page was also created. |
| June 7th, 2016 | * Several tasks were accomplished today. * All buttons were put into place so that the user could move about all the pages just by entering the home page and going around as necessary. * All 4 screens (Main Menu, Game Screen, Rules of the Game, Results) were updated with images and text to fulfill their purpose. * For the results section, I got the number of times that white and black have each won to be stored in a separate file called “Win Count for White and Black.txt”. This file was accessed using the I/O (input/output) methods provided by the Java libraries. These features helped me in the process. * **Citation:** I consulted both a student (Omar Ashqar) and the teacher (Ms. Megson) and received some help in trying to understand how the I/O methods worked.   <https://www.youtube.com/watch?v=1ZODyCD-1hU>   * For the rules of the game section, I managed to display all the rules pertaining to my modified chess game. These rules were not exact, as there was not much room to elaborate, but the basics of the game were conveyed. * All the buttons were tested to be linked correctly. * For the chess game itself, if a player then clicked on a piece and clicked on another square that was NOT a valid move for the piece, then the piece would be unselected. If the square to which it was moved was a valid move, then the piece would move and the last move would be highlighted by the colour red. |
| June 8th, 2016 | * I started testing my code with other people and acting on their feedback. One of the adjustments I made in order to make the screen more user friendly was that, when the user hovered over a button, the button would become translucent to indicate that the user’s mouse was hovering over that button. * I also started to slightly optimize my game from previous versions by removing code that was unnecessary to the game. |
| June 9th, 2016 | * I further worked on some of the suggestions given. For example, I edited some of the buttons so that if the user was going back to the game screen, then the button would say “Continue”, but if the player(s) were starting a new game, then the button would read “Play Game”. |

Trial Cases: ChessBasic Program

I performed several tests in the code throughout the time period in which this code was completed. These tests included testing the logic primarily, as there are *many* possible combinations of games that are possible, even for a simplified version of the game as is my chess game.

After counting, in my modified version of chess, I found that each player has 23 legal moves in their arsenal on the first piece they move. Thus, after the first complete move (i.e. white moves, then black moves), there are already 529 possible games that can arise. This meant that instead of testing all the possible games in my chess program, I tested *families* of games that applied general scenarios to test if all the possible games might work.

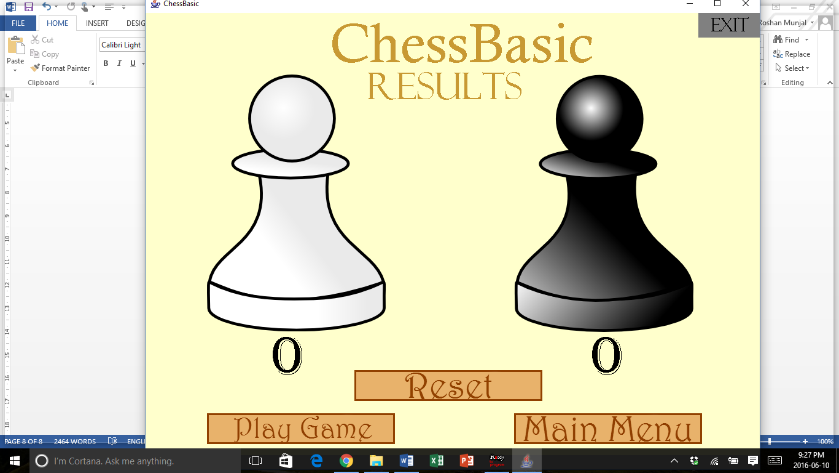
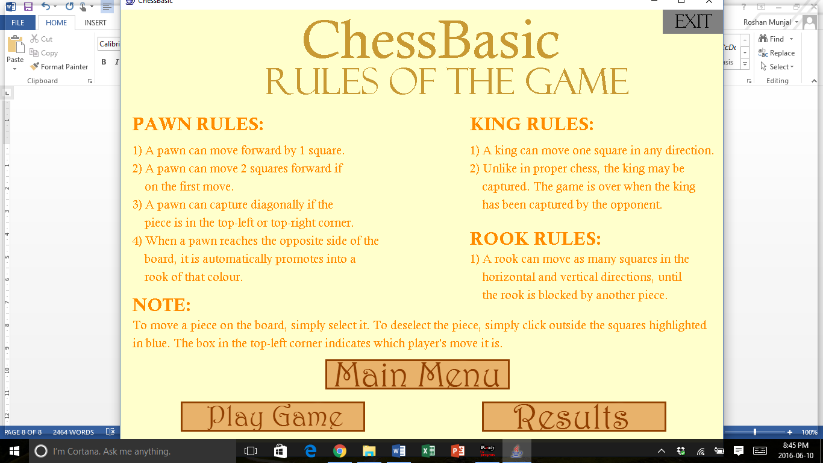
Below are some of the trial cases for the logic of the program to see if in all cases, that the pieces have the correct available moves. Also, it makes sure that none of the logic is significantly flawed (e.g. the piece moves outside the board).

|  |  |
| --- | --- |
| *Trial Case* | *Screen Image* |
| Here is a situation from one of the games that I played on the program. It is seen that the white king, in this situation, has a limited number of squares that it can travel to. The top-left and top-right square would each result in a check if the king moves there, but my program does NOT encompass any methods to check for checks or checkmates. However, the spaces that are blocked by white pieces cannot be occupied by the white king, as shown. |  |
| A situation in a regular game. This is a situation in which a normal game may turn out. It involves a situation where one rook of each colour is on the board. Here, it can be seen that the king has been given all its moves as possible moves. It is seen that the king can move to a square where the white pawn can immediately attack it. This is because there is no check or checkmate method because time constraints did not allow for this feature. |  |
| Here, it is shown that if a piece is blocking the pawn, then the pawn cannot move forward. However, since there is a pawn to the top-left of the white pawn of the opponent, the white pawn can capture the other pawn. |  |
| Here, the possible moves for the black pawn are shown. It shows that the pawn can move one space forward, two spaces forward, or can capture the white pawn that is diagonally to the bottom-left. This shows that the program can determine if these moves are possible. |  |
| Here, it can be seen that the rook can move up to 6 squares left in the chess situation. It cannot capture the pawn ahead of it because it is a pawn of the same colour – black. Thus, all the highlighted squares are valid moves for the rook. |  |
| Here, this specific case shows that the white rook has a very specific case of possible moves. It can capture the black rook, here, but cannot go to the square beyond it. Similarly, to the left, the rook can move up to the white pawn, but cannot capture it as it is of the same colour. Below, the rook can move a maximum of two spaces down, and to the right, the rook cannot cross the king. Thus, this shows that all the possible moves for the rook in this case have been highlighted. |  |

These were some of the general test cases to prove that all the pieces on the board function as they are supposed to under normal gameplay. Next, various test cases will check if the program functions properly under extreme cases.

The test I will perform now will be to check if all the screens can be interlinked via moving around, and if they correctly display the screen that was clicked on.







In the diagrams above, all the screens have been tested to be linked to one another. Thus, any screen can essentially be accessed from any other screen. The only exception is that you cannot go directly from the “Results” page to the “Rules of the Game” page, which in itself is not a very useful or necessary feature, as the user would likely go back to the “Main Menu” page or the “Game Screen”. Thus, it was not included in the final program.

Next, some of the extreme cases involving the actual game itself will be tested, to see if any bugs occur. Each test case has a description written next to it, to explain the test case. During all these test cases, it was also checked that the user could go back and forth between the game screen, rules of the game screen, and the results screen.

|  |  |
| --- | --- |
| *Trial Case* | *Screen Image* |
| Here, a special case was performed where only the white pawn was moved forward. In turn, the king only moved sideways until it was captured by the white pawn, which correctly promoted into a rook as it captured the black king to finish the game. As seen, it says “White Wins!” on the bottom screen, but it cannot be seen because the screen size is too small. |  |
| This was another extreme test case to see if all pieces would function properly, if, at the start of the game, only the pawns would be moved to line up with each other at the centre. This is proven because the pawn highlighted has the correct available moves. Also, the kings and rooks were also shown to move properly through the board as they were checked for inconsistencies – none were found. |  |
| In this test scenario, I wanted to check if the program could recognize if there was no black king on the board, even when no other pieces were on the board. I did this by having the white king capture the black king, just to check if it was possible. And thus, as seen from the text at the bottom (hidden due to screen size), the program does recognize white as having won. |  |
| In this test scenario, although technically, the white king has no legal moves, it is shown as having 3 legal moves. In my modified version of chess, the king must be captured to win the game, which means that there is almost NO scenario that encompasses a situation where a player has NO moves at all (even those that put the king in check, as shown here). |  |
| In this test case, I wanted to test that all pieces would work fine even when all rooks were removed off the board. Evidently, everything seems to be in working order as the highlighted black king seems to have the right valid moves for his certain position. Also, all the pawns seem to be acting properly, as they would in a regular game. |  |
| Here, even under these extreme conditions, white is not recognized as having lost because the definition of losing is very strict in my game – to remove the opponent’s king. Thus, if the white king captured the black king in this case, it would still be an example of white winning. As seen, the program still generates all the possible values for where the white king can move, even though it truly has no legal squares in a real chess game. |  |