1 Appendix: The Code

2 Appendix: The Code

3 Appendix: The Code

4 Appendix: The Code

5 Appendix: The Code

6 Appendix: The Code

7 Appendix: The Code

8 Appendix: The Code

9 Appendix: The Code

10 Appendix: The Code

11 Appendix: The Code

12 Appendix: The Code

13 Appendix: The Code

14 Appendix: The Code

15 Appendix: The Code

16 Appendix: The Code

17 Appendix: The Code

18 Appendix: The Code

19 Appendix: The Code

20 Appendix: The Code

21 Appendix: The Code

22 Appendix: The Code

23 Appendix: The Code

24 Appendix: The Code

A Appendix: The Code

A.1 Back-end

A.1.1 MovementRook.m

```
1
2  function [possiblemoves] = MovementRook(chessboard,piece_colour,p_x,p_y)
3
4  %Initialisation values ------------------------------------------------------
5  r_colour = piece_colour(p_x,p_y);
6  possiblemoves = zeros(8,8);
7
8  %This section allows movement in vertical direction ------------------------
9  i = 1;
10 while(p_x+i<9)
11     if(piece_colour(p_x+i,p_y)== r_colour)
12         break
13     end
14     if(piece_colour(p_x+i,p_y)~= r_colour && chessboard(p_x+i,p_y)~=0)
15         possiblemoves(p_x+i,p_y) = 2;
16         break
17     end
18     possiblemoves(p_x+i,p_y) = 1;
19     i = i+1;
20 end
21
22 i = 1;
23 while(p_x-i>0)
24     if(piece_colour(p_x-i,p_y)== r_colour)
25         break
26     end
27      if(piece_colour(p_x-i,p_y)~= r_colour && chessboard(p_x-i,p_y)~=0)
28         possiblemoves(p_x-i,p_y) = 2;
29         break
30     end
31     possiblemoves(p_x-i,p_y) = 1;
32     i = i+1;
33 end
34
35 %This section allows movement in the horizontal direction
36 i = 1;
37 while(p_y+i<9)
38     if(piece_colour(p_x,p_y+i)== r_colour)
39         break
40     end
41     if(piece_colour(p_x,p_y+i)~= r_colour && chessboard(p_x,p_y+i)~=0)
42         possiblemoves(p_x,p_y+i) = 2;
43         break
44     end
45     possiblemoves(p_x,p_y+i) = 1;
46     i = i+1;
47 end
48
49 i = 1;
50 while(p_y-i>0)
51     if(piece_colour(p_x,p_y-i)== r_colour)
52         break
53     end
54     if(piece_colour(p_x,p_y-i)~= r_colour && chessboard(p_x,p_y-i)~=0)
55         possiblemoves(p_x,p_y-i) = 2;
56         break
57     end
```

```
58      possiblemoves(p_x,p_y-i) = 1;
59      i = i+1;
60  end
61
62  %-------------------------------------------------------------------------
63
64  end
```

A.1.2 MovementQueen.m

```
1   function [possiblemoves] = MovementQueen(chessboard,piece_colour,p_x,p_y)
2
3   %Initialisation values ------------------------------------------------------
4   possiblemoves = zeros(8,8);
5   r_colour = piece_colour(p_x,p_y);
6
7   %This section allows movement in / direction ---------------------------------
8   i=1;
9   while(p_x+i<9 && p_y+i<9)
10      if(piece_colour(p_x+i,p_y+i)== r_colour)
11          break
12      end
13      if(piece_colour(p_x+i,p_y+i)~= r_colour && chessboard(p_x+i,p_y+i)~=0)
14          possiblemoves(p_x+i,p_y+i) = 2;
15          break
16      end
17      possiblemoves(p_x+i,p_y+i) = 1;
18      i = i+1;
19  end
20
21  i=1;
22  while(p_x-i>0 && p_y-i>0)
23      if(piece_colour(p_x-i,p_y-i)== r_colour)
24          break
25      end
26      if(piece_colour(p_x-i,p_y-i)~= r_colour && chessboard(p_x-i,p_y-i)~=0)
27          possiblemoves(p_x-i,p_y-i) = 2;
28          break
29      end
30      possiblemoves(p_x-i,p_y-i) = 1;
31      i = i+1;
32  end
33
34  %This section allows movement in the \ direction-----------------------------
35  i=1;
36  while(p_x+i<9 && p_y-i>0)
37      if(piece_colour(p_x+i,p_y-i)== r_colour)
38          break
39      end
40      if(piece_colour(p_x+i,p_y-i)~= r_colour && chessboard(p_x+i,p_y-i)~=0)
41          possiblemoves(p_x+i,p_y-i) = 2;
42          break
43      end
44      possiblemoves(p_x+i,p_y-i) = 1;
45      i = i+1;
46  end
47
48  i=1;
49  while(p_x-i>0 && p_y+i<9)
50      if(piece_colour(p_x-i,p_y+i)== r_colour)
51          break
52      end
```

```
53      if(piece_colour(p_x-i,p_y+i)~= r_colour && chessboard(p_x-i,p_y+i)~=0)
54          possiblemoves(p_x-i,p_y+i) = 2;
55          break
56      end
57      possiblemoves(p_x-i,p_y+i) = 1;
58      i = i+1;
59  end
60
61  %This section allows movement in vertical direction ------------------------------
62  i = 1;
63  while(p_x+i<9)
64      if(piece_colour(p_x+i,p_y)== r_colour)
65          break
66      end
67      if(piece_colour(p_x+i,p_y)~= r_colour && chessboard(p_x+i,p_y)~=0)
68          possiblemoves(p_x+i,p_y) = 2;
69          break
70      end
71      possiblemoves(p_x+i,p_y) = 1;
72      i = i+1;
73  end
74
75  i = 1;
76  while(p_x-i>0)
77      if(piece_colour(p_x-i,p_y)== r_colour)
78          break
79      end
80       if(piece_colour(p_x-i,p_y)~= r_colour && chessboard(p_x-i,p_y)~=0)
81          possiblemoves(p_x-i,p_y) = 2;
82          break
83      end
84      possiblemoves(p_x-i,p_y) = 1;
85      i = i+1;
86  end
87
88  %This section allows movement in the horizontal direction-------------------------
89  i = 1;
90  while(p_y+i<9)
91      if(piece_colour(p_x,p_y+i)== r_colour)
92          break
93      end
94      if(piece_colour(p_x,p_y+i)~= r_colour && chessboard(p_x,p_y+i)~=0)
95          possiblemoves(p_x,p_y+i) = 2;
96          break
97      end
98      possiblemoves(p_x,p_y+i) = 1;
99      i = i+1;
100 end
101
102 i = 1;
103 while(p_y-i>0)
104     if(piece_colour(p_x,p_y-i)== r_colour)
105         break
106     end
107     if(piece_colour(p_x,p_y-i)~= r_colour && chessboard(p_x,p_y-i)~=0)
108         possiblemoves(p_x,p_y-i) = 2;
109         break
110     end
111     possiblemoves(p_x,p_y-i) = 1;
112     i = i+1;
113 end
114 %-------------------------------------------------------------------------------
115
116 end
```

A.1.3 MovementPawn.m

```
function [possiblemoves] = MovementPawn(chessboard,piece_colour,num_moves,p_x,p_y)

%Initialisation values ------------------------------------------------------
r_colour = piece_colour(p_x,p_y);
possiblemoves = zeros(8,8);

%This section allows all movements after checking whether it exceeds the board or not
    ----------------
switch r_colour
    case 119 %White case
        %En passant-----------------------------------------------------------
        if (p_x==4)

            if(p_x-1>0 && p_y-1 >0) %Capture left
                if(piece_colour(p_x,p_y-1)~=r_colour && chessboard(p_x,p_y-1)==1 &&
                    num_moves(p_x,p_y-1)==1)
                    possiblemoves(p_x-1,p_y-1) = 3;
                end
            end

            if(p_x-1>0 && p_y+1<9) %Capture right
                if(piece_colour(p_x,p_y+1)~=r_colour && chessboard(p_x,p_y+1)==1 &&
                    num_moves(p_x,p_y+1)==1)
                    possiblemoves(p_x-1,p_y+1) = 3;
                end
            end

        end

        if(p_x-1>0) %Forward movement
            if(chessboard(p_x-1,p_y)==0)
                possiblemoves(p_x-1,p_y) = 1;
            end
        end

        %Initial forward movement
        if(p_x==7 && chessboard(p_x-2,p_y)==0 && chessboard(p_x-1,p_y)==0)
        possiblemoves(p_x-2,p_y) = 1;
        end

        if(p_x-1>0 && p_y-1 >0) %Capture left
            if(piece_colour(p_x-1,p_y-1)~=r_colour && chessboard(p_x-1,p_y-1)~=0)
                possiblemoves(p_x-1,p_y-1) = 2;
                   if(p_x==2) %Capture and pawn promotion
                      possiblemoves(p_x-1,p_y-1) = 5;
                   end
            end
        end

        if(p_x-1>0 && p_y+1<9) %Capture right
            if(piece_colour(p_x-1,p_y+1)~=r_colour && chessboard(p_x-1,p_y+1)~=0)
                possiblemoves(p_x-1,p_y+1) = 2;
                   if(p_x==2) %Capture and pawn promotion
                      possiblemoves(p_x-1,p_y+1) = 5;
                   end
            end
        end

        %Pawn promotion-------------------------------------------------------
        if(p_x==2)
            if(chessboard(p_x-1,p_y)==0)
```

```
59                  possiblemoves(p_x-1,p_y) = 5;
60              end
61          end
62
63      case 98 %Black Case
64
65          %En passant-------------------------------------------------------------
66          if (p_x==5)
67
68              if(p_x-1>0 && p_y-1 >0) %Capture left
69                  if(piece_colour(p_x,p_y-1)~=r_colour && chessboard(p_x,p_y-1)==1 &&
                        num_moves(p_x,p_y-1)==1)
70                      possiblemoves(p_x+1,p_y-1) = 3;
71                  end
72              end
73
74              if(p_x-1>0 && p_y+1<9) %Capture right
75                  if(piece_colour(p_x,p_y+1)~=r_colour && chessboard(p_x,p_y+1)==1 &&
                        num_moves(p_x,p_y+1)==1)
76                      possiblemoves(p_x+1,p_y+1) = 3;
77                  end
78              end
79
80          end
81
82          if(p_x+1<9) %Forward movement
83              if(chessboard(p_x+1,p_y)==0)
84                  possiblemoves(p_x+1,p_y) = 1;
85              end
86          end
87
88          %Initial Forward movement
89          if(p_x==2 && chessboard(p_x+2,p_y)==0 && chessboard(p_x+1,p_y)==0)
90              possiblemoves(p_x+2,p_y) = 1;
91          end
92
93          if(p_x+1<9 && p_y-1>0) %Capture left
94              if(piece_colour(p_x+1,p_y-1)~=r_colour && chessboard(p_x+1,p_y-1)~=0)
95                  possiblemoves(p_x+1,p_y-1) = 2;
96                    if(p_x==7) %Capture and pawn promotion
97                        possiblemoves(p_x+1,p_y-1) = 5;
98                     end
99              end
100         end
101
102         if(p_x+1<9 && p_y+1<9) %Capture right
103             if(piece_colour(p_x+1,p_y+1)~=r_colour && chessboard(p_x+1,p_y+1)~=0)
104                 possiblemoves(p_x+1,p_y+1) = 2;
105                    if(p_x==7) %Capture and pawn promotion
106                       possiblemoves(p_x+1,p_y+1) = 5;
107                    end
108             end
109         end
110
111         %Pawn promotion---------------------------------------------------------
112         if(p_x==7)
113             if(chessboard(p_x+1,p_y)==0)
114                 possiblemoves(p_x+1,p_y) = 5;
115             end
116         end
117
118 end
119 %-----------------------------------------------------------------------------
120 end
```

A.1.4 MovementKnight.m

```
function [possiblemoves] = MovementKnight(chessboard,piece_colour,p_x,p_y)

%Initialisation values ------------------------------------------------------
r_colour = piece_colour(p_x,p_y);
possiblemoves = zeros(8,8);

%This sections allows L shaped movements for knight
if(p_x-2>0 & p_y-1>0)
    if (piece_colour(p_x-2,p_y-1)~= r_colour && chessboard(p_x-2,p_y-1)~=0)
        possiblemoves(p_x-2,p_y-1) = 2;
    elseif (piece_colour(p_x-2,p_y-1)== r_colour)
        ;
    else
        possiblemoves(p_x-2,p_y-1) = 1;
    end
end

if(p_x-2>0 & p_y+1<9)
    if (piece_colour(p_x-2,p_y+1)~= r_colour && chessboard(p_x-2,p_y+1)~=0)
        possiblemoves(p_x-2,p_y+1) = 2;
    elseif (piece_colour(p_x-2,p_y+1)== r_colour)
        ;
    else
        possiblemoves(p_x-2,p_y+1) = 1;
    end
end

if(p_x-1>0 & p_y-2>0)
    if (piece_colour(p_x-1,p_y-2)~= r_colour && chessboard(p_x-1,p_y-2)~=0)
        possiblemoves(p_x-1,p_y-2) = 2;
    elseif (piece_colour(p_x-1,p_y-2)== r_colour)
        ;
    else
        possiblemoves(p_x-1,p_y-2) = 1;
    end
end

if(p_x-1>0 & p_y+2<9)
    if (piece_colour(p_x-1,p_y+2)~= r_colour && chessboard(p_x-1,p_y+2)~=0)
        possiblemoves(p_x-1,p_y+2) = 2;
    elseif (piece_colour(p_x-1,p_y+2)== r_colour)
        ;
    else
        possiblemoves(p_x-1,p_y+2) = 1;
    end
end

if(p_x+1<9 & p_y-2>0)
    if (piece_colour(p_x+1,p_y-2)~= r_colour && chessboard(p_x+1,p_y-2)~=0)
        possiblemoves(p_x+1,p_y-2) = 2;
    elseif (piece_colour(p_x+1,p_y-2)== r_colour)
        ;
    else
        possiblemoves(p_x+1,p_y-2) = 1;
    end
end

if(p_x+1<9 & p_y+2<9)
    if (piece_colour(p_x+1,p_y+2)~= r_colour && chessboard(p_x+1,p_y+2)~=0)
        possiblemoves(p_x+1,p_y+2) = 2;
    elseif (piece_colour(p_x+1,p_y+2)== r_colour)
```

```
62          ;
63      else
64          possiblemoves(p_x+1,p_y+2) = 1;
65      end
66  end
67
68  if(p_x+2<9 & p_y-1>0)
69      if (piece_colour(p_x+2,p_y-1)~= r_colour && chessboard(p_x+2,p_y-1)~=0)
70          possiblemoves(p_x+2,p_y-1) = 2;
71      elseif (piece_colour(p_x+2,p_y-1)== r_colour)
72          ;
73      else
74          possiblemoves(p_x+2,p_y-1) = 1;
75      end
76  end
77
78  if(p_x+2<9 & p_y+1<9)
79      if (piece_colour(p_x+2,p_y+1)~= r_colour && chessboard(p_x+2,p_y+1)~=0)
80          possiblemoves(p_x+2,p_y+1) = 2;
81      elseif (piece_colour(p_x+2,p_y+1)== r_colour)
82          ;
83      else
84          possiblemoves(p_x+2,p_y+1) = 1;
85      end
86  end
87
88  %-------------------------------------------------------------------------
89
90  end
```

A.1.5 MovementKing.m

```
1   function [possiblemoves] = MovementKing(chessboard,piece_colour,num_moves,
        potential_moves,p_x,p_y)
2
3   %Initialisation values ------------------------------------------------------
4   r_colour = piece_colour(p_x,p_y);
5   possiblemoves = zeros(8,8);
6
7   %This section allows all movements after checking whether it exceeds the board or not
8   %and ensures that the king is not moving into square that is in check --------------
9
10  %-------------------------------------------------------------------------
11  %                        Movement (8 Directions)
12  %-------------------------------------------------------------------------
13
14
15      if(p_x+1<9)
16          if (piece_colour(p_x+1,p_y)~= r_colour && chessboard(p_x+1,p_y)~=0)
17              possiblemoves(p_x+1,p_y) = 2;
18          elseif (piece_colour(p_x+1,p_y)== r_colour)
19              ;
20          else
21              possiblemoves(p_x+1,p_y) = 1;
22          end
23      end
24
25
26      if(p_x+1<9 && p_y+1<9)
27          if (piece_colour(p_x+1,p_y+1)~= r_colour && chessboard(p_x+1,p_y+1)~=0)
28              possiblemoves(p_x+1,p_y+1) = 2;
29          elseif (piece_colour(p_x+1,p_y+1)== r_colour)
```

```
30              ;
31          else
32              possiblemoves(p_x+1,p_y+1) = 1;
33          end
34      end
35
36
37      if(p_x+1<9 && p_y-1>0)
38          if (piece_colour(p_x+1,p_y-1)~= r_colour && chessboard(p_x+1,p_y-1)~=0)
39              possiblemoves(p_x+1,p_y-1) = 2;
40          elseif (piece_colour(p_x+1,p_y-1)== r_colour)
41              ;
42          else
43              possiblemoves(p_x+1,p_y-1) = 1;
44          end
45      end
46
47
48      if(p_y+1<9)
49          if (piece_colour(p_x,p_y+1)~= r_colour && chessboard(p_x,p_y+1)~=0)
50              possiblemoves(p_x,p_y+1) = 2;
51          elseif (piece_colour(p_x,p_y+1)== r_colour)
52              ;
53          else
54              possiblemoves(p_x,p_y+1) = 1;
55          end
56      end
57
58
59      if(p_y-1>0)
60          if (piece_colour(p_x,p_y-1)~= r_colour && chessboard(p_x,p_y-1)~=0)
61              possiblemoves(p_x,p_y-1) = 2;
62          elseif (piece_colour(p_x,p_y-1)== r_colour)
63              ;
64          else
65              possiblemoves(p_x,p_y-1) = 1;
66          end
67      end
68
69
70      if(p_x-1>0)
71          if (piece_colour(p_x-1,p_y)~= r_colour && chessboard(p_x-1,p_y)~=0)
72              possiblemoves(p_x-1,p_y) = 2;
73          elseif (piece_colour(p_x-1,p_y)== r_colour)
74              ;
75          else
76              possiblemoves(p_x-1,p_y) = 1;
77          end
78      end
79
80
81      if(p_x-1>0 && p_y+1<9)
82          if (piece_colour(p_x-1,p_y+1)~= r_colour && chessboard(p_x-1,p_y+1)~=0)
83              possiblemoves(p_x-1,p_y+1) = 2;
84          elseif (piece_colour(p_x-1,p_y+1)== r_colour)
85              ;
86          else
87              possiblemoves(p_x-1,p_y+1) = 1;
88          end
89      end
90
91
92      if(p_x-1>0 && p_y-1>0)
93          if (piece_colour(p_x-1,p_y-1)~= r_colour && chessboard(p_x-1,p_y-1)~=0)
94              possiblemoves(p_x-1,p_y-1) = 2;
```

```
95          elseif (piece_colour(p_x-1,p_y-1)== r_colour)
96              ;
97          else
98              possiblemoves(p_x-1,p_y-1) = 1;
99          end
100     end
101
102 %-------------------------------------------------------------------------
103 %                                 Castling
104 %-------------------------------------------------------------------------
105
106
107 %------------------------------ For white king ---------------------------
108 %Checks to see if traversed squares are in check and if the final square is
109 %checked. Also checks if squares in between are empty
110     if (piece_colour(p_x,p_y)==119 && num_moves(p_x,p_y)==0 && num_moves(8,8)==0 ....
111             && piece_colour(8,6)==0 && piece_colour(8,7)==0 && potential_moves(8,6)==0
                    ...
112             && potential_moves(8,7)==0 && chessboard(8,8)==5 && piece_colour(8,8)==119 )
113         possiblemoves(8,7) = 4;
114     end
115     if (piece_colour(p_x,p_y)==119 && num_moves(p_x,p_y)==0 && num_moves(8,1)==0 ....
116             && piece_colour(8,2)==0 && piece_colour(8,3)==0 && piece_colour(8,4)==0 ...
117             && potential_moves(8,3)==0 && potential_moves(8,4)==0 && chessboard(8,1)==5
                    ...
118             && piece_colour(8,1)==119)
119         possiblemoves(8,3) = 4;
120     end
121
122 %------------------------------ For black king ---------------------------
123 %Checks to see if traversed squares are in check and if the final square is
124 %checked
125     if (piece_colour(p_x,p_y)==98 && num_moves(p_x,p_y)==0 && num_moves(1,1)==0 ....
126             && piece_colour(1,2)==0 && piece_colour(1,3)==0 && piece_colour(1,4)==0 ....
127             && potential_moves(1,3)==0 && potential_moves(1,4)==0 && ...
128             chessboard(1,1)==5 && piece_colour(1,1)==98)
129         possiblemoves(1,3) = 4;
130     end
131     if (piece_colour(p_x,p_y)==98 && num_moves(p_x,p_y)==0 && num_moves(1,8)==0 ....
132             && piece_colour(1,6)==0 && piece_colour(1,7)==0 && potential_moves(1,6)==0
                    ....
133             && potential_moves(1,7)==0 && chessboard(1,8)==5 && piece_colour(1,8)==98)
134         possiblemoves(1,7) = 4;
135     end
136
137
138 possiblemoves(p_x,p_y)=0;
139 %-------------------------------------------------------------------------
140 end
```

A.1.6 MovementBishop.m

```
1  function [possiblemoves] = MovementBishop(chessboard,piece_colour,p_x,p_y)
2
3  %Initialisation values ---------------------------------------------------
4  r_colour = piece_colour(p_x,p_y);
5  possiblemoves = zeros(8,8);
6
7  %This section allows movement in / direction -----------------------------
8  i=1;
9  while(p_x+i<9 && p_y+i<9)
10     if(piece_colour(p_x+i,p_y+i)== r_colour)
```

```
        break
    end
    if(piece_colour(p_x+i,p_y+i)~= r_colour && chessboard(p_x+i,p_y+i)~=0)
        possiblemoves(p_x+i,p_y+i) = 2;
        break
    end
    possiblemoves(p_x+i,p_y+i) = 1;
    i = i+1;
end

i=1;
while(p_x-i>0 && p_y-i>0)
    if(piece_colour(p_x-i,p_y-i)== r_colour)
        break
    end
    if(piece_colour(p_x-i,p_y-i)~= r_colour && chessboard(p_x-i,p_y-i)~=0)
        possiblemoves(p_x-i,p_y-i) = 2;
        break
    end
    possiblemoves(p_x-i,p_y-i) = 1;
    i = i+1;
end

%This section allows movement in the \ direction-------------------------
i=1;
while(p_x+i<9 && p_y-i>0)
    if(piece_colour(p_x+i,p_y-i)== r_colour)
        break
    end
    if(piece_colour(p_x+i,p_y-i)~= r_colour && chessboard(p_x+i,p_y-i)~=0)
        possiblemoves(p_x+i,p_y-i) = 2;
        break
    end
    possiblemoves(p_x+i,p_y-i) = 1;
    i = i+1;
end

i=1;
while(p_x-i>0 && p_y+i<9)
    if(piece_colour(p_x-i,p_y+i)== r_colour)
        break
    end
    if(piece_colour(p_x-i,p_y+i)~= r_colour && chessboard(p_x-i,p_y+i)~=0)
        possiblemoves(p_x-i,p_y+i) = 2;
        break
    end
    possiblemoves(p_x-i,p_y+i) = 1;
    i = i+1;
end

%-------------------------------------------------------------------------

end
```

A.2 Front-end

A.2.1 ClickPiece.m

```
%ClickPiece Obtains all the data from a user's click, highlights possible
%moves and allows the user to make that move.
function [varargout]=ClickPiece(var1,var2,B,piece_colour,chessboard,...
    num_moves,parameters,potentialmoves,handles,varargin )
```

```
5
6  set(handles.gameconsole,'String','')
7
8  %------------Determines which colour is able to be selected--------------------
9  if(mod(B.info.turn,2)==1)
10     colourturn = 119;
11     oppositecolour = 98;
12 else
13     colourturn = 98;
14     oppositecolour = 119;
15 end
16
17 onlyAIoption = 0;
18 %-------------------------------------------------------------------------------
19  clickP = get(gca,'CurrentPoint');
20       x = ceil(clickP(1,2));
21       y = ceil(clickP(1,1));
22 %------ Conversion from Graph Grid to B.top grid ------------------------------
23       x = 13-x;
24       y = y + 4;
25 %-------------------------------------------------------------------------------
26 %This is the board
27       piecetype = B.top(x,y).name;
28
29 %---------Conversion from B.Top grid to Chessboard grid----------------------
30       p_x = x - 4;
31       p_y = y - 4;
32
33 if(piece_colour(p_x,p_y) == colourturn)
34 %-------------------------Generates Possible Moves-------------------------------
35
36 switch piecetype
37     case 'pawn'
38         [possiblemoves] = MovementPawn(chessboard,piece_colour,num_moves,p_x,p_y);
39     case 'rook'
40         [possiblemoves] = MovementRook(chessboard,piece_colour,p_x,p_y);
41     case 'knight'
42         [possiblemoves] = MovementKnight(chessboard,piece_colour,p_x,p_y);
43     case 'bishop'
44         [possiblemoves] = MovementBishop(chessboard,piece_colour,p_x,p_y);
45     case 'queen'
46         [possiblemoves] = MovementQueen(chessboard,piece_colour,p_x,p_y);
47     case 'king'
48         [possiblemoves] = MovementKing(chessboard,piece_colour,num_moves,...
49             potentialmoves,p_x,p_y);
50 end
51
52 %-------------------------------------------------------------------------------
53 %             REDRAWS THE BOARD BUT HIGHLIGHTS POSSIBLE MOVES
54 %-------------------------------------------------------------------------------
55 %-----------------------------------Draws Rectangles----------------------------
56 icount=0;
57 for i=1:71
58          icount=icount+1;
59          if mod(i,2)==1
60              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
61                  parameters.dx ,parameters.dx],'Curvature',[0,0],...
62                  'FaceColor',[0.82 0.545 0.278])
63          else
64              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
65                  parameters.dx ,parameters.dx],...
66                  'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
67          end
68 end
69
```

```
%------------ Highlights possible moves-----------------------------------
for r=1:parameters.rows
    for c=1:parameters.cols
        switch possiblemoves(r,c)
%----------------------Highlights movable squares-----------------------
            case 1
             rectangle('Position',[parameters.xx(9-r,c),parameters.yy(9-r,c),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],'FaceColor','y',...
                 'ButtonDownFcn',{@ClickMovePiece,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0})
%----------------------Highlights capturable squares---------------------
            case 2
             rectangle('Position',[parameters.xx(9-r,c),parameters.yy(9-r,c),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],'FaceColor','r')
%----------------------Highlights Enpassant Squares----------------------
            case 3
             rectangle('Position',[parameters.xx(9-r,c),parameters.yy(9-r,c),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],'FaceColor','r',...
                 'ButtonDownFcn',{@ClickEnpassant,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0})
%----------------------Highlights Castling Squares-----------------------
            case 4
             rectangle('Position',[parameters.xx(9-r,c),parameters.yy(9-r,c),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],'FaceColor','b',...
                 'ButtonDownFcn',{@ClickCastling,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0})
%----------------------Highlights Pawn Promotion Square------------------
            case 5
             rectangle('Position',[parameters.xx(9-r,c),parameters.yy(9-r,c),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],'FaceColor','c',...
                 'ButtonDownFcn',{@ClickPawnPromo,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0,0})
         end
    end
end
%-------------------------------------------------------------------------
%                            Redraws images
%-------------------------------------------------------------------------
for r=1:parameters.rows
    for c=1:parameters.cols
        if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
            % load the image
            [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
            % draw the image
            %If Statement enables capture move
            if possiblemoves(r,c) == 2
                imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r
                    +1,...
                mirrorImage(X),'AlphaData',mirrorImage(alpha),...
                'ButtonDownFcn',{@ClickCapturePiece,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0});
             %Enables Pawn Promotion
            elseif possiblemoves(r,c) == 5 && chessboard(r,c)~=0
                imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r
                    +1,...
                mirrorImage(X),'AlphaData',mirrorImage(alpha),...
                'ButtonDownFcn',{@ClickPawnPromo,x,y,B,piece_colour,chessboard...
                 ,num_moves,parameters,possiblemoves,handles,onlyAIoption,0,0});
             %Else enable click piece
            else
            imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
                mirrorImage(X),'AlphaData',mirrorImage(alpha),...
                'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
                num_moves,parameters,potentialmoves,handles,onlyAIoption,0,0});
            end
```

```
        end
    end
end
drawnow;
end
end
%-------------------------------------------------------------------------
%-------------------------------------------------------------------------
```

A.2.2 ClickCapturePiece.m

```
%CapturePiece Part of the Click Series of Functions - Enables capture
function [chessboard,piece_colour, num_moves,allowscheck]=ClickCapturePiece(v1,v2,x_ori,
    y_ori,B,piece_colour,chessboard,...
    num_moves,parameters,PM,handles,onlyAIoption,move_x,move_y,varargin)

%-------------------------------------------------------------------------
%                  Init values,conversions and click location
%-------------------------------------------------------------------------
if(mod(B.info.turn,2)==1)
    colourturn = 119;
    oppositecolour = 98;
else
    colourturn = 98;
    oppositecolour = 119;
end

if onlyAIoption == 0
clickP = get(gca,'CurrentPoint');
      x = ceil(clickP(1,2));
      y = ceil(clickP(1,1));
%---- Conversion from Graph grid to B.top grid ----------------------------
      x = 13-x;
      y = y + 4;
%---------Conversion from B.Top grid to Chessboard grid-------------------
      p_x = x - 4; %p_x is necessary because it is the current clicked position
      p_y = y - 4;
      ori_x = x_ori - 4; %The difference is that ori_x is for chessboard,
      ori_y = y_ori - 4; %x_ori is for B.top
else
    p_x = move_x;   %Where is it moving to
    p_y = move_y;
    ori_x = x_ori;   %Where was it originally
    ori_y = y_ori;
end

%-------------------------------------------------------------------------
%            Checks if King is exposed to check in any way
%-------------------------------------------------------------------------
%The method used is to create a future chessboard based on the move
%requested

fboard = chessboard;
f_p_colour= piece_colour;
f_num_moves = num_moves;
%This step officially moves the piece
fboard(p_x,p_y) = chessboard(ori_x,ori_y);
f_p_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
f_num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
%This step empties the previous box
fboard(ori_x,ori_y) = 0;
f_p_colour(ori_x,ori_y) = 0;
```

```
f_num_moves(ori_x,ori_y) = 0;

%Analyses the future board
[potentialfuturemoves,capt_index_future] = analyseboard(fboard,...
    f_p_colour,f_num_moves,oppositecolour);
[allowscheck]=KingCheck(fboard,f_p_colour,colourturn,...
    capt_index_future,potentialfuturemoves);
if allowscheck==1 && onlyAIoption == 0
    set(handles.gameconsole,'String','King will be left in check, move invalid')
end
%-------------------------------------------------------------------------
%Ensures it can only move legally
if PM(p_x,p_y)==2 && chessboard(p_x,p_y) ~= 10 && allowscheck==0

B.info.turn = B.info.turn + 1;
%-------------------------------------------------------------------------
%              This is to edit the backend chessboard matrix
%-------------------------------------------------------------------------
%This step officially moves the piece
chessboard = fboard;
piece_colour = f_p_colour;
num_moves = f_num_moves;


%---------------To Check Opposing Side ---------------------------------
[potentialmoves,capt_index] = analyseboard(chessboard,piece_colour,num_moves,colourturn)
    ;
[checkopp]=KingCheck(chessboard,piece_colour,oppositecolour,capt_index,potentialmoves);
if checkopp == 1 && onlyAIoption == 0
   set(handles.checkstat,'String','Check')
    [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
    if ischeckmate
        set(handles.checkstat,'String','Checkmate, White Wins')
    end
elseif checkopp == 0 && onlyAIoption ==0
    [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
    if ischeckmate
        set(handles.checkstat,'String','Stalemate')
    else
        set(handles.checkstat,'String','')
    end
end

if onlyAIoption == 0
    [B] = readchessboard(B,chessboard,piece_colour);
%-------------------------------------------------------------------------
%                           Redraws the Board
%-------------------------------------------------------------------------
icount=0;
for i=1:71
         icount=icount+1;
         if mod(i,2)==1
             rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],...
                 'FaceColor',[0.82 0.545 0.278])
         else
            rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
                parameters.dx ,parameters.dx],...
                'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
         end
end

for r=1:parameters.rows
    for c=1:parameters.cols
        if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
```

```
115             % load the image
116             [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
117             % draw the image
118             imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
119                 mirrorImage(X),'AlphaData',mirrorImage(alpha),...
120                 'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
121                 num_moves,parameters,potentialmoves,handles});
122         end
123     end
124 end
125 drawnow;
126 if(get(handles.choice2,'Value')==1)
127     AIControl(B,piece_colour,chessboard,num_moves,parameters, handles);
128 end
129 if(get(handles.choice3,'Value')==1)
130     AIvsAI(B,piece_colour,chessboard,num_moves,parameters, handles)
131 end
132 if(get(handles.choice1,'Value')==1)
133     PlayerVsPlayer( B,piece_colour,chessboard,num_moves,parameters, handles )
134 end
135 end
136 %-------------------------------------------------------------------------------------------
137 end
138 end
```

A.2.3 ClickCastling.m

```
1  %Castling Enables frontend implementation of castling
2  function [chessboard,piece_colour, num_moves,allowscheck]=ClickCastling(v1,v2,x_ori,
       y_ori,B,piece_colour,chessboard,...
3      num_moves,parameters,PM,handles,onlyAIoption,move_x,move_y,varargin)
4
5  %-------------------------------------------------------------------------
6  %                  Init values,conversions and click location
7  %-------------------------------------------------------------------------
8  if(mod(B.info.turn,2)==1)
9      colourturn = 119;
10     oppositecolour = 98;
11 else
12     colourturn = 98;
13     oppositecolour = 119;
14 end
15
16 if onlyAIoption == 0
17 clickP = get(gca,'CurrentPoint');
18       x = ceil(clickP(1,2));
19       y = ceil(clickP(1,1));
20 %----- Conversion from Graph grid to B.top grid ------------------------------
21       x = 13-x;
22       y = y + 4;
23 %---------Conversion from B.Top grid to Chessboard grid---------------------
24       p_x = x - 4; %p_x is necessary because it is the current clicked position
25       p_y = y - 4;
26       ori_x = x_ori - 4; %The difference is that ori_x is for chessboard,
27       ori_y = y_ori - 4; %x_ori is for B.top
28 else
29     p_x = move_x;   %Where is it moving to
30     p_y = move_y;
31     ori_x = x_ori;   %Where was it originally
32     ori_y = y_ori;
33 end
34
```

```
35 %-------------------------------------------------------------------------
36 %            Checks if King is exposed to check in any way
37 %-------------------------------------------------------------------------
38 %The method used is to create a future chessboard based on the move
39 %requested
40
41 fboard = chessboard;
42 f_p_colour= piece_colour;
43 f_num_moves = num_moves;
44 %This step officially moves the piece
45 fboard(p_x,p_y) = chessboard(ori_x,ori_y);
46 f_p_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
47 f_num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
48 %This step empties the previous box
49 fboard(ori_x,ori_y) = 0;
50 f_p_colour(ori_x,ori_y) = 0;
51 f_num_moves(ori_x,ori_y) = 0;
52
53 %Analyses the future board
54 [potentialfuturemoves,capt_index_future] = analyseboard(fboard,...
55     f_p_colour,f_num_moves,oppositecolour);
56 [allowscheck]=KingCheck(fboard,f_p_colour,colourturn,...
57     capt_index_future,potentialfuturemoves);
58 if allowscheck ==1 && onlyAIoption == 0
59     set(handles.gameconsole,'String','King will be left in check, move invalid')
60 end
61 %-------------------------------------------------------------------------
62 %Ensures it can only move legally
63 if PM(p_x,p_y)==4 && allowscheck==0
64
65 %-------------------------------------------------------------------------
66 %                   B.top
67 %-------------------------------------------------------------------------
68 %Coordinate system is X_rook = [B.top Chessboard]
69 if ( p_x == 8 && p_y == 7)
70     x_rook = [12 8]; %Initial Rook Position
71     y_rook = [12 8];
72     move_x = [12 8]; %Final Rook Position
73     move_y = [10 6];
74 elseif ( p_x == 8 && p_y == 3 )
75     x_rook = [12 8];
76     y_rook = [5 1];
77     move_x = [12 8];
78     move_y = [8 4];
79 elseif ( p_x == 1 && p_y == 7)
80     x_rook = [5 1];
81     y_rook = [12 8];
82     move_x = [5 1];
83     move_y = [10 6];
84 elseif ( p_x == 1 && p_y == 3)
85     x_rook = [5 1];
86     y_rook = [5 1];
87     move_x = [5 1];
88     move_y = [8 4];
89 end
90
91 B.info.turn = B.info.turn + 1;
92 %-------------------------------------------------------------------------
93 %              This is to edit the backend chessboard matrix
94 %-------------------------------------------------------------------------
95 %-----------------------------------King----------------------------------
96 %This step officially moves the piece
97 chessboard(p_x,p_y) = chessboard(ori_x,ori_y);
98 piece_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
99 num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
```

```
100
101 %This step empties the previous box
102 chessboard(ori_x,ori_y) = 0;
103 piece_colour(ori_x,ori_y) = 0;
104 num_moves(ori_x,ori_y) = 0;
105
106 %----------------------------------Rook-----------------------------------------
107 %This step officially moves the piece
108 chessboard(move_x(2),move_y(2)) = chessboard(x_rook(2),y_rook(2));
109 piece_colour(move_x(2),move_y(2)) = piece_colour(x_rook(2),y_rook(2));
110 num_moves(move_x(2),move_y(2)) = num_moves(x_rook(2),y_rook(2)) + 1;
111
112 %This step empties the previous box
113 chessboard(x_rook(2),y_rook(2)) = 0;
114 piece_colour(x_rook(2),y_rook(2)) = 0;
115 num_moves(x_rook(2),y_rook(2)) = 0;
116
117 %--------------Analyses for potential checks & provides game stats----------
118 [potentialmoves,capt_index] = analyseboard(chessboard,piece_colour,num_moves,colourturn)
      ;
119 [checkopp]=KingCheck(chessboard,piece_colour,oppositecolour,capt_index,potentialmoves);
120 if checkopp == 1 && onlyAIoption == 0
121     set(handles.checkstat,'String','Check')
122     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
123     if ischeckmate
124         set(handles.checkstat,'String','Checkmate, White Wins')
125     end
126     elseif checkopp == 0 && onlyAIoption ==0
127     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
128     if ischeckmate
129         set(handles.checkstat,'String','Stalemate')
130     else
131         set(handles.checkstat,'String','')
132     end
133 end
134
135 if onlyAIoption ==0
136     [B] = readchessboard(B,chessboard,piece_colour);
137 %-------------------------------------------------------------------------------
138 %                            Redraws the Board
139 %-------------------------------------------------------------------------------
140 icount=0;
141 for i=1:71
142          icount=icount+1;
143          if mod(i,2)==1
144              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
145                  parameters.dx ,parameters.dx],'Curvature',[0,0],...
146                  'FaceColor',[0.82 0.545 0.278])
147          else
148             rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
149                 parameters.dx ,parameters.dx],...
150                 'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
151          end
152 end
153
154 for r=1:parameters.rows
155     for c=1:parameters.cols
156         if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
157             % load the image
158             [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
159             % draw the image
160             imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
161                 mirrorImage(X),'AlphaData',mirrorImage(alpha),...
162                 'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
163                 num_moves,parameters,potentialmoves,handles});
```

```
164         end
165     end
166 end
167 drawnow;
168 if(get(handles.choice2,'Value')==1)
169     AIControl(B,piece_colour,chessboard,num_moves,parameters, handles);
170 end
171 if(get(handles.choice3,'Value')==1)
172     AIvsAI(B,piece_colour,chessboard,num_moves,parameters, handles)
173 end
174 if(get(handles.choice1,'Value')==1)
175     PlayerVsPlayer( B,piece_colour,chessboard,num_moves,parameters, handles )
176 end
177 end
178 %-------------------------------------------------------------------------------
179 end
180 end
```

A.2.4 ClickEnpassant.m

```
1  %Enpassant Enables frontend implementation of En Passant
2  function [chessboard,piece_colour, num_moves,allowscheck]=ClickEnpassant(v1,v2,x_ori,
       y_ori,B,piece_colour,chessboard,...
3      num_moves,parameters,PM, handles,onlyAIoption,move_x,move_y,varargin)
4
5  %-------------------------------------------------------------------------
6  %                  Init values,conversions and click location
7  %-------------------------------------------------------------------------
8  if(mod(B.info.turn,2)==1)
9      colourturn = 119;
10     oppositecolour = 98;
11 else
12     colourturn = 98;
13     oppositecolour = 119;
14 end
15
16 if onlyAIoption == 0
17 clickP = get(gca,'CurrentPoint');
18       x = ceil(clickP(1,2));
19       y = ceil(clickP(1,1));
20 %---- Conversion from Graph grid to B.top grid ----------------------------
21       x = 13-x;
22       y = y + 4;
23 %---------Conversion from B.Top grid to Chessboard grid--------------------
24       p_x = x - 4; %p_x is necessary because it is the current clicked position
25       p_y = y - 4;
26       ori_x = x_ori - 4; %The difference is that ori_x is for chessboard,
27       ori_y = y_ori - 4; %x_ori is for B.top
28 else
29     p_x = move_x;   %Where is it moving to
30     p_y = move_y;
31     ori_x = x_ori;   %Where was it originally
32     ori_y = y_ori;
33 end
34
35 %-------------------------------------------------------------------------
36 %        Checks if King is exposed to check in any way due to move
37 %-------------------------------------------------------------------------
38 %The method used is to create a future chessboard based on the move
39 %requested
40
41 fboard = chessboard;
```

```
42  f_p_colour= piece_colour;
43  f_num_moves = num_moves;
44  %This step officially moves the piece
45  fboard(p_x,p_y) = chessboard(ori_x,ori_y);
46  f_p_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
47  f_num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
48  %This step empties the previous box
49  fboard(ori_x,ori_y) = 0;
50  f_p_colour(ori_x,ori_y) = 0;
51  f_num_moves(ori_x,ori_y) = 0;
52
53  %Analyses the future board
54  [potentialfuturemoves,capt_index_future] = analyseboard(fboard,...
55      f_p_colour,f_num_moves,oppositecolour);
56  [allowscheck]=KingCheck(fboard,f_p_colour,colourturn,...
57      capt_index_future,potentialfuturemoves);
58  if allowscheck==1 && onlyAIoption == 0
59      set(handles.gameconsole,'String','King will be left in check, move invalid')
60  end
61  %--------------------------------------------------------------------------
62  %Ensures it can only move legally
63  if PM(p_x,p_y)==3 && allowscheck==0
64  %--------------------------------------------------------------------------
65  %                Moves Data in B.TOP & deletes previous cell
66  %--------------------------------------------------------------------------
67  %Coordinates of the captured piece
68  if (piece_colour(ori_x,ori_y)==98)
69     del_x = [p_x+3 p_x-1];
70     del_y = [p_y+4 p_y];
71  end
72
73  if (piece_colour(ori_x,ori_y)==119)
74     del_x = [p_x+5 p_x+1];
75     del_y = [p_y+4 p_y];
76  end
77
78  B.info.turn = B.info.turn + 1;
79  %--------------------------------------------------------------------------
80  %              This is to edit the backend chessboard matrix
81  %--------------------------------------------------------------------------
82  %This step officially moves the piece
83  chessboard(p_x,p_y) = chessboard(ori_x,ori_y);
84  piece_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
85  num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
86
87  %This step empties the previous box
88  chessboard(ori_x,ori_y) = 0;
89  piece_colour(ori_x,ori_y) = 0;
90  num_moves(ori_x,ori_y) = 0;
91
92  %This step deletes the capured piece
93  chessboard(del_x(2),del_y(2)) = 0;
94  piece_colour(del_x(2),del_y(2)) = 0;
95  num_moves(del_x(2),del_y(2)) = 0;
96
97
98  %---------------Analyses for potential checks & provides game stats----------
99  [potentialmoves,capt_index] = analyseboard(chessboard,piece_colour,num_moves,colourturn)
        ;
100 [checkopp]=KingCheck(chessboard,piece_colour,oppositecolour,capt_index,potentialmoves);
101 if checkopp == 1 && onlyAIoption == 0
102     set(handles.checkstat,'String','Check')
103     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
104     if ischeckmate
105         set(handles.checkstat,'String','Checkmate, White Wins')
```

```
    end
    elseif checkopp == 0 && onlyAIoption ==0
    [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
    if ischeckmate
        set(handles.checkstat,'String','Stalemate')
    else
        set(handles.checkstat,'String','')
    end
end

if onlyAIoption == 0
    [B] = readchessboard(B,chessboard,piece_colour);
%-------------------------------------------------------------------------
%                           Redraws the Board
%-------------------------------------------------------------------------
icount=0;
for i=1:71
         icount=icount+1;
         if mod(i,2)==1
             rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
                 parameters.dx ,parameters.dx],'Curvature',[0,0],...
                 'FaceColor',[0.82 0.545 0.278])
         else
            rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
                parameters.dx ,parameters.dx],...
                'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
         end
end

for r=1:parameters.rows
    for c=1:parameters.cols
        if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
            % load the image
            [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
            % draw the image
            imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
                mirrorImage(X),'AlphaData',mirrorImage(alpha),...
                'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
                num_moves,parameters,potentialmoves,handles});
        end
    end
end
drawnow;
if(get(handles.choice2,'Value')==1)
    AIControl(B,piece_colour,chessboard,num_moves,parameters, handles);
end
if(get(handles.choice3,'Value')==1)
    AIvsAI(B,piece_colour,chessboard,num_moves,parameters, handles)
end
if(get(handles.choice1,'Value')==1)
    PlayerVsPlayer( B,piece_colour,chessboard,num_moves,parameters, handles )
end
end
%-------------------------------------------------------------------------
end
end
```

A.2.5 ClickPawnPromo.m

```
%PawnPromo Enables Front End Implementation of Pawn Promo
function [chessboard,piece_colour, num_moves,allowscheck]=ClickPawnPromo(v1,v2,x_ori,
    y_ori,B,piece_colour,chessboard,...
```

```
3       num_moves,parameters,PM,handles,onlyAIoption,move_x,move_y,promo,varargin)
4
5   %-------------------------------------------------------------------------------
6   %                   Init values,conversions and click location
7   %-------------------------------------------------------------------------------
8   if(mod(B.info.turn,2)==1)
9       colourturn = 119;
10      oppositecolour = 98;
11  else
12      colourturn = 98;
13      oppositecolour = 119;
14  end
15
16  if onlyAIoption == 0
17  clickP = get(gca,'CurrentPoint');
18        x = ceil(clickP(1,2));
19        y = ceil(clickP(1,1));
20  %------ Conversion from Graph grid to B.top grid ------------------------------
21        x = 13-x;
22        y = y + 4;
23  %---------Conversion from B.Top grid to Chessboard grid-----------------------
24        p_x = x - 4; %p_x is necessary because it is the current clicked position
25        p_y = y - 4;
26        ori_x = x_ori - 4; %The difference is that ori_x is for chessboard,
27        ori_y = y_ori - 4; %x_ori is for B.top
28  else
29      p_x = move_x;   %Where is it moving to
30      p_y = move_y;
31      ori_x = x_ori;   %Where was it originally
32      ori_y = y_ori;
33  end
34
35  %-------------------------------------------------------------------------------
36  %            Checks if King is exposed to check in any way
37  %-------------------------------------------------------------------------------
38  %The method used is to create a future chessboard based on the move
39  %requested
40
41  fboard = chessboard;
42  f_p_colour= piece_colour;
43  f_num_moves = num_moves;
44  %This step officially moves the piece
45  fboard(p_x,p_y) = chessboard(ori_x,ori_y);
46  f_p_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
47  f_num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
48  %This step empties the previous box
49  fboard(ori_x,ori_y) = 0;
50  f_p_colour(ori_x,ori_y) = 0;
51  f_num_moves(ori_x,ori_y) = 0;
52
53  %Analyses the future board
54  [potentialfuturemoves,capt_index_future] = analyseboard(fboard,...
55      f_p_colour,f_num_moves,oppositecolour);
56  [allowscheck]=KingCheck(fboard,f_p_colour,colourturn,...
57      capt_index_future,potentialfuturemoves);
58  if allowscheck==1 && onlyAIoption == 0
59      set(handles.gameconsole,'String','King will be left in check, move invalid')
60  end
61  %-------------------------------------------------------------------------------
62  %Ensures it can only move legally
63  if PM(p_x,p_y)==5 && allowscheck==0
64  %-------------------------------------------------------------------------------
65  %                 Moves Data in B.TOP & deletes previous cell
66  %-------------------------------------------------------------------------------
67  %Allows user to input desired piece. Checks legality.
```

```
68  if ~onlyAIoption
69  set(handles.gameconsole,'String','Pawn has been promoted');
70              flags=0;
71            while(flags==0)
72                flags=1;
73                v=0;
74                while v == 0
75                [pawn_prom,v] = listdlg('PromptString','Select a piece:',...
76                  'SelectionMode','single',...
77                  'ListString',{'Rook','Queen','Knight','Bishop'});
78                end
79              switch pawn_prom
80                  case 1
81                      chessboard(p_x,p_y)= 5;
82                  case 2
83                      chessboard(p_x,p_y)= 9;
84                  case 3
85                      chessboard(p_x,p_y)= 3;
86                  case 4
87                      chessboard(p_x,p_y)= 4;
88                  otherwise
89                      disp('Invalid input');
90                      flags=0;
91              end
92            end
93  else
94              switch promo
95                          case 'rook'
96                              chessboard(p_x,p_y)= 5;
97                          case 'queen'
98                              chessboard(p_x,p_y)= 9;
99                          case 'knight'
100                             chessboard(p_x,p_y)= 3;
101                         case 'bishop'
102                             chessboard(p_x,p_y)= 4;
103             end
104 end
105
106 B.info.turn = B.info.turn + 1;
107
108 %-------------------------------------------------------------------------
109 %               This is to edit the backend chessboard matrix
110 %-------------------------------------------------------------------------
111 %This step officially moves the piece
112 num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
113 piece_colour(p_x,p_y)= colourturn;
114
115 %This step empties the previous box
116 chessboard(ori_x,ori_y) = 0;
117 piece_colour(ori_x,ori_y) = 0;
118 num_moves(ori_x,ori_y) = 0;
119
120 %---------------Analyses for potential checks & provides game stats----------
121 [potentialmoves,capt_index] = analyseboard(chessboard,piece_colour,num_moves,colourturn)
        ;
122 [checkopp]=KingCheck(chessboard,piece_colour,oppositecolour,capt_index,potentialmoves);
123 if checkopp == 1 && onlyAIoption == 0
124    set(handles.checkstat,'String','Check')
125     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
126     if ischeckmate
127         set(handles.checkstat,'String','Checkmate, White Wins')
128     end
129 elseif checkopp == 0 && onlyAIoption ==0
130     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
131     if ischeckmate
```

```
132         set(handles.checkstat,'String','Stalemate')
133     else
134         set(handles.checkstat,'String','')
135     end
136 end
137
138 if onlyAIoption == 0
139     [B] = readchessboard(B,chessboard,piece_colour);
140 %-------------------------------------------------------------------------
141 %                           Redraws the Board
142 %-------------------------------------------------------------------------
143 icount=0;
144 for i=1:71
145          icount=icount+1;
146          if mod(i,2)==1
147              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
148                  parameters.dx ,parameters.dx],'Curvature',[0,0],...
149                  'FaceColor',[0.82 0.545 0.278])
150          else
151             rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
152                 parameters.dx ,parameters.dx],...
153                 'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
154          end
155 end
156
157 for r=1:parameters.rows
158     for c=1:parameters.cols
159         if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
160             % load the image
161             [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
162             % draw the image
163             imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
164                 mirrorImage(X),'AlphaData',mirrorImage(alpha),...
165                 'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
166                 num_moves,parameters,potentialmoves,handles});
167         end
168     end
169 end
170 drawnow;
171 if(get(handles.choice2,'Value')==1)
172     AIControl(B,piece_colour,chessboard,num_moves,parameters, handles);
173 end
174 if(get(handles.choice3,'Value')==1)
175     AIvsAI(B,piece_colour,chessboard,num_moves,parameters, handles)
176 end
177 if(get(handles.choice1,'Value')==1)
178     PlayerVsPlayer( B,piece_colour,chessboard,num_moves,parameters, handles )
179 end
180 end
181 %-------------------------------------------------------------------------
182 end
183 end
```

A.2.6 ClickMovePiece.m

```
1  %Movepiece Part of the Click Series of Functions - Enables movement
2  function [chessboard,piece_colour, num_moves,allowscheck]=ClickMovePiece(v1,v2,x_ori,
       y_ori,B,piece_colour,chessboard,...
3      num_moves,parameters,PM,handles,onlyAIoption,move_x,move_y,varargin)
4
5  %-------------------------------------------------------------------------
6  %                   Init values,conversions and click location
```

```
7  %-------------------------------------------------------------------------
8  if(mod(B.info.turn,2)==1)
9      colourturn = 119;
10     oppositecolour = 98;
11 else
12     colourturn = 98;
13     oppositecolour = 119;
14 end
15
16 if onlyAIoption == 0
17 clickP = get(gca,'CurrentPoint');
18       x = ceil(clickP(1,2));
19       y = ceil(clickP(1,1));
20 %------ Conversion from Graph grid to B.top grid ------------------------------
21       x = 13-x;
22       y = y + 4;
23 %---------Conversion from B.Top grid to Chessboard grid---------------------
24       p_x = x - 4; %p_x is necessary because it is the current clicked position
25       p_y = y - 4;
26       ori_x = x_ori - 4; %The difference is that ori_x is for chessboard,
27       ori_y = y_ori - 4; %x_ori is for B.top
28 else
29     p_x = move_x;   %Where is it moving to
30     p_y = move_y;
31     ori_x = x_ori;   %Where was it originally
32     ori_y = y_ori;
33 end
34
35 %-------------------------------------------------------------------------
36 %            Checks if King is exposed to check in any way
37 %-------------------------------------------------------------------------
38 %The method used is to create a future chessboard based on the move
39 %requested
40
41 fboard = chessboard;
42 f_p_colour= piece_colour;
43 f_num_moves = num_moves;
44 %This step officially moves the piece
45 fboard(p_x,p_y) = chessboard(ori_x,ori_y);
46 f_p_colour(p_x,p_y) = piece_colour(ori_x,ori_y);
47 f_num_moves(p_x,p_y) = num_moves(ori_x,ori_y) + 1;
48 %This step empties the previous box
49 fboard(ori_x,ori_y) = 0;
50 f_p_colour(ori_x,ori_y) = 0;
51 f_num_moves(ori_x,ori_y) = 0;
52
53 %Analyses the future board
54 [potentialfuturemoves,capt_index_future] = analyseboard(fboard,...
55     f_p_colour,f_num_moves,oppositecolour);
56 [allowscheck]=KingCheck(fboard,f_p_colour,colourturn,...
57     capt_index_future,potentialfuturemoves);
58 if allowscheck ==1 && onlyAIoption == 0
59     set(handles.gameconsole,'String','King will be left in check, move invalid')
60 end
61 %-------------------------------------------------------------------------
62 %-------------------------------------------------------------------------
63
64 if PM(p_x,p_y)==1 && allowscheck==0 %Ensures it can only move legally
65
66 %Iterates the turn
67 B.info.turn = B.info.turn + 1;
68
69 %-------------------------------------------------------------------------
70 %              This is to edit the backend chessboard matrix
71 %-------------------------------------------------------------------------
```

```
72  %This step officially moves the piece
73  chessboard = fboard;
74  piece_colour = f_p_colour;
75  num_moves = f_num_moves;
76
77  %--------------Analyses for potential checks & provides game stats----------
78  [potentialmoves,capt_index] = analyseboard(chessboard,piece_colour,num_moves,colourturn)
        ;
79  [checkopp]=KingCheck(chessboard,piece_colour,oppositecolour,capt_index,potentialmoves);
80
81  if checkopp == 1 && onlyAIoption ==0
82      set(handles.checkstat,'String','Check')
83      [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
84      if ischeckmate
85          set(handles.checkstat,'String','Checkmate, White Wins')
86      end
87  elseif checkopp == 0 && onlyAIoption ==0
88      [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
89      if ischeckmate
90          set(handles.checkstat,'String','Stalemate')
91      else
92          set(handles.checkstat,'String','')
93      end
94  end
95
96  %-------------------------------------------------------------------------
97  if onlyAIoption == 0
98      [B] = readchessboard(B,chessboard,piece_colour);
99  %-------------------------------------------------------------------------
100 %                            Redraws the Board
101 %-------------------------------------------------------------------------
102 icount=0;
103 for i=1:71
104           icount=icount+1;
105           if mod(i,2)==1
106               rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
107                   parameters.dx ,parameters.dx],'Curvature',[0,0],...
108                   'FaceColor',[0.82 0.545 0.278])
109           else
110              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
111                  parameters.dx ,parameters.dx],...
112                  'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
113           end
114 end
115
116 %-------------------------------------------------------------------------
117 for r=1:parameters.rows
118     for c=1:parameters.cols
119         if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
120             % load the image
121             [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
122             % draw the image
123             imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
124                 mirrorImage(X),'AlphaData',mirrorImage(alpha),...
125                 'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
126                 num_moves,parameters,potentialmoves,handles});
127         end
128     end
129 end
130 drawnow;
131 if(get(handles.choice2,'Value')==1)
132     AIControl(B,piece_colour,chessboard,num_moves,parameters, handles);
133 end
134 if(get(handles.choice3,'Value')==1)
135     AIvsAI(B,piece_colour,chessboard,num_moves,parameters, handles)
```

```
136 end
137 if(get(handles.choice1,'Value')==1)
138     PlayerVsPlayer( B,piece_colour,chessboard,num_moves,parameters, handles )
139 end
140 end
141 %-------------------------------------------------------------------------
142 end
143 end
```

A.3 AI

A.3.1 AIControl.m

```
1  function [B,piece_colour,chessboard,num_moves,parameters, handles]=AIControl(B,
       piece_colour,chessboard,...
2                   num_moves,parameters, handles)
3  %AIControl Enables AI to be in action
4
5  %-------------------------------------------------------------------------
6  %                         Init Values
7  %-------------------------------------------------------------------------
8  if(mod(B.info.turn-1,2)==1)
9      colourturn = 119;
10     oppositecolour = 98;
11 else
12     colourturn = 98;
13     oppositecolour = 119;
14 end
15
16 [userboardscore] = heuristicanalysis(B,chessboard, piece_colour,num_moves,119,handles);
17 set(handles.UPS,'String',userboardscore)
18 handles.userboardscore = [handles.userboardscore userboardscore];
19 depth = 2;
20 set(handles.depth,'String',depth)
21
22 %-------------------- Stops Game Execution if White Wins ---------------------
23 % [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
24 % if ischeckmate
25 %     return
26 % end
27
28 [oppcolourpotentialmoves,oppcolourcapt_index] = analyseboard(chessboard, piece_colour,
       num_moves,colourturn);
29
30 [ischeck]=KingCheck(chessboard,piece_colour,oppositecolour, oppcolourcapt_index,
       oppcolourpotentialmoves);
31 if ischeck == 1
32     set(handles.checkstat,'String','Check')
33     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
34     if ischeckmate
35         set(handles.checkstat,'String','Checkmate, White Wins')
36     end
37 elseif ischeck == 0
38     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
39     if ischeckmate
40         set(handles.checkstat,'String','Stalemate')
41     else
42         set(handles.checkstat,'String','')
43     end
44 end
45 %----------------------Plot UserBoardScore-----------------------------------
46
```

```
47  handles.turnforwhite = [handles.turnforwhite B.info.turn];
48  plot(handles.graph,handles.turnforwhite,handles.userboardscore,'-b',...
49      handles.turnforblack,handles.AIBoardscore,'-r','LineWidth',2)
50  set(handles.graph,'XColor','w','YColor','w')
51  xlabel(handles.graph,'Turn')
52  ylabel(handles.graph,'Score')
53
54 %--------------------------------------------------------------------------
55 set(handles.AIMsgs,'String','Thinking Really Hard')
56
57 %Produces AI's decision
58 tic
59 [boardscore,chessboard,piece_colour,num_moves]=...
60     AI_GenerateAllMoves(B,chessboard,piece_colour,num_moves,depth,1,-99999,99999,handles
          );
61 time =toc;
62
63 set(handles.AIMsgs,'String',['Time Taken To Think Was: ' num2str(time) ' seconds'])
64
65 %Translates the results into B.top
66 [B] = readchessboard(B,chessboard,piece_colour);
67 %Iterates turn
68 B.info.turn = B.info.turn + 1;
69
70 %------------------------ Shows AI Board Score----------------------------------
71 [AIBoardScore] = heuristicanalysis(B,chessboard, piece_colour,num_moves,98,handles);
72 set(handles.APS,'String',AIBoardScore)
73 handles.AIBoardscore = [handles.AIBoardscore AIBoardScore];
74
75 %------------------------Plots AI Board Score----------------------------------
76
77  handles.turnforblack = [handles.turnforblack B.info.turn];
78  plot(handles.graph,handles.turnforwhite,handles.userboardscore,'-b',...
79      handles.turnforblack,handles.AIBoardscore,'-r','LineWidth',2)
80  set(handles.graph,'XColor','w','YColor','w')
81  xlabel(handles.graph,'Turn')
82  ylabel(handles.graph,'Score')
83
84 %--------------------------- Checks if AI has checkmated User -----------------
85 [oppcolourpotentialmoves,oppcolourcapt_index] = analyseboard(chessboard, piece_colour,
       num_moves,oppositecolour);
86
87 [ischeck]=KingCheck(chessboard,piece_colour,colourturn, oppcolourcapt_index,
       oppcolourpotentialmoves);
88 if ischeck == 1
89     set(handles.checkstat,'String','Check')
90     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
91     if ischeckmate
92         set(handles.checkstat,'String','Checkmate, Black Wins')
93     end
94 elseif ischeck == 0
95     [ischeckmate]=checkmate(B,chessboard,piece_colour, num_moves);
96     if ischeckmate
97         set(handles.checkstat,'String','Stalemate')
98     else
99         set(handles.checkstat,'String','')
100    end
101 end
102 %--------------------------------------------------------------------------
103
104 %--------------------------------------------------------------------------
105 %                            Redraws the Board
106 %--------------------------------------------------------------------------
107 icount=0;
108 for i=1:71
```

```
109           icount=icount+1;
110           if mod(i,2)==1
111               rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
112                   parameters.dx ,parameters.dx],'Curvature',[0,0],...
113                   'FaceColor',[0.82 0.545 0.278])
114           else
115              rectangle('Position',[parameters.xx(icount),parameters.yy(icount),...
116                  parameters.dx ,parameters.dx],...
117                  'Curvature',[0,0],'FaceColor',[1 0.808 0.62])
118           end
119   end
120
121   %-------------------------------------------------------------------------
122   for r=1:parameters.rows
123       for c=1:parameters.cols
124           if ~isempty(B.top(r+B.info.pad/2,c+B.info.pad/2).image)
125               % load the image
126               [X, map, alpha]  = imread(B.top(r+B.info.pad/2,c+B.info.pad/2).image);
127               % draw the image
128               imHdls(r,c) = image(c+[0 1]-1,[parameters.rows-1 parameters.rows]-r+1,...
129                   mirrorImage(X),'AlphaData',mirrorImage(alpha),...
130                   'ButtonDownFcn',{@ClickPiece,B,piece_colour,chessboard,...
131                   num_moves,parameters,oppcolourpotentialmoves,handles});
132           end
133       end
134   end
135   drawnow;
136   %-------------------------------------------------------------------------
137   end
```

A.3.2 AI_GenerateAllMoves.m

```
1   %AI - Generates moves and stores them for 1 PLY (Only for DATA Tree)
2   function [boardscore,bchessboard,bpiece_colour,bnum_moves,handles]=...
3       AI_GenerateAllMoves(B,chessboard,piece_colour,num_moves,depth,maxormin,alpha,beta,
            handles)
4   %-------------------------------------------------------------------------
5   %                            Init Values
6   %-------------------------------------------------------------------------
7   TmpB = B;
8
9   if(mod(TmpB.info.turn,2)==1)
10      colour = 119;
11      oppcolour = 98;
12  else
13      colour = 98;
14      oppcolour = 119;
15  end
16
17  TmpB.info.turn = TmpB.info.turn +1;
18  %-------------------------------------------------------------------------
19
20  if depth == 0
21     TmpB.info.turn = TmpB.info.turn-1;
22    [boardscore] = heuristicanalysis(TmpB,chessboard, piece_colour,num_moves,colour,
          handles);
23    bchessboard = chessboard;
24    bpiece_colour = piece_colour;
25    bnum_moves = num_moves;
26  else
27
28  if maxormin == 1  %Maximizing Player
```

```
%==================== Generates Future Nodes or Leafs ====================
%-------------------------------------------------------------------------
%        Loop that generates all possible moves
%-------------------------------------------------------------------------
[p_x,p_y] = find(piece_colour == colour);
perm_index = randperm(length(p_x));
p_x = p_x(perm_index);
p_y = p_y(perm_index);
n_remaining = length(p_x);
[potentialmoves] = analyseboard(chessboard, piece_colour,num_moves,oppcolour);
previousboardscore = -99999;
%In essence, we are going through each piece, looking at it's possible
%moves, make those possible moves, evaluate, save bestboard.
for i=1:n_remaining
    p_type = chessboard(p_x(i),p_y(i));
    switch p_type
        case 1
            [move] = MovementPawn(chessboard,piece_colour,num_moves,p_x(i),p_y(i));
        case 5
            [move] = MovementRook(chessboard,piece_colour,p_x(i),p_y(i));
        case 4
            [move] = MovementBishop(chessboard,piece_colour,p_x(i),p_y(i));
        case 3
            [move] = MovementKnight(chessboard,piece_colour,p_x(i),p_y(i));
        case 9
            [move] = MovementQueen(chessboard,piece_colour,p_x(i),p_y(i));
        case 10
            [move] = MovementKing(chessboard,piece_colour,num_moves,potentialmoves,p_x(i
                ),p_y(i));
    end

%-------------------------------------------------------------------------
%            Individual Piece Moves That Generate New Game States
%                   Recursion is also added in each loop
%-------------------------------------------------------------------------
     [move_x,move_y] = find(move ~= 0);
     perm_index2 = randperm(length(move_x));
     move_x = move_x(perm_index2);
     move_y = move_y(perm_index2);
     n_move = length(move_x);
     pruneflag = 0;
%This loop generates all the game states from 1 piece
     for j = 1:n_move
         switch move(move_x(j),move_y(j))
             case 1
                 [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickMovePiece
                     (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
                     num_moves,0,move,0,1,move_x(j),move_y(j));
             case 2
                 [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCapturePiece
                     (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
                     num_moves,0,move,0,1,move_x(j),move_y(j));
             case 3
                 [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickEnpassant
                     (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
                     num_moves,0,move, 0,1,move_x(j),move_y(j));
             case 4
                 [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCastling(0,0,
                     p_x(i),p_y(i),B,piece_colour,chessboard,...
                     num_moves,0,move,0,1,move_x(j),move_y(j));
             case 5
                 [pchessboard,ppiece_colour, pnum_moves,kingincheck]=ClickPawnPromo(0,0,
                     p_x(i),p_y(i),B,piece_colour,chessboard,...
                     num_moves,0,move,0,1,move_x(j),move_y(j),'queen');
         end
```

```
88  %————————A node has been generated, what do you want to do with it?————————
89            if kingincheck
90                %ignore because move not valid
91                if ~exist('boardscore','var')
92                boardscore = -99999;
93                bchessboard = 0;
94                bpiece_colour =0;
95                bnum_moves =0;
96                end
97            else
98                    %Generate another layer with recursive parameters
99                    [boardscore,~,~,~,handles]=...
100               AI_GenerateAllMoves(TmpB,pchessboard,ppiece_colour,pnum_moves,depth-1,-
                      maxormin,alpha,beta,handles);
101
102                       if boardscore > previousboardscore
103                           previousboardscore = boardscore;
104                           bchessboard = pchessboard;
105                           bpiece_colour = ppiece_colour;
106                           bnum_moves = pnum_moves;
107                       end
108                       if boardscore>alpha
109                           alpha = boardscore;
110                       end
111 %              disp([depth alpha beta boardscore previousboardscore i j n_remaining
        n_move])
112                       if alpha>beta
113                           pruneflag = 1;
114                           break
115                       end
116           end
117 %————————————————————————————————————————————————————————————————————————
118       end
119       if pruneflag
120           break
121       end
122 end
123 %========================================================================
124
125
126 elseif maxormin == -1 %Minimizing Player
127 %====================== Generates Future Nodes or Leafs ====================
128 %————————————————————————————————————————————————————————————————————————
129 %        Loop that generates all possible moves
130 %————————————————————————————————————————————————————————————————————————
131 [p_x,p_y] = find(piece_colour == colour);
132 perm_index = randperm(length(p_x));
133 p_x = p_x(perm_index);
134 p_y = p_y(perm_index);
135 n_remaining = length(p_x);
136 [potentialmoves] = analyseboard(chessboard, piece_colour,num_moves,oppcolour);
137 previousboardscore = 99999;
138 %In essence, we are going through each piece, looking at it's possible
139 %moves, make those possible moves, evaluate, save bestboard.
140 for i=1:n_remaining
141     p_type = chessboard(p_x(i),p_y(i));
142     switch p_type
143         case 1
144             [move] = MovementPawn(chessboard,piece_colour,num_moves,p_x(i),p_y(i));
145         case 5
146             [move] = MovementRook(chessboard,piece_colour,p_x(i),p_y(i));
147         case 4
148             [move] = MovementBishop(chessboard,piece_colour,p_x(i),p_y(i));
149         case 3
150             [move] = MovementKnight(chessboard,piece_colour,p_x(i),p_y(i));
```

```
151          case 9
152              [move] = MovementQueen(chessboard,piece_colour,p_x(i),p_y(i));
153          case 10
154              [move] = MovementKing(chessboard,piece_colour,num_moves,potentialmoves,p_x(i
                     ),p_y(i));
155      end
156
157  %-------------------------------------------------------------------------
158  %             Individual Piece Moves That Generate New Game States
159  %                     Recursion is also added in each loop
160  %-------------------------------------------------------------------------
161       [move_x,move_y] = find(move ~= 0);
162       perm_index2 = randperm(length(move_x));
163       move_x = move_x(perm_index2);
164       move_y = move_y(perm_index2);
165       n_move = length(move_x);
166       pruneflag = 0;
167  %This loop generates all the game states from 1 piece
168       for j = 1:n_move
169           switch move(move_x(j),move_y(j))
170               case 1
171                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickMovePiece
                           (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
172                        num_moves,0,move,0,1,move_x(j),move_y(j));
173               case 2
174                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCapturePiece
                           (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
175                        num_moves,0,move,0,1,move_x(j),move_y(j));
176               case 3
177                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickEnpassant
                           (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
178                        num_moves,0,move, 0,1,move_x(j),move_y(j));
179               case 4
180                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCastling(0,0,
                           p_x(i),p_y(i),B,piece_colour,chessboard,...
181                        num_moves,0,move,0,1,move_x(j),move_y(j));
182               case 5
183                   [pchessboard,ppiece_colour, pnum_moves,kingincheck]=ClickPawnPromo(0,0,
                           p_x(i),p_y(i),B,piece_colour,chessboard,...
184                        num_moves,0,move,0,1,move_x(j),move_y(j),'queen');
185           end
186  %----------A node has been generated, what do you want to do with it?----------
187           if kingincheck
188               %ignore because move not valid
189               if ~exist('boardscore','var')
190               boardscore = 99999;
191               bchessboard = 0;
192               bpiece_colour =0;
193               bnum_moves =0;
194               end
195           else
196                   %Generate another layer with recursive parameters
197                   [boardscore,~,~,~,handles]=...
198              AI_GenerateAllMoves(TmpB,pchessboard,ppiece_colour,pnum_moves,depth-1,-
                     maxormin,alpha,beta,handles);
199
200                   if boardscore < previousboardscore
201                       previousboardscore = boardscore;
202                       bchessboard = pchessboard;
203                       bpiece_colour = ppiece_colour;
204                       bnum_moves = pnum_moves;
205                   end
206                   if boardscore<beta
207                       beta = boardscore;
208                   end
```

```
209
210 % disp([depth alpha beta boardscore previousboardscore i j n_remaining n_move])
211                 if alpha>beta
212                     pruneflag = 1;
213                     break
214                 end
215             end
216         end
217          if pruneflag
218             break
219         end
220 %--------------------------------------------------------------------------
221 end
222 %==========================================================================
223 end % For if maxormin
224 end % For DEPTH if condition
225 end %For Function
```

A.3.3 heuristicanalysis.m

```
1  function [boardscore] = heuristicanalysis(B,chessboard, piece_colour,num_moves,
       currentcolour,handles)
2  %Colour should be the side in which it is being analysed for
3
4  %--------------------------------------------------------------------------
5  %                         Init Values
6  %--------------------------------------------------------------------------
7  if currentcolour == 119
8      oppcolour = 98;
9  else
10     oppcolour = 119;
11 end
12 %Generates potential moves of the currently investigated game state colour
13 [potentialmoves,capt_index] = analyseboard(chessboard, piece_colour,num_moves,
       currentcolour);
14 %Generates potential moves of the opponent
15 [oppcolourpotentialmoves, oppcolourcapt_index] = analyseboard(chessboard, piece_colour,
       num_moves,oppcolour);
16
17 %Finds the locations of own pieces and opponent's pieces
18 piece_index = find(piece_colour==currentcolour);
19 opp_piece_index = find(piece_colour==oppcolour);
20
21 %--------------------------------------------------------------------------
22
23 %---------------------Capture Analysis-------------------------------------
24 %A move is good because it opens up capture possibilities
25 num_pot_capture = length(capt_index); %Number of potential Captures
26 capt_value_sum = sum(chessboard(capt_index)); %The total capture value
27
28 %A move is good if it increases the number of capture
29 capt_value_diff = 51 - sum(chessboard(opp_piece_index));
30
31 %--------------------- Moves Analysis ---------------------------------------
32 %A move is good because it opens up space for other pieces to move
33 nocapture = potentialmoves;
34 nocapture(capt_index) = 0;
35 num_moves_available = sum(sum(nocapture));
36
37 %--------------------- Threats ----------------------------------------------
38 %If the move causes other pieces to be under threat, the move is worse.
39 opp_num_pot_capture = length(oppcolourcapt_index);
```

```
40  opp_capt_value_sum = sum(chessboard(oppcolourcapt_index));
41
42  %———————————— Number of own pieces ————————————————————————————
43  %A move is good if it prevents the number of own pieces from decreasing.
44  own_piece_sum_diff = 51 — sum(chessboard(piece_index));
45
46
47  %———————————— Control of centre space ————————————————————————
48  %A move is good if it increases control of the centre of the board
49  centre_piece=zeros(8,8);
50  centre_piece([28 29 36 37])=chessboard([28 29 36 37]);
51  centre_piece = centre_piece~=0;
52  centre_space_sum =  sum(centre_piece(piece_index));
53
54
55  %———————————— Own King Checked? ——————————————————————————————
56  %Checks if own king is in check. If in check, also checks if its a checkmate
57  own_ischeck = KingCheck(chessboard,piece_colour,currentcolour,oppcolourcapt_index,
        oppcolourpotentialmoves);
58  if own_ischeck==1
59      own_ischeckmate = checkmate(B,chessboard,piece_colour,num_moves);
60      else own_ischeckmate = 0;
61  end
62
63  %———————————————————— Castling? ——————————————————————————————
64  %Checks if castling has taken place
65  rook_pos = find(chessboard==5 & piece_colour==currentcolour);
66  king_pos = find(chessboard==10 & piece_colour==currentcolour);
67  castle = 0;
68
69  if currentcolour == 98 %Black case
70      if (king_pos==49 && ismember(41,rook_pos) && num_moves(41)==1 && num_moves(49)==1)
71          castle = 1;
72      elseif (king_pos==17 && ismember(25,rook_pos) && num_moves(25)==1 && num_moves(17)
            ==1)
73              castle = 1;
74      end
75
76  else %White case
77      if (king_pos==56 && ismember(48,rook_pos) && num_moves(48)==1 && num_moves(56)==1)
78          castle = 1;
79      elseif (king_pos==24 && ismember(32,rook_pos) && num_moves(32)==1 && num_moves(24)
            ==1)
80              castle = 1;
81      end
82  end
83
84  %———————————————— Opponent Checkmate?————————————————————————————
85  %Checks if opponent king is in check. If in check, also checks if its a checkmate
86  opp_ischeck = KingCheck(chessboard,piece_colour,oppcolour,capt_index,potentialmoves);
87  if opp_ischeck==1
88      opp_ischeckmate = checkmate(B,chessboard,piece_colour,num_moves);
89  else opp_ischeckmate = 0;
90  end
91
92  %———————————— Possibility of opponenet's promotion? ——————————————
93  %A move is bad if it brings opponent's pawn closer to the end of the board for promotion
        .
94  pawn_index = find(chessboard==1 & piece_colour==oppcolour);
95  if oppcolour == 98 %Black case
96      end_dist = 8—rem(pawn_index,8);
97      sum_opp_pawn_dist = sum(end_dist==0) + 0.5*sum(end_dist==1);
98  else %White case
99      end_dist = rem(pawn_index,8)—1;
100     sum_opp_pawn_dist = sum(end_dist==0) + 0.5*sum(end_dist==1);
```

```
101 end
102
103 %————————————————— Possibility of own promotion? ————————————————————
104 %A move is good if it brings own pawn closer to the end of the board for promotion.
105 pawn_index = find(chessboard==1 & piece_colour==currentcolour);
106 if currentcolour == 98 %Black case
107     end_dist = 8-rem(pawn_index,8);
108     sum_own_pawn_dist = sum(end_dist==0) + 0.5*sum(end_dist==1);
109 else %White case
110     end_dist = rem(pawn_index,8)-1;
111     sum_own_pawn_dist = sum(end_dist==0) + 0.5*sum(end_dist==1);
112 end
113
114 %————————————— Gain Factor for Hard————————————————————————————
115 if(get(handles.setHard,'Value')==1)
116 gainCapture = 3;  %Encourages AI to position a piece such that it can capture more
        pieces in the next move
117 gainMoves = 10; %Encourages AI to position such that it opens space for other pieces
118 gainThreats = -4; %Discourages AI to make moves that will lead to threats
119 gainOpppieces = 25; %Encourages to make moves that decrease opponents pieces
120 gainOwnpieces = -5; %Discourages AI from making moves that decrease own pieces
121 gainCentre = 1; %Encourages AI to increase control of centre space
122 gainOwnprom = 1; %Encourages AI to promote own pawns close to the end of the board
123 gainOppprom = -10; %Discourages AI to promote opponent's pawns
124 end
125 %————————————————————Gain Factor for Easy ————————————————————————
126 if(get(handles.setEasy,'Value')==1)
127 gainCapture = 2;  %Encourages AI to position a piece such that it can capture more
        pieces in the next move
128 gainMoves = 10; %Encourages AI to position such that it opens space for other pieces
129 gainThreats = -2; %Discourages AI to make moves that will lead to threats
130 gainOpppieces = 3.5; %Encourages to make moves that decrease opponents pieces
131 gainOwnpieces = 1; %Discourages AI from making moves that decrease own pieces
132 gainCentre = 10; %Encourages AI to increase control of centre space
133 gainOwnprom = 10; %Encourages AI to promote own pawns close to the end of the board
134 gainOppprom = -1; %Discourages AI to promote opponent's pawns
135 end
136 %————————————————— Gain Factor for Random ——————————————————————
137 if(get(handles.setRandom,'Value')==1)
138 gainCapture = 0;  %Encourages AI to position a piece such that it can capture more
        pieces in the next move
139 gainMoves = 0; %Encourages AI to position such that it opens space for other pieces
140 gainThreats = 0; %Discourages AI to make moves that will lead to threats
141 gainOpppieces = 0; %Encourages to make moves that decrease opponents pieces
142 gainOwnpieces = 0; %Discourages AI from making moves that decrease own pieces
143 gainCentre = 0; %Encourages AI to increase control of centre space
144 gainOwnprom = 0; %Encourages AI to promote own pawns close to the end of the board
145 gainOppprom = 0; %Discourages AI to promote opponent's pawns
146 end
147 %————————————————— Final Score Calculation ——————————————————————
148 boardscore =  gainCapture * capt_value_sum...
149          + gainMoves * num_moves_available...
150          + gainThreats * opp_capt_value_sum...
151          + gainOpppieces * capt_value_diff...
152          + gainOwnpieces * own_piece_sum_diff...
153          + gainCentre * centre_space_sum...
154          + gainOwnprom * sum_own_pawn_dist...
155          + gainOppprom * sum_opp_pawn_dist;
156 %Checks if castling has occured
157     if castle == 1
158         boardscore = boardscore + 250;
159     end
160 %If a checkmate has occured, new boadscores are assigned
161 if(get(handles.setHard,'Value')==1 || get(handles.setEasy,'Value')==1 )
162     if opp_ischeckmate == 1
```

```
163         boardscore = 99999;
164     end
165
166     if own_ischeckmate == 1
167         boardscore = -99999;
168     end
169 end
170
171 if(get(handles.setRandom,'Value')==1)
172     boardscore=rand* 2000;
173 end
174 end
```

A.4 Board analysis

A.4.1 analyseboard.m

```
1  %Analyseboard Looks at one colour, sees where each piece is able to
2  %move. This is to allow for the Check function and castling.
3  %Colour in this case can be either current one or opposing one
4  %Use oppositecolour to generate threats and threat captures
5  function [potentialmoves,capt_index] = analyseboard(chessboard, piece_colour,num_moves,
       colour)
6
7  %Initialisation ---------------------------------------------------------------
8  [p_x, p_y] = find(piece_colour == colour);
9  n_remaining = length(p_x);
10 potentialmoves = zeros(8,8);
11
12 %Loop to look at every piece's moves ------------------------------------------
13 for i=1:n_remaining
14     %Determines what piece is selected
15     p_type = chessboard(p_x(i),p_y(i));
16
17     %Based on the type of piece, its movement is calculated
18     switch p_type
19         case 1
20             [move] = MovementPawn(chessboard,piece_colour,num_moves,p_x(i),p_y(i));
21             %disp('Pawn');
22         case 5
23             [move] = MovementRook(chessboard,piece_colour,p_x(i),p_y(i));
24             %disp('Rook');
25         case 4
26             [move] = MovementBishop(chessboard,piece_colour,p_x(i),p_y(i));
27             %disp('Bishop');
28         case 3
29             [move] = MovementKnight(chessboard,piece_colour,p_x(i),p_y(i));
30             %disp('Knight');
31         case 9
32             [move] = MovementQueen(chessboard,piece_colour,p_x(i),p_y(i));
33             %disp('Queen');
34         case 10
35             [move] = MovementKing(chessboard,piece_colour,num_moves,potentialmoves,p_x(i
                   ),p_y(i));
36             %disp('King');
37     end
38
39     %Sums up all possible moves of 1 colour.
40     potentialmoves = potentialmoves+move;
41 end
42 %-------------------------------------------------------------------------------
43 %                     Analysis of potentialmoves
```

```
%-------------------------------------------------------------------------
%----------------------Capture Analysis-----------------------------------
potentialcaptures = potentialmoves ~= 0 & chessboard~= 0;
capt_index = find(potentialcaptures==1);
%num_pot_capture = length(capt_index);
%capt_value_sum = sum(chessboard(capt_index));

%---------------------- Moves Analysis ---------------------------------
%nocapture = potentialmoves;
%nocapture(capt_index) = 0;
%num_moves_available = sum(sum(nocapture));

end
```

A.4.2 KingCheck.m

```
%KingCheck Checks if the king is in check, checkmate or stalemate
%Colour in this case must be the current colour
%King Colour must be contrary to CAPT_INDEX & POTENTIAL MOVES
function [value]=KingCheck(chessboard,piece_colour,ownkingcolour, oppcolourcapt_index,
    oppcolourpotentialmoves)

%------------------------ King In Check ---------------------------------
king_index = find(chessboard == 10 & piece_colour == ownkingcolour);
kingincheck = ismember(king_index,oppcolourcapt_index);
if(kingincheck)
    value = 1;
%Otherwise not in check
else
    value = 0;
end
end
```

A.4.3 checkmate.m

```
%Checkmate Determines if the currentboard is a checkmate state for
%specified colour
%Gives 1 for Checkmate, 0 for not checkmate
function [result]=checkmate(B,chessboard,piece_colour, num_moves)

if(mod(B.info.turn,2)==1)
    colour = 119;
    oppcolour = 98;
else
    colour = 98;
    oppcolour = 119;
end
result = 1;
%-------------------------------------------------------------------------
%        Loop that generates all possible moves
%-------------------------------------------------------------------------
[p_x,p_y] = find(piece_colour == colour);
n_remaining = length(p_x);
[potentialmoves] = analyseboard(chessboard, piece_colour,num_moves,oppcolour);

%In essence, we are going through each piece, looking at it's possible
%moves, make those possible moves, evaluate, save bestboard.
for i=1:n_remaining
```

```
24      p_type = chessboard(p_x(i),p_y(i));
25      switch p_type
26          case 1
27              [move] = MovementPawn(chessboard,piece_colour,num_moves,p_x(i),p_y(i));
28          case 5
29              [move] = MovementRook(chessboard,piece_colour,p_x(i),p_y(i));
30          case 4
31              [move] = MovementBishop(chessboard,piece_colour,p_x(i),p_y(i));
32          case 3
33              [move] = MovementKnight(chessboard,piece_colour,p_x(i),p_y(i));
34          case 9
35              [move] = MovementQueen(chessboard,piece_colour,p_x(i),p_y(i));
36          case 10
37              [move] = MovementKing(chessboard,piece_colour,num_moves,potentialmoves,p_x(i
                    ),p_y(i));
38      end
39
40  %-------------------------------------------------------------------------
41  %            Individual Piece Moves That Generate New Game States
42  %                   Recursion is also added in each loop
43  %-------------------------------------------------------------------------
44       [move_x,move_y] = find(move ~= 0);
45       n_move = length(move_x);
46  %This loop generates all the game states from 1 piece
47       for j = 1:n_move
48           switch move(move_x(j),move_y(j))
49               case 1
50                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickMovePiece
                         (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
51                       num_moves,0,move,0,1,move_x(j),move_y(j));
52               case 2
53                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCapturePiece
                         (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
54                       num_moves,0,move,0,1,move_x(j),move_y(j));
55               case 3
56                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickEnpassant
                         (0,0,p_x(i),p_y(i),B,piece_colour,chessboard,...
57                       num_moves,0,move, 0,1,move_x(j),move_y(j));
58               case 4
59                   [pchessboard, ppiece_colour, pnum_moves,kingincheck]=ClickCastling(0,0,
                         p_x(i),p_y(i),B,piece_colour,chessboard,...
60                       num_moves,0,move,0,1,move_x(j),move_y(j));
61               case 5
62                   [pchessboard,ppiece_colour, pnum_moves,kingincheck]=ClickPawnPromo(0,0,
                         p_x(i),p_y(i),B,piece_colour,chessboard,...
63                       num_moves,0,move,0,1,move_x(j),move_y(j),'queen');
64           end
65
66           result = min(kingincheck, result);
67           if result == 0
68               break
69           end
70
71       end
72       if result == 0
73           break
74       end
75  end
```

A.4.4 readchessboard.m

```
1  %readchessboard takes in chessboard and creates B
```

```
function [B] = readchessboard(B,chessboard,piece_colour)

X = struct(NewPiece([]));
% build the initial board with everything non-playable at first
% add paddings to the non-playable areas of 4 squares and place pieces
for i=1:size(chessboard,1)+B.info.pad
    for j=1:size(chessboard,2)+B.info.pad
        X(i,j) = NewPiece([]);
    end
end


% now place pieces and playable areas
for i=1:size(chessboard,1)
    for j=1:size(chessboard,2)
        if chessboard(i,j) == 0
            pName = []; pColour = 0;
        else
            switch chessboard(i,j)
                case 1
                    pName = 'pawn';
                case 3
                    pName = 'knight';
                case 4
                    pName = 'bishop';
                case 5
                    pName = 'rook';
                case 9
                    pName = 'queen';
                case 10
                    pName = 'king';
            end

            switch piece_colour(i,j)
                case 119
                    pColour = 1;
                case 98
                    pColour = -1;
        end
        X(i+B.info.pad/2,j+B.info.pad/2) = NewPiece(pName,pColour);
    end
end
end

B.top = X;
end
```