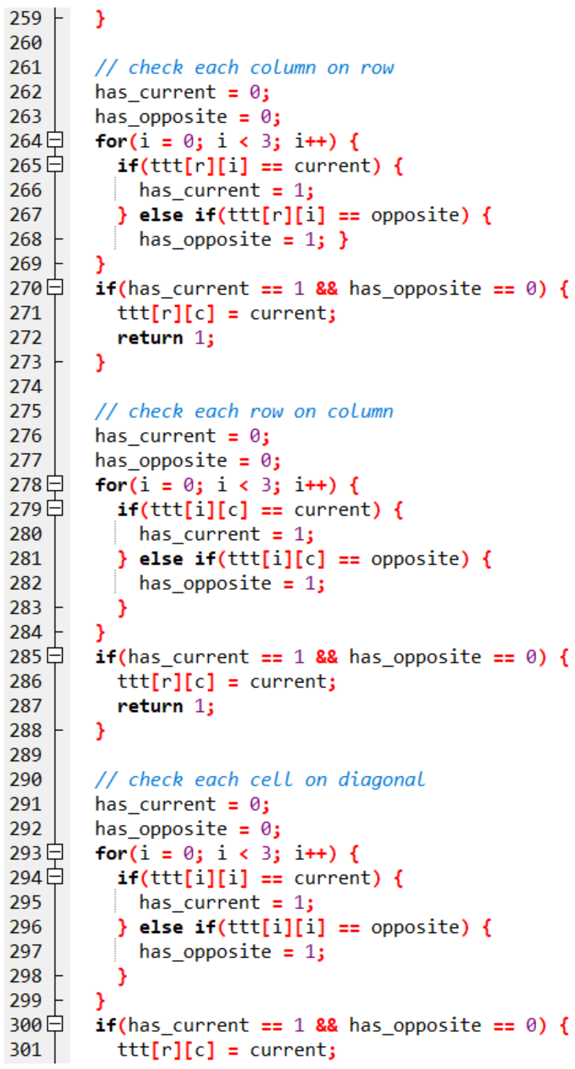
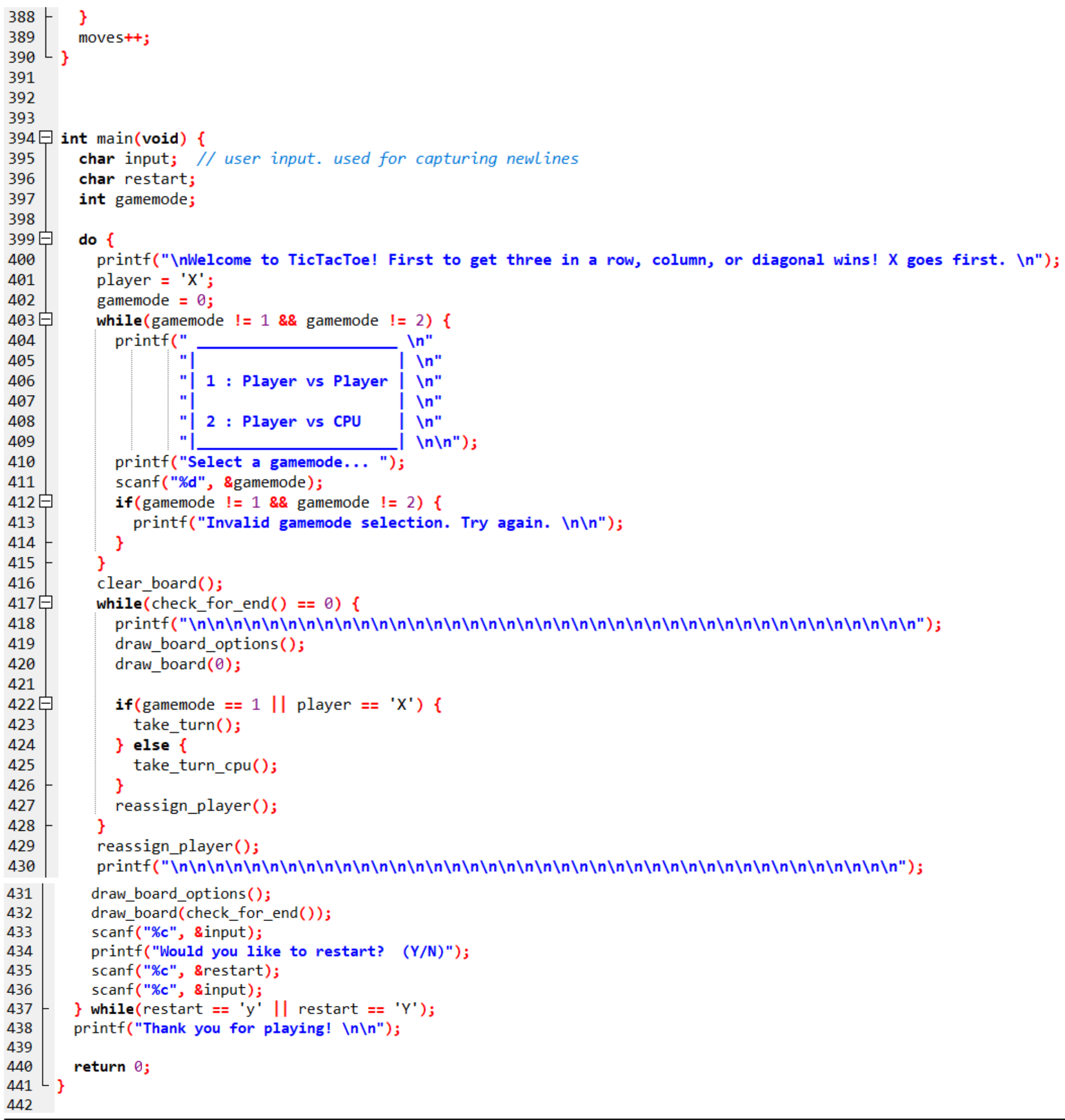
```c
/* C program by Dave Russillo. Made for CS1310. TicTacToe.
 *
 *           _                      _
 *          | |                    | |
 *         / /                     \ \
 *        /  \____             ____/  \
 *   ____/   (|____)          (____|)  \____
 *  |   |   (|_____)          (_____|) |    |
 *  |   |    (|____)          (____|)  |    |
 *  |___|_____(|___)          (___|)___|____|
 *
 */

#include <stdio.h>

char ttt[3][3];  // array represents board
char player;  // current player: either X or O
int row;  // variable used for loops and more
int col;  // variable used for loops and more
int moves;  // tracks number of moves


void draw_board_options(void) {  // draws equivalency of numbers [1-9] to places on board
  printf("\n"
         "_1_|_2_|_3_\n"
         "_4_|_5_|_6_\n"
         " 7 | 8 | 9 \n\n");
}



void draw_board(int winstate) {  // draws board and current status. adds lines based on winstate
  int i;
  int wintype = winstate / 10;  // relies on truncating
  int wincoord = winstate - winstate / 10 * 10;  // relies on truncating

  for(row = 0; row < 3; row++) {  // iterate through rows
    for(i = 0; i < 5; i++) {  // iterate through character lines within the row
      for(col = 0; col < 3; col++) {  // iterate through each column
        if(ttt[row][col] == ' ') {  // based on which line it's on, draw different parts of the ascii for X
          if(i == 4 && row != 2) printf("_________");  // separator between rows: dont add if last row
          else printf("         ");
          if(col != 2) printf("|");  // dont add column separator if last column
    } else {
      switch(i) {
        case 0:  // first line in row
          if(wintype == 2 && wincoord == col) printf("    |    ");  // if win on column, add pipe
          else if(wintype == 3 && row == col) printf("\\\\       ");  // if backward win, add backslashes
          else if(wintype == 4 && row + col == 2) printf("        /");  // if backward win, add slash
          else printf("         ");  // if no win yet
          break;
        case 1:  // second line in row
          if(ttt[row][col] == 'X') {
            if(wintype == 2 && wincoord == col) printf("   X|X   ");  // if win on column, add pipe
            else if(wintype == 3 && row == col) printf("  \\X X   ");  // if backward win, add backslash
            else if(wintype == 4 && row + col == 2) printf("   X X// ");  // if forward win, add slashes
            else printf("   X X   ");  // if no win yet
          } else {
            if(wintype == 2 && wincoord == col) printf("   O|O   ");  // if win on column, add pipe
            else if(wintype == 3 && row == col) printf("  \\OOO   ");  // if backward win, add backslash
            else if(wintype == 4 && row + col == 2) printf("   OOO// ");  // if forward win, add slashes
            else printf("   OOO   ");  // if no win yet
          }
          break;
        case 2:  // third line in row
          if(ttt[row][col] == 'X') {
            if(wintype == 1 && wincoord == row) printf("========="); // if win on row, add equal signs
            else if(wintype == 2 && wincoord == col) printf("    |    ");
            else if(wintype == 3 && row == col) printf("   \\\\    ");  // if backward win, add backslashes
            else if(wintype == 4 && row + col == 2) printf("   //    ");  // if forward win, add slashes
            else printf("    X    ");  // if no win yet
          } else {
            if(wintype == 1 && wincoord == row) printf("==O===O=="); // if win on row, add equal signs
            else if(wintype == 2 && wincoord == col) printf("  O | O  ");
            else if(wintype == 3 && row == col) printf("  O \\\\O  ");  // if backward win, add backslashes
            else if(wintype == 4 && row + col == 2) printf("  O //O  ");  // if forward win, add slashes
            else printf("  O   O  ");  // if no win yet
          }
          break;
        case 3:  // fourth line in row
          if(ttt[row][col] == 'X') {
            if(wintype == 2 && wincoord == col) printf("   X|X   ");  // if win on column, add pipe
            else if(wintype == 3 && row == col) printf("   X X\\\\ ");  // if backward win, add backslashes
            else if(wintype == 4 && row + col == 2) printf("  /X X   ");  // if forward win, add slash
            else printf("   X X   ");  // if not win yet
          } else {
                if(wintype == 2 && wincoord == col) printf("   0|0   ");  // if win on column, add pipe
                else if(wintype == 3 && row == col) printf("   000\\\\ ");  // if backward win, add backslashes
                else if(wintype == 4 && row + col == 2) printf("  /000   ");  // if forward win, add slash
                else printf("   000   ");  // if not win yet
              }
              break;
            case 4:  // last line in row
              if(row != 2) {  // if not on last row, add separator between rows
                if(wintype == 2 && wincoord == col) printf("____|____");  // if win on column, add pipe
                else if(wintype == 3 && row == col) printf("________\\");  // if backward win, add backslash
                else if(wintype == 4 && row + col == 2) printf("//_______");  // if forward win, add slash
                else printf("_________");  // if no win yet
              } else {
                if(wintype == 2 && wincoord == col) printf("    |    ");  // if win on column, add pipe
                else if(wintype == 3 && row == col) printf("        \\");  // if backward win, add backslash
                else if(wintype == 4 && row + col == 2) printf("//       ");  // if forward win, add slash
                else printf("         ");  // if no win yet
              }
          }
          // separator between columns
          if(col != 2 && wintype == 1 && wincoord == row && i == 2) printf("=");  // if win is on row, add equal sign
          else if(wintype == 3 && row == col && i == 4) printf("\\");  // add backslash instead of pipe if it's a backward win
          else if(wintype == 4 && row + col == 2 && i == 0) printf("/");  // add slash instead of pipe if it's a backward win
          else if(col != 2) printf("|");  // dont add pipe if last column

        }
      }
      printf("\n");  // newline between each line in row
    }
  }
  if(wintype == 5) {  // if it's a tie
    printf("\nTie! \n\n");
  } else if(wintype != 0) {
    printf("\n%c won! \n\n", player);
  }
}



void clear_board(void) {  // clears board
  for(row = 0; row < 3; row++) {  // iterate through rows
    for(col = 0; col < 3; col++) {  // iterate through columns
      ttt[row][col] = ' ';
    }
  }
  moves = 0;  // reset moves
}



void take_turn(void) {  // lets current player take turn and checks for valid move
  int position;
  int position_valid = 0;
  int x = 0;
  int y = 0;

  while(position_valid == 0) {  // check for valid move
    printf("\nIt's %c's turn. Choose an empty field (1-9):  ", player);  // prints instructions
    scanf("%d", &position);
    position_valid = 1;  // assumes valid
    switch(position) {  // sets coords based on position number
      case 1:
        x = 0;
        y = 0;
        break;
      case 2:
        x = 0;
        y = 1;
        break;
      case 3:
        x = 0;
        y = 2;
        break;
      case 4:
        x = 1;
        y = 0;
        break;
      case 5:
        x = 1;
        y = 1;
        break;
      case 6:
        x = 1;
        y = 2;
        break;
      case 7:
        x = 2;
        y = 0;
        break;
      case 8:
        x = 2;
        y = 1;
        break;
      case 9:
        x = 2;
        y = 2;
        break;
      default:  // means position is invalid because not in [1-9]
        position_valid = 0;
        break;
    }
    if(ttt[x][y] == ' ' && position_valid == 1) {  // check if spot is free
      ttt[x][y] = player;
    } else {
      position_valid = 0;
      printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
      draw_board_options();
      draw_board(0);
      printf("Invalid input! Try again.");
    }
  }
  moves++;  // increment moves to keep track of them
}



void reassign_player(void) {  // switches player between X and O
  if(player == 'X')
    player = 'O';
  else
    player = 'X';
}



int check_for_end(void) {  // returns wintype in tenths and coord number in units
  // stores two single digit ints in double digit int: 2, 4 -> 24
  int i;

  for(i = 0; i < 3; i++) {  // i is which row or column the win is on
    if(ttt[i][0] != ' ' && ttt[i][0] == ttt[i][1] && ttt[i][1] == ttt[i][2]) {
      return 10 + i;  // wintype 1 is horizontal win
    } else if(ttt[0][i] != ' ' && ttt[0][i] == ttt[1][i] && ttt[1][i] == ttt[2][i]) {
      return 20 + i;  // wintype 2 is vertical win
    }
  }
  if(ttt[0][0] != ' ' && ttt[0][0] == ttt[1][1] && ttt[1][1] == ttt[2][2]) {
    return 30;  // wintype 3 is backward diagonal (needs no coord)
  } else if(ttt[0][2] != ' ' && ttt[0][2] == ttt[1][1] && ttt[1][1] == ttt[2][0]) {
    return 40;  // wintype 4 is forward diagonal (needs no coord)
  } else if(moves == 9) {
    return 50;  // wintype 5 is for ties
  }
  return 0;
}



int check_for_two_in_row(int coords) {
  int r = coords / 10;  // extract row out of coords. relies on truncating.
  int c = coords % 10;  // extract column out of coords
  char current = ttt[r][c];  // current sign in cell
  char opposite;
  int i;  // to iterate (row and col already in use)
  int has_current;  // keeps track if there's the current sign on the row/column/diagonal
  int has_opposite;  // keeps track if there's an opposite to the current sign on the row/column/diagonal

  if(current == 'X') {  // find opposite
    opposite = 'O';
  } else {
    opposite = 'X';
  }

  if(ttt[r][c] == ' ') {  // don't check if given cell is empty
    return 0;
  } else {
    ttt[r][c] = ' ';
  }

  // assume coords are not (1, 1) because middle will be occupied either way
  if(coords == 11) {
    return 0;
```

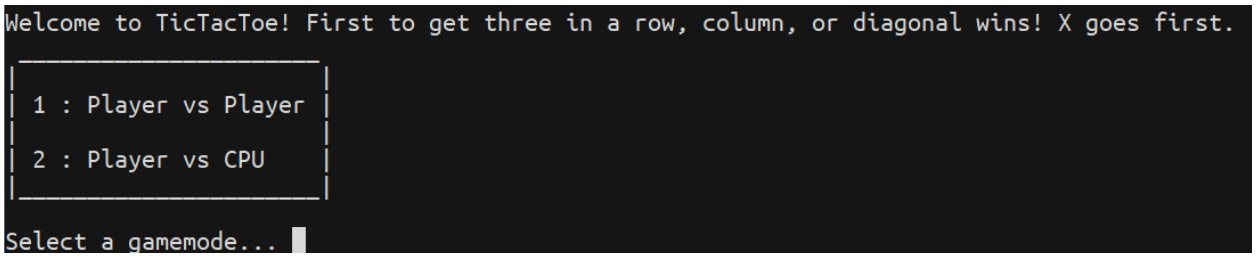


```
  }

  // check each column on row
  has_current = 0;
  has_opposite = 0;
  for(i = 0; i < 3; i++) {
    if(ttt[r][i] == current) {
      has_current = 1;
    } else if(ttt[r][i] == opposite) {
      has_opposite = 1; }
  }
  if(has_current == 1 && has_opposite == 0) {
    ttt[r][c] = current;
    return 1;
  }

  // check each row on column
  has_current = 0;
  has_opposite = 0;
  for(i = 0; i < 3; i++) {
    if(ttt[i][c] == current) {
      has_current = 1;
    } else if(ttt[i][c] == opposite) {
      has_opposite = 1;
    }
  }
  if(has_current == 1 && has_opposite == 0) {
    ttt[r][c] = current;
    return 1;
  }

  // check each cell on diagonal
  has_current = 0;
  has_opposite = 0;
  for(i = 0; i < 3; i++) {
    if(ttt[i][i] == current) {
      has_current = 1;
    } else if(ttt[i][i] == opposite) {
      has_opposite = 1;
    }
  }
  if(has_current == 1 && has_opposite == 0) {
    ttt[r][c] = current;
```

```c
    return 1;
  }

  // check each cell on backward diagonal
  has_current = 0;
  has_opposite = 0;
  for(i = 0; i < 3; i++) {
    if(ttt[i][2 - i] == current) {
      has_current = 1;
    } else if(ttt[i][2 - i] == opposite) {
      has_opposite = 1;
    }
  }
  if(has_current == 1 && has_opposite == 0) {
    ttt[r][c] = current;
    return 1;
  }

  ttt[r][c] = current;
  return 0;
}



// can make 3 in a row
// block an opponents 3 in a row
// make 2 in a row
// place in middle if available
// place in corner
// place in available spot

void take_turn_cpu(void) {
  int coords;

  // 3 in a rows
  for(row = 0; row < 3; row++) {
    for(col = 0; col < 3; col++) {
      if(ttt[row][col] == ' ') {
        ttt[row][col] = 'O';  // test with O for three in a rows
        if(check_for_end() == 0) {  // if it's not the end
          ttt[row][col] = 'X';  // test with X for three in a rows
          if(check_for_end() == 0) {  // if it's not the end
            ttt[row][col] = ' ';  // clear because no 3 in a rows
        } else {
          ttt[row][col] = 'O';
          moves++;
          return;
        }
      } else {
        ttt[row][col] = 'O';
        moves++;
        return;
      }
    }
  }
}

// 2 in a rows
for(row = 0; row < 3; row++) {
  for(col = 0; col < 3; col++) {
    if(ttt[row][col] == ' ') {
      ttt[row][col] = 'O';  // test with O for two in a rows
      coords = row * 10 + col;  // store current single digit coords in double digit int
      if(check_for_two_in_row(coords) == 0) {  // if it's not two O in a row
          ttt[row][col] = ' '; // clear because no 2 in a rows
      } else {
        ttt[row][col] = 'O';  // set to O to two in a row
        moves++;  // keep track of moves
        return;
      }
    }
  }
}

if(ttt[1][1] == ' ') {  // if middle is available
  ttt[1][1] = 'O';  // set middle to 'O'
} else if(ttt[0][0] == ' ') {  // if top left is available
  ttt[0][0] = 'O';
} else if(ttt[0][2] == ' ') {  // if top right is available
  ttt[0][2] = 'O';
} else if(ttt[2][0] == ' ') {  // if bottom left is available
  ttt[2][0] = 'O';
} else if(ttt[2][2] == ' ') {  // if bottom right is available
  ttt[2][2] = 'O';
} else {
  for(row = 0; row < 3; row++) {
    for(col = 0; col < 3; col++) {
      if(ttt[row][col] == ' ') {
        ttt[row][col] = 'O';  // pick first available spot
      }
    }
  }
```

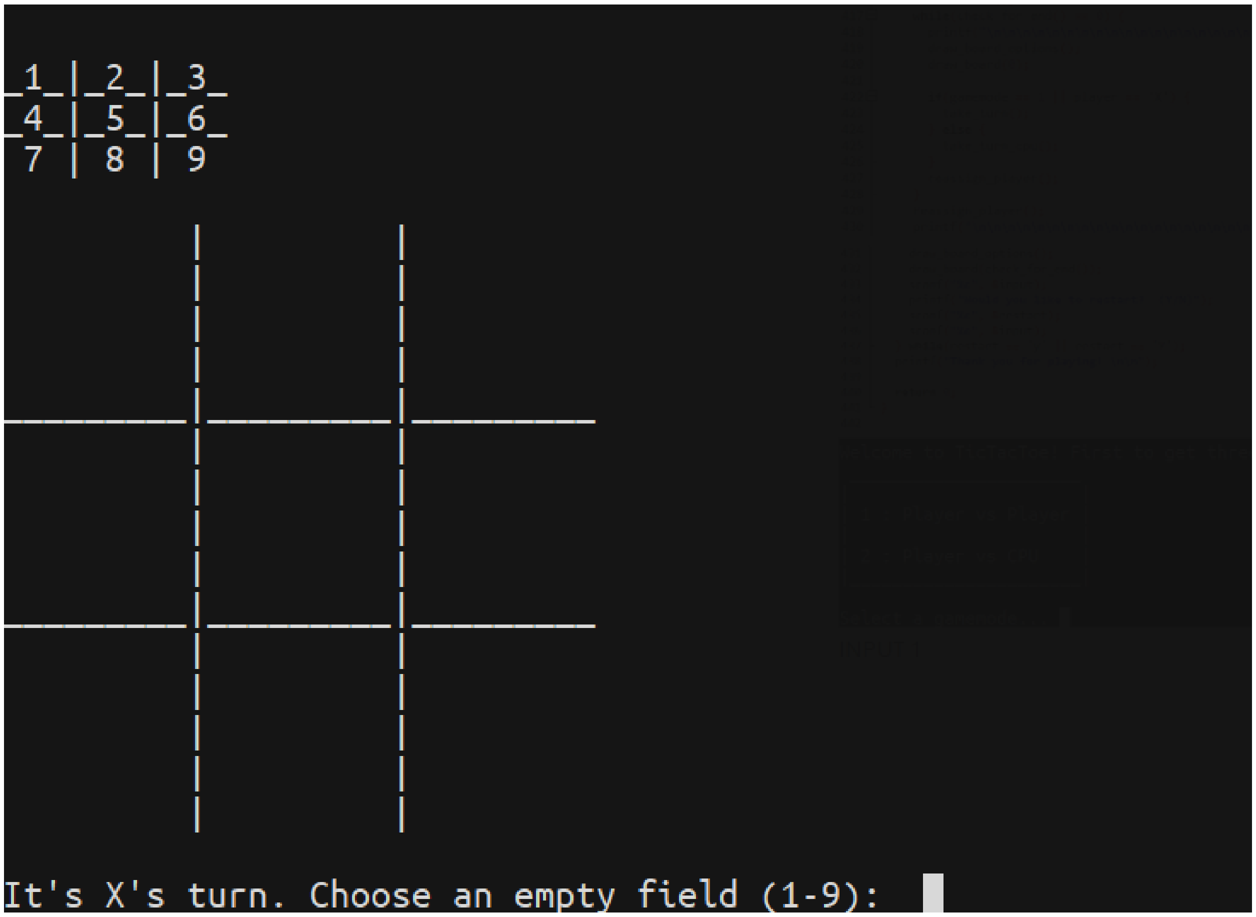


```c
  }
  moves++;
}



int main(void) {
  char input;  // user input. used for capturing newlines
  char restart;
  int gamemode;

  do {
    printf("\nWelcome to TicTacToe! First to get three in a row, column, or diagonal wins! X goes first. \n");
    player = 'X';
    gamemode = 0;
    while(gamemode != 1 && gamemode != 2) {
      printf("  ______________________  \n"
             "|                      |  \n"
             "|  1 : Player vs Player |  \n"
             "|                      |  \n"
             "|  2 : Player vs CPU    |  \n"
             "|______________________|  \n\n");
      printf("Select a gamemode... ");
      scanf("%d", &gamemode);
      if(gamemode != 1 && gamemode != 2) {
        printf("Invalid gamemode selection. Try again. \n\n");
      }
    }
    clear_board();
    while(check_for_end() == 0) {
      printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
      draw_board_options();
      draw_board(0);

      if(gamemode == 1 || player == 'X') {
        take_turn();
      } else {
        take_turn_cpu();
      }
      reassign_player();
    }
    reassign_player();
    printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
    draw_board_options();
    draw_board(check_for_end());
    scanf("%c", &input);
    printf("Would you like to restart?  (Y/N)");
    scanf("%c", &restart);
    scanf("%c", &input);
  } while(restart == 'y' || restart == 'Y');
  printf("Thank you for playing! \n\n");

  return 0;
}
```

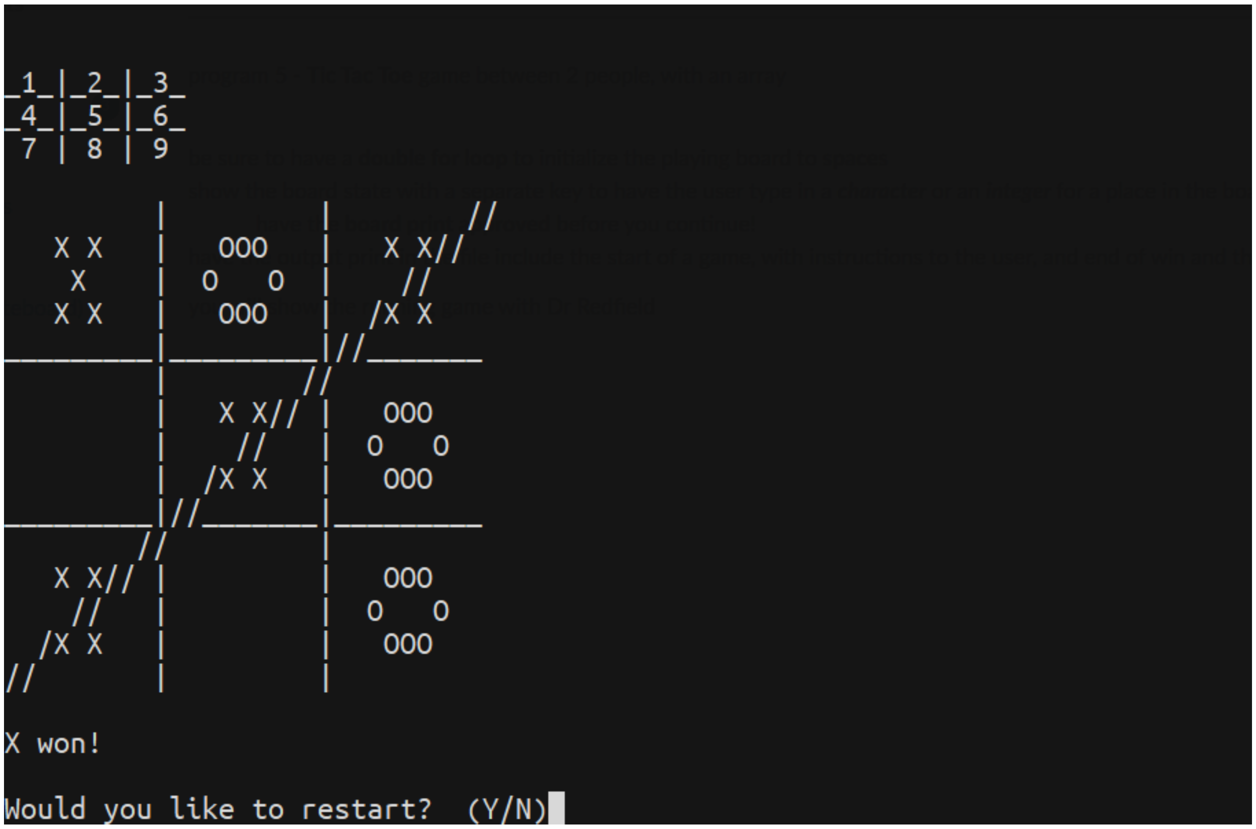


```
Welcome to TicTacToe! First to get three in a row, column, or diagonal wins! X goes first.
 ______________________
|                      |
|  1 : Player vs Player |
|                      |
|  2 : Player vs CPU    |
|______________________|

Select a gamemode...
```

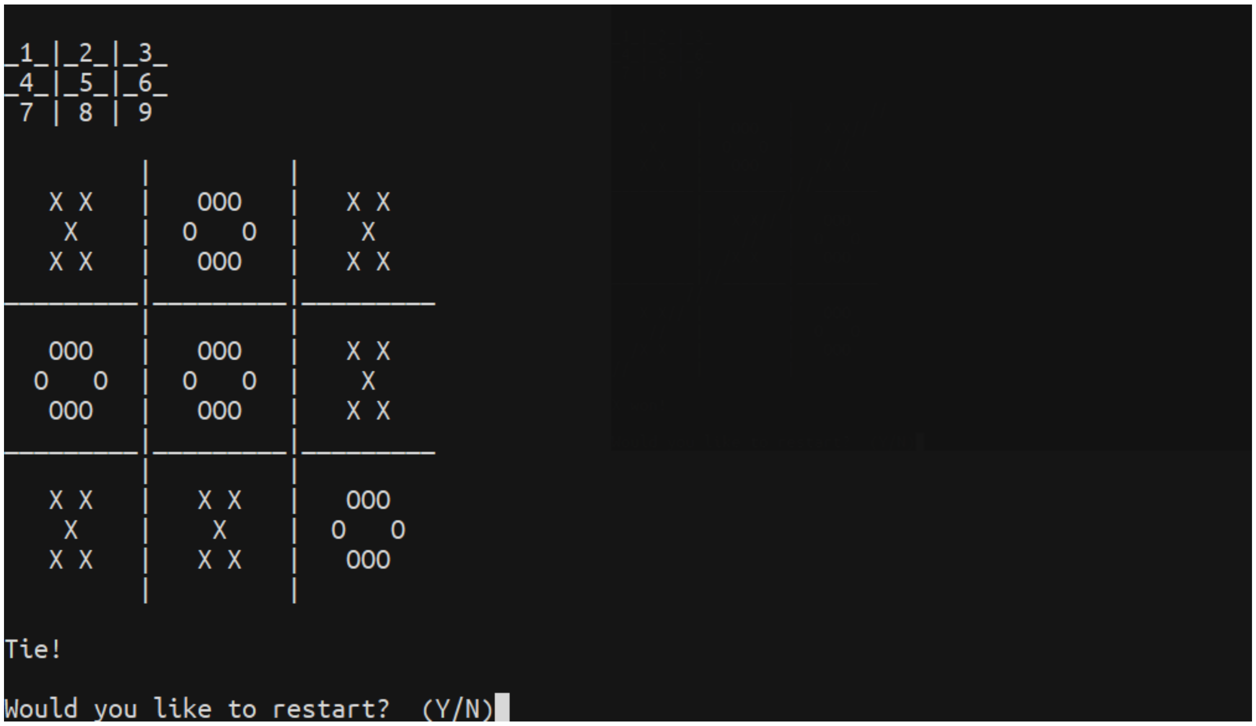
INPUT 1



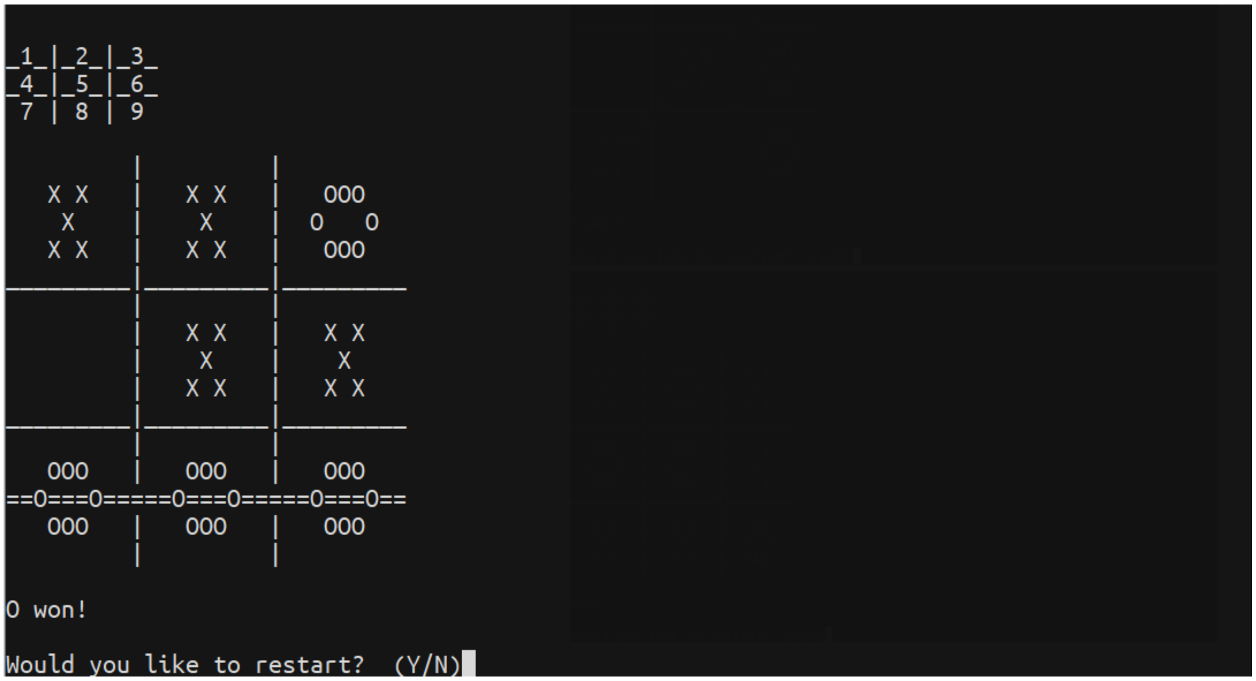
```
_1_|_2_|_3_
_4_|_5_|_6_
 7 | 8 | 9

         |         |
         |         |
         |         |
         |         |
_________|_________|_________
         |         |
         |         |
         |         |
         |         |
_________|_________|_________
         |         |
         |         |
         |         |
         |         |
         |         |

It's X's turn. Choose an empty field (1-9):
```



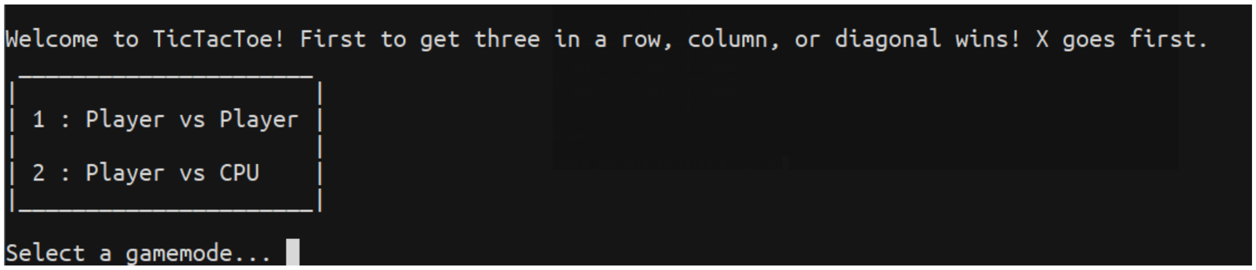
```
_1_|_2_|_3_
_4_|_5_|_6_
 7 | 8 | 9

         |         |       //
   X X   |   OOO   |  X X//
    X    |  O   O  |   //
   X X   |   OOO   |  /X X
_________|_________|//_______
         |       //
         |  X X//  |  OOO
         |   //    | O   O
         |  /X X   |  OOO
_________|//_______|_________
       //          |
   X X// |         |  OOO
    //   |         | O   O
  /X X   |         |  OOO
//       |         |

X won!

Would you like to restart?  (Y/N)
```



```
_1_|_2_|_3_
_4_|_5_|_6_
 7 | 8 | 9

         |         |
   X X   |   OOO   |   X X
    X    |  O   O  |    X
   X X   |   OOO   |   X X
_________|_________|_________
         |         |
   OOO   |   OOO   |   X X
  O   O  |  O   O  |    X
   OOO   |   OOO   |   X X
_________|_________|_________
         |         |
   X X   |   X X   |   OOO
    X    |    X    |  O   O
   X X   |   X X   |   OOO
         |         |

Tie!

Would you like to restart?  (Y/N)
```

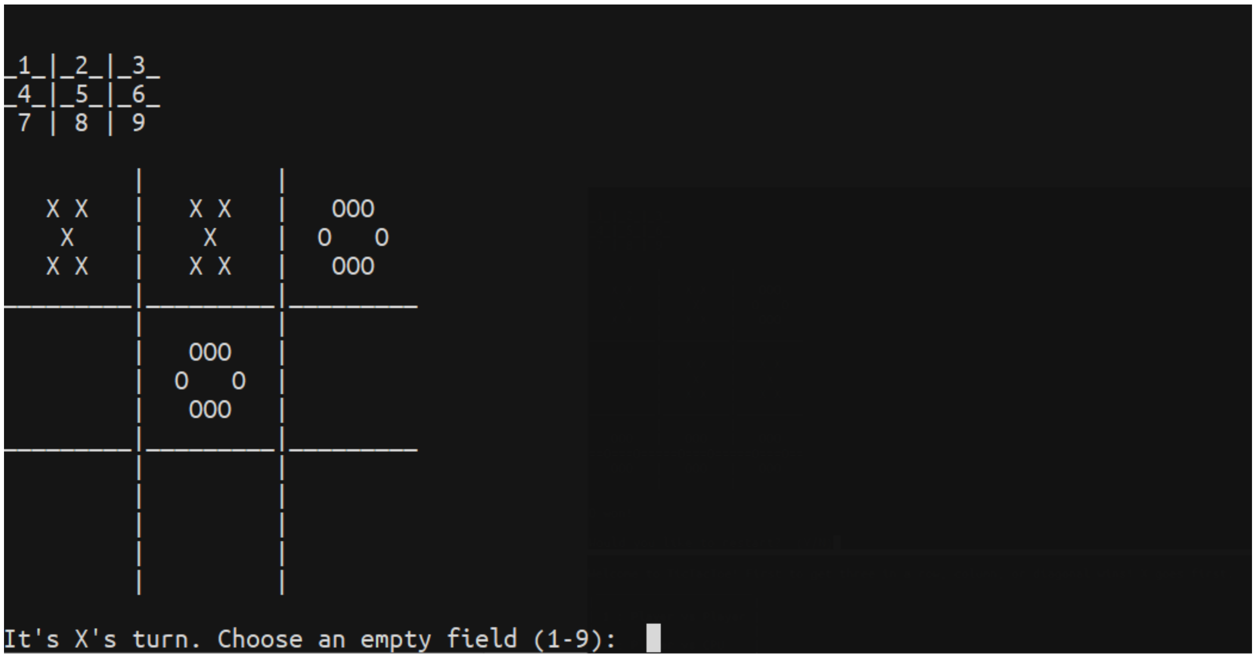


```
_1_|_2_|_3_
_4_|_5_|_6_
 7 | 8 | 9

         |         |
  X X    |   X X   |   000
   X     |    X    |  0   0
  X X    |   X X   |   000
_________|_________|_________
         |         |
         |   X X   |   X X
         |    X    |    X
         |   X X   |   X X
_________|_________|_________
         |         |
   000   |   000   |   000
==0===0=====0===0=====0===0==
   000   |   000   |   000
         |         |

O won!

Would you like to restart?  (Y/N)
```



```
Welcome to TicTacToe! First to get three in a row, column, or diagonal wins! X goes first.
 ______________________
|                      |
| 1 : Player vs Player |
|                      |
| 2 : Player vs CPU    |
|______________________|

Select a gamemode...
```

INPUT 2



```
_1_|_2_|_3_
_4_|_5_|_6_
 7 | 8 | 9

         |         |
  X X    |   X X   |   000
   X     |    X    |  0   0
  X X    |   X X   |   000
_________|_________|_________
         |         |
         |   000   |
         |  0   0  |
         |   000   |
_________|_________|_________
         |         |
         |         |
         |         |
         |         |
         |         |

It's X's turn. Choose an empty field (1-9):
```

Player is X and CPU plays O (smart)