SingleplayerVsAi.c

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

/\*

int board[25] = {

:,:,:,:,: :, :, : ,:,:

:,-,-,-,: -> :, 6, 7, 8,:

:,-,-,-,: -> :,11,12,13,:

:,-,-,-,: :,16,17,18,:

:,:,:,:,: :, :, :, :,:

0, 1, 2, 3, 4,

5, 6, 7, 8, 9,

10 ,11,12,13, 14,

15, 16,17,18, 19,

20, 21,22,23, 24

}

\*/

//Enumarate constants -> better than just initializing constants

// 0 1 2 3

enum **{** NOUGHTS**,** CROSSES**,** BORDER**,** EMPTY **};**

enum **{** HUMANWIN**,** COMPWIN**,** DRAW **};**

//12-13 is one

//12-18 is six

//12-17 is five

//12-16 is four

//times by minus one -> opposite direction

const int Directions**[**4**]** **=** **{** 1**,** 5**,** 4**,** 6 **};**

const int ConvertTo25**[**9**]** **=** **{**

6**,** 7**,** 8**,**

11**,**12**,**13**,**

16**,**17**,**18

**};**

//Arrays to Block Winning Moves

const int InMiddle **=** 4**;**

const int Corners**[**4**]** **=** **{** 0**,** 2**,** 6**,** 8 **};**

int ply **=** 0**;** // how many moves depth of the tree we are

int positions **=** 0**;** // get search positions from computer

int maxPly **=** 0**;** //see how depth we go

//gets the starting square as an argument

// 17 5 player

int GetNumForDir**(**int startSq**,** const int dir**,** const int **\***board**,** const int us**)** **{**

int found **=** 0**;**

//while we have not hit the border

**while(**board**[**startSq**]** **!=** BORDER**)**

**{**

//the the square is not equal to the same mark

**if(**board**[**startSq**]** **!=** us**)**

**{**

**break;**

**}**

//otherwise increment the score by one

found**++;**

//add the direction to the start square (if from 17 we add 5 it would go to 22 and it would break

//because it is a border

startSq **+=** dir**;**

**}**

**return** found**;**

**}**

//got out board; ourindex is the square that we are on the moment, and player(mark)

int FindThreeInARow**(**const int **\***board**,** const int ourindex**,** const int us**)** **{**

int DirIndex **=** 0**;** //loop index

int Dir **=** 0**;** //particular direction in the array

int threeCount **=** 1**;** //how many marks in a row

//do checks for all possible directions

**for(**DirIndex **=** 0**;** DirIndex **<** 4**;** **++**DirIndex**)** **{**

Dir **=** Directions**[**DirIndex**];** //gets Directions[4] = { 1, 5, 4, 6 };

//find a mark on player in direction{1,5,4 or 6}, starting at our square + 1

// going to direction 1

//getting board

// and player mark

threeCount **+=** GetNumForDir**(**ourindex **+** Dir**,** Dir**,** board**,** us**);**

//do exactly the same but multiply by -1 so we go to the opposite dirrection

threeCount **+=** GetNumForDir**(**ourindex **+** Dir **\*** **-**1**,** Dir **\*** **-**1**,** board**,** us**);**

//if we get three we break and win

**if(**threeCount **==** 3**)** **{**

**break;**

**}**

//otherwise we set threeCount to one and loop in next direction

threeCount **=** 1**;**

**}**

**return** threeCount**;**

**}**

int FindThreeInARowAllBoard**(**const int **\***board**,** const int us**)** **{**

int threeFound **=** 0**;**

int index**;**

//Loops through nine squares

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

//if at any point the loop finds humans mark ( NOUGHTS )

**if(**board**[**ConvertTo25**[**index**]]** **==** us**)**

**{**

//it loops for three in a roll as if our last move was a NOUGHT on the square

**if(**FindThreeInARow**(**board**,** ConvertTo25**[**index**],** us**)** **==** 3**)** **{**

threeFound **=** 1**;** //->yes

**break;**

**}**

**}**

**}**

**return** threeFound**;**

**}**

//takes our side on the board

int EvalForWin**(**const int **\***board**,** const int us**)** **{**

//if three are found in a roll

**if(**FindThreeInARowAllBoard**(**board**,** us**)** **!=** 0**)**

**return** 1**;** // return positive

//if on the oposite site it is crosses has found three in a roll

**if(**FindThreeInARowAllBoard**(**board**,** us **^** 1**)** **!=** 0**)**

**return** **-**1**;** // -> loss

**return** 0**;**

**}**

//Recursive function

int MinMax**(**int **\***board**,** int side**)** **{**

// check is a win

// gen all moves for side

// loop moves, make move, mimax() on move to get score

// assess bestscore

// end moves return bestscore

int MoveList**[**9**];** // the nine squares

int MoveCount **=** 0**;** // count of our moves

int bestScore **=** **-**2**;** // score would always be beaten

int score **=** **-**2**;** // keeps track of a current score for a move

int bestMove **=** **-**1**;** // keeps track of best move

int Move**;** //current move being made

int index**;** // loop index

//keep track of our counters

//depth > how depth we go

**if(**ply **>** maxPly**)**

maxPly **=** ply**;**

//increment the position because we have gone to a new position

positions**++;**

//if we are not at the top of the tree

**if(**ply **>** 0**)**

**{**

//is the current position a win or a loss

score **=** EvalForWin**(**board**,** side**);**

//if position was not won

**if(**score **!=** 0**)**

**{**

//return the current score

**return** score**;**

**}**

**}**

// fill Move List

//loop through 9 squares

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

//if any empty square

**if(** board**[**ConvertTo25**[**index**]]** **==** EMPTY**)**

**{**

MoveList**[**MoveCount**++]** **=** ConvertTo25**[**index**];**

**}**

**}**

// loop all moves

**for(**index **=** 0**;** index **<** MoveCount**;** **++**index**)**

**{**

//move is one of the moves on the move list

Move **=** MoveList**[**index**];**

//make move on board

board**[**Move**]** **=** side**;**

//increment depth

ply**++;**

//always trying to maximaze the score

//the score would be positive if a cross win (computer) - EvalForWin would return -1

score **=** **-**MinMax**(**board**,** side**^**1**);**

//if our current best score is better than the best score

**if(**score **>** bestScore**)**

**{**

bestScore **=** score**;**

bestMove **=** Move**;**

**}**

//undo our move

board**[**Move**]** **=** EMPTY**;**

ply**--;**

**}**

//if no more moves

**if(**MoveCount**==**0**)**

**{**

//search for a three in a roll

bestScore **=** FindThreeInARowAllBoard**(**board**,** side**);**

**}**

//if not on top of the tree

**if(**ply**!=**0**)**

**return** bestScore**;**

**else**

**return** bestMove**;** //if on top we will return bestoMove

**}**

//Get empty squares at middle

void InitialiseBoard**(**int **\***board**)** **{**

int index **=** 0**;** //index for looping

**for(**index **=** 0**;** index **<** 25**;** **++**index**)** **{**

board**[**index**]** **=** BORDER**;** //Get BORDER -> :

**}**

**for(**index **=** 0**;** index **<** 9**;** **++**index**)** **{**

board**[**ConvertTo25**[**index**]]** **=** EMPTY**;** //Get Empty squares -> -

**}**

**}**

void PrintBoard**(**const int **\***board**)** **{**

int index **=** 0**;** //index for looping

//Get O and X and - for board

char pceChars**[]** **=** "OX|-"**;**

printf**(**"\n\nBoard:\n\n"**);**

//Print proper board

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

//in the end of every 3 squares print line

**if(**index**!=**0 **&&** index**%**3**==**0**)**

**{**

printf**(**"\n\n"**);**

**}**

// with four characters print board

printf**(**"%4c"**,**pceChars**[**board**[**ConvertTo25**[**index**]]]);** //convert to 25 for our 9 base format

**}**

printf**(**"\n"**);**

**}**

int HasEmpty**(**const int **\***board**)**

**{**

int index **=** 0**;** // index for loop

//loop through our nine squares

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

// if we come across a square that is empty

**if(** board**[**ConvertTo25**[**index**]]** **==** EMPTY**)**

**return** 1**;** // because we found an empty square

**}**

**return** 0**;**

**}**

//get a pointer to our board, the square that is the move made and the player

void MakeMove**(**int **\***board**,** const int sq**,** const side**)**

**{**

//board at square is equal to player ( x or o)

board**[**sq**]** **=** side**;**

**}**

int GetNextBest**(**const int **\***board**)** **{**

//firstly look at middle square

int ourMove **=** ConvertTo25**[**InMiddle**];**

**if(**board**[**ourMove**]** **==** EMPTY**)**

**{**

//move to MiddleSquare

**return** ourMove**;**

**}**

int index **=** 0**;**

ourMove **=** **-**1**;**

//Looping through four corners

**for(**index **=** 0**;** index **<** 4**;** index**++)** **{**

//puts every corner in out move to see if it is emtpy

ourMove **=** ConvertTo25**[**Corners**[**index**]];**

**if(**board**[**ourMove**]** **==** EMPTY**)** **{**

**break;**

**}**

//going to next loop

ourMove **=** **-**1**;**

**}**

**return** ourMove**;**

**}**

int GetWinningMove**(**int **\***board**,** const int side**)** **{**

int ourMove **=** **-**1**;**

int winFound **=** 0**;**

int index **=** 0**;** //index for looping

//loop through the nine squares

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

//if we get an empty square

**if(** board**[**ConvertTo25**[**index**]]** **==** EMPTY**)**

**{**

ourMove **=** ConvertTo25**[**index**];**

//board on our move is set to a X or O

board**[**ourMove**]** **=** side**;**

//if win is found with the move - win

**if(**FindThreeInARow**(**board**,** ourMove**,** side**)** **==** 3**)**

**{**

winFound **=** 1**;**

**}**

//board set to empty because of pointer

board**[**ourMove**]** **=** EMPTY**;**

**if(**winFound **==** 1**)** **{**

**break;**

**}**

ourMove **=** **-**1**;**

**};**

**}**

**return** ourMove**;**

**}**

//pointer to board and computer side

//not constant because it is always changing

int GetComputerMove**(**int **\***board**,** const int side**)**

**{**

ply**=**0**;**

positions**=**0**;**

maxPly**=**0**;**

int best **=** MinMax**(**board**,** side**);** //get best move from MinMax

printf**(**"Finished Searching positions:%d maxDepth:%d bestMove:%d\n"**,**positions**,**maxPly**,**best**);**

**return** best**;**

**}**

int GetComputerMoveEASY**(**const int **\***board**)**

**{**

int index **=** 0**;**

int numFree **=** 0**;**

int availableMoves**[**9**];**

int randMove **=** 0**;**

/\* 2,4,8

availableMoves[0] = 2 numFree++ -> 1;

availableMoves[numFree] = 4 numFree++ -> 2;

availableMoves[numFree] = 8 numFree++ -> 3

rand() % numFree gives 0 to 2

rand from 0 to 2, return availableMoves[rand]

\*/

//Loop through board

**for(**index **=** 0**;** index **<** 9**;** **++**index**)**

**{**

//If square empty

**if(** board**[**ConvertTo25**[**index**]]** **==** EMPTY**)**

**{**

//Assign square to available move

availableMoves**[**numFree**++]** **=** ConvertTo25**[**index**];** //postincrementing

//numFree++ --> first use numFree and then increment it

//++numFree --> first increment and then use numFree

**};**

**}**

//get random move

randMove **=** **(**rand**()** **%** numFree**);**

//get move from one of the available move

**return** availableMoves**[**randMove**];**

**}**

int GetHumanMove**(**const int **\***board**)**

**{**

//move characters is 2 characters

char userInput**[**4**];**

int moveOk **=** 0**;**

int move **=** **-**1**;**

**while** **(**moveOk **==** 0**)**

**{**

printf**(**"Please enter a move from 1 to 9:"**);**

fgets**(**userInput**,** 3**,** stdin**);** //take 3 characters

//fflush gets rid of crappy inputs like -> fjakfjalfja6

//if not for fflush, program would take 3 characters fail and until the end until we have 6 and make a move

fflush**(**stdin**);**

// a number and an enter is 2 so if not print error

**if(**strlen**(**userInput**)** **!=** 2**)**

**{**

printf**(**"Invalid strlen()\n"**);**

**continue;**

**}**

//use sscanf to format our string (if for date %d/%d/%d)

//we have one number to pass

**if(** sscanf**(**userInput**,** "%d"**,** **&**move**)** **!=** 1**)**

**{**

//if not valid make move to nothing (nothing stored just in case

move **=** **-**1**;**

printf**(**"Invalid sscanf()\n"**);**

**continue;**

**}**

//test if move is in range

**if(** move **<** 1 **||** move **>** 9**)** **{**

move **=** **-**1**;**

printf**(**"Invalid range\n"**);**

**continue;**

**}**

move**--;** // Zero indexing

//Check if square is free to move on to

**if(** board**[**ConvertTo25**[**move**]]!=**EMPTY**)**

**{**

//if not valid make move to nothing (nothing stored just in case

move**=-**1**;**

printf**(**"Square not available\n"**);**

**continue;**

**}**

//if move okey make moveOK=1 and end while loop

moveOk **=** 1**;**

**}**

//print move made by human

printf**(**"Making Move...%d\n"**,(**move**+**1**));**

**return** ConvertTo25**[**move**];**

**}**

void RunGame**()** **{**

int GameOver **=** 0**;** //If game over it would be 1

int Side **=** CROSSES**;** // the computer would be first

int LastMoveMade **=** 0**;**

int board**[**25**];**

InitialiseBoard**(&**board**[**0**]);**

PrintBoard**(&**board**[**0**]);**

//if game over is one the game is going to run

**while(!**GameOver**)**

**{**

//Human

**if(**Side**==**NOUGHTS**)**

**{**

//get move from human, make move on board, change side;

LastMoveMade **=** GetHumanMove**(&**board**[**0**]);**

MakeMove**(&**board**[**0**],**LastMoveMade**,**Side**);**

Side**=**CROSSES**;**

**}** **else**

**{**

//get move from computer, make move on board, change side;

LastMoveMade **=** GetComputerMove**(&**board**[**0**],** Side**);**

MakeMove**(&**board**[**0**],**LastMoveMade**,**Side**);**

Side**=**NOUGHTS**;**

PrintBoard**(&**board**[**0**]);**

**}**

// if three in a row exists Game is over

//find the in a row (in board, take the last move made(ourindex), change the side) == 3

**if(** FindThreeInARow**(**board**,** LastMoveMade**,** Side **^** 1**)** **==** 3**)**

**{**

printf**(**"Game over!\n"**);**

GameOver **=** 1**;**

//we know the computer the computer won because he was crosses on LastMoveMade before changing side

**if(**Side**==**NOUGHTS**)**

**{**

printf**(**"Computer Wins\n"**);**

**}**

**else**

**{**

printf**(**"Human Wins\n"**);**

**}**

**}**

// if no more moves, game is a draw

**if(!**HasEmpty**(**board**))** **{**

printf**(**"Game over!\n"**);**

GameOver **=** 1**;**

printf**(**"It's a draw\n"**);**

**}**

**}**

//print a cross on square 1

PrintBoard**(&**board**[**0**]);**

**}**

void RunGameEasy**()**

**{**

int GameOver **=** 0**;** //If game over it would be 1

int Side **=** NOUGHTS**;** // the computer would be first

int LastMoveMade **=** 0**;**

int board**[**25**];**

InitialiseBoard**(&**board**[**0**]);**

PrintBoard**(&**board**[**0**]);**

//if game over is one the game is going to run

**while(!**GameOver**)**

**{**

//Human

**if(**Side**==**NOUGHTS**)**

**{**

//get move from human, make move on board, change side;

LastMoveMade **=** GetHumanMove**(&**board**[**0**]);**

MakeMove**(&**board**[**0**],**LastMoveMade**,**Side**);**

Side**=**CROSSES**;**

**}**

**else**

**{**

//get move from computer, make move on board, change side;

LastMoveMade **=** GetComputerMoveEASY**(&**board**[**0**]);**

MakeMove**(&**board**[**0**],**LastMoveMade**,**Side**);**

Side**=**NOUGHTS**;**

PrintBoard**(&**board**[**0**]);**

**}**

// if three in a row exists Game is over

//find the in a row (in board, take the last move made(ourindex), change the side) == 3

**if(** FindThreeInARow**(**board**,** LastMoveMade**,** Side **^** 1**)** **==** 3**)**

**{**

printf**(**"Game over!\n"**);**

GameOver **=** 1**;**

//we know the computer the computer won because he was crosses on LastMoveMade before changing side

**if(**Side**==**NOUGHTS**)**

**{**

printf**(**"Computer Wins\n"**);**

**}**

**else**

**{**

printf**(**"Human Wins\n"**);**

**}**

**}**

// if no more moves, game is a draw

**if(!**HasEmpty**(**board**))** **{**

printf**(**"Game over!\n"**);**

GameOver **=** 1**;**

printf**(**"It's a draw\n"**);**

**}**

**}**

//print a cross on square 1

PrintBoard**(&**board**[**0**]);**

**}**

int SinglePlayer**(**int choice1**)**

**{**

**if(**choice1 **==** 1**)**

**{**

//Random number generator

srand**(**time**(NULL));**

RunGameEasy**();**

**}**

**else** **if(**choice1 **==** 2**)**

**{**

//Random number generator

srand**(**time**(NULL));**

RunGame**();**

**}**

**return** 0**;**

**}**

Tic-Tac-Toe.c

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#define MAX 10

//Creating a Stack for undo option and redo option

struct stack

**{**

int array**[**MAX**];**

int top**;**

**};**

void init\_stack**(**struct stack **\*);**

void push**(**struct stack **\*,** int item**);**

int **\***pop**(**struct stack **\*);**

//Game board

char square**[**10**]** **=** **{** 'o'**,** '1'**,** '2'**,** '3'**,** '4'**,** '5'**,** '6'**,** '7'**,** '8'**,** '9' **};**

//initializing checkwin() function

int checkwin**();**

//initializing board() function

void board**();**

int MultiplePlayer**()**

**{**

FILE **\***fp**;**//=fopen("test.txt", "w");

// test for files not existing.

struct stack undoTracker**;**

struct stack redoTracker**;**

init\_stack**(&**undoTracker**);**

init\_stack**(&**redoTracker**);**

int player **=** 1**,** i**,** choice**;**

int stop **=** 1**;**

char mark**;**

char name**[**50**];**

//Select a name for file that saves moves so we can do replay

printf**(**"Select a name for saving the game as replay: "**);**

scanf**(**"%s"**,**name**);**

//Open file (create new file)

fp **=** fopen**(**name**,**"w"**);**

//If file is not open print error

**if** **(**fp **==** **NULL)**

**{**

printf**(**"Error! Could not open file\n"**);**

exit**(-**1**);** // must include stdlib.h

**}**

**do**

**{**

//Draw board

board**();**

//Two players

**if(**player **%** 2**)**

player **=** 1**;** // Odd

**else**

player **=** 2**;**

//Prompt Player for a move

printf**(**"Player %d, enter a number: "**,** player**);**

scanf**(**"%d"**,** **&**choice**);**

//Mark the move for the player

**if(**player **==** 1**)**

**{**

mark **=** 'X'**;**

**}**

**else**

**{**

mark **=** 'O'**;**

**}**

//If statements to check for choice and mark square

**if(**choice **==** 1 **&&** square**[**1**]** **==** '1'**)**

**{**

square**[**1**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 2 **&&** square**[**2**]** **==** '2'**)**

**{**

square**[**2**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 3 **&&** square**[**3**]** **==** '3'**)**

**{**

square**[**3**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 4 **&&** square**[**4**]** **==** '4'**)**

**{**

square**[**4**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 5 **&&** square**[**5**]** **==** '5'**)**

**{**

square**[**5**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 6 **&&** square**[**6**]** **==** '6'**)**

**{**

square**[**6**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 7 **&&** square**[**7**]** **==** '7'**)**

**{**

square**[**7**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 8 **&&** square**[**8**]** **==** '8'**)**

**{**

square**[**8**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 9 **&&** square**[**9**]** **==** '9'**)**

**{**

square**[**9**]** **=** mark**;**

**}**

**else** **if** **(**choice **==** 0**)**

**{**

stop **=** 0**;**

**break;**

**}**

**else** **if(**choice **==** 10**)** //Undo action if choice is 10

**{**

int **\***i **=** **NULL;**

//Remove last item from stack, replacing the square with the number

i **=** pop**(&**undoTracker**);**

**if(**i**)**

**{**

square**[\***i**]** **=** **\***i**+**'0'**;**

printf**(**"Undo move: %d\n"**,** **\***i**);**

**}**

//Propt user for key insert for continue

getch**();**

**}**

**else** //any other number would be invalid

**{**

printf**(**"Invalid move "**);**

player**--;**

getch**();**

**}**

i **=** checkwin**();** // check who wins

player**++;**

**if(**choice **!=** 10**)** //if choice is not undo, choice would be added to stack

**{**

//add to top of stack

push**(&**undoTracker**,**choice**);**

**}**

fprintf**(**fp**,**"%d \n"**,** choice**);** // write to file

**}while** **(**i **==** **-** 1**);**

//Draw board

board**();**

**if** **(**i **==** 1**)**

printf**(**"==>\aPlayer %d win "**,** **--**player**);**

**else** **if** **(**stop **==** 0**)**

printf**(**"==>\aStopped Game\n"**);**

**else**

printf**(**"==>\aGame draw"**);**

//Clear the whole board so can start again

**do**

**{**

int **\***i **=** **NULL;**

i **=** pop**(&**undoTracker**);** //Remove last item from stack, replacing the square with the number

**if(**i**)**

**{**

square**[\***i**]** **=** **\***i**+**'0'**;**

**}**

**}while(**isempty**(&**undoTracker**)** **==** 0**);**

//Close file

fclose**(**fp**);**

//Prompt for user input to continue

getch**();**

**return** 0**;**

**}**

//// function to create a stack of given capacity. It initializes size of

// stack as 0

void init\_stack**(**struct stack **\***s**)**

**{**

s**->**top **=** **-**1**;**

**}**

//Returns top element of stack.

int peek**(**struct stack **\***s**)** **{**

**return** s**->**top**;**

**}**

//Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition

void push**(**struct stack **\***s**,** int item**)**

**{**

**if(**s**->**top **==** MAX**-**1**)**

**{**

printf**(**"Stack is full"**);**

**return;**

**}**

s**->**top**++;**

s**->**array**[**s**->**top**]** **=** item**;**

**}**

//Removes an item from the stack. The items are popped in the reversed order in

//which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

int **\***pop**(**struct stack **\***s**)**

**{**

int **\***data**;**

**if(**s**->**top **==** **-**1**)**

**{**

printf**(**"Stack is empty\n"**);**

**return** **NULL;**

**}**

data **=** **&**s**->**array**[**s**->**top**];**

s**->**top**--;**

**return** data**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION TO RETURN GAME STATUS

1 FOR GAME IS OVER WITH RESULT

-1 FOR GAME IS IN PROGRESS

O GAME IS OVER AND NO RESULT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int checkwin**()**

**{**

**if** **(**square**[**1**]** **==** square**[**2**]** **&&** square**[**2**]** **==** square**[**3**])**

**return** 1**;**

**else** **if** **(**square**[**4**]** **==** square**[**5**]** **&&** square**[**5**]** **==** square**[**6**])**

**return** 1**;**

**else** **if** **(**square**[**7**]** **==** square**[**8**]** **&&** square**[**8**]** **==** square**[**9**])**

**return** 1**;**

**else** **if** **(**square**[**1**]** **==** square**[**4**]** **&&** square**[**4**]** **==** square**[**7**])**

**return** 1**;**

**else** **if** **(**square**[**2**]** **==** square**[**5**]** **&&** square**[**5**]** **==** square**[**8**])**

**return** 1**;**

**else** **if** **(**square**[**3**]** **==** square**[**6**]** **&&** square**[**6**]** **==** square**[**9**])**

**return** 1**;**

**else** **if** **(**square**[**1**]** **==** square**[**5**]** **&&** square**[**5**]** **==** square**[**9**])**

**return** 1**;**

**else** **if** **(**square**[**3**]** **==** square**[**5**]** **&&** square**[**5**]** **==** square**[**7**])**

**return** 1**;**

**else** **if** **(**square**[**1**]** **!=** '1' **&&** square**[**2**]** **!=** '2' **&&** square**[**3**]** **!=** '3' **&&**

square**[**4**]** **!=** '4' **&&** square**[**5**]** **!=** '5' **&&** square**[**6**]** **!=** '6' **&&** square**[**7**]**

**!=** '7' **&&** square**[**8**]** **!=** '8' **&&** square**[**9**]** **!=** '9'**)**

**return** 0**;**

**else**

**return** **-** 1**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FUNCTION TO DRAW BOARD OF TIC TAC TOE WITH PLAYERS MARK

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void board**()**

**{**

system**(**"cls"**);**

printf**(**"\n\n\tTic Tac Toe\n\n"**);**

printf**(**"Player 1 (X) - Player 2 (O)\n\n\n"**);**

printf**(**"\nIf you want to stop game enter 0\n"**);**

printf**(**" | | \n"**);**

printf**(**" %c | %c | %c \n"**,** square**[**1**],** square**[**2**],** square**[**3**]);**

printf**(**"\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\n"**);**

printf**(**" | | \n"**);**

printf**(**" %c | %c | %c \n"**,** square**[**4**],** square**[**5**],** square**[**6**]);**

printf**(**"\_\_\_\_\_|\_\_\_\_\_|\_\_\_\_\_\n"**);**

printf**(**" | | \n"**);**

printf**(**" %c | %c | %c \n"**,** square**[**7**],** square**[**8**],** square**[**9**]);**

printf**(**" | | \n\n"**);**

**}**

//Returns true if stack is empty, else false.

int isempty**(**struct stack **\***s**)**

**{**

**if** **(**s**->**top**==-**1**)**

**return** 1**;**

**else**

**return** 0**;**

**}**

//Function for replay moves in game

int ReplayMulti**()**

**{**

char name**[**10**];**

FILE **\***fp**;**

printf**(**"Enter file name for replay: "**);**

scanf**(**"%s"**,** name**);**

fp**=**fopen**(**name**,**"r"**);**

**if** **(**fp **==** **NULL)**

**{**

printf**(**"Error! Could not open file\n"**);**

exit**(-**1**);** // must include stdlib.h

**}**

struct stack undoTracker**;**

struct stack redoTracker**;**

init\_stack**(&**undoTracker**);**

init\_stack**(&**redoTracker**);**

int player **=** 1**,** i**,** choice**;**

int stop **=** 1**;**

char mark**;**

char nameF**[**50**];**

**if** **(**fp **==** **NULL** **)**

**{**

printf**(**"Error! Could not open file\n"**);**

exit**(-**1**);** // must include stdlib.h

**}**

int c**;**

**do**

**{**

board**();**

player **=** **(**player **%** 2**)** **?** 1 **:** 2**;**

printf**(**"Player %d, enter a number: "**,** player**);**

fscanf**(**fp**,**"%d"**,** **&**choice**);** // get input from file each choice

mark **=** **(**player **==** 1**)** **?** 'X' **:** 'O'**;**

**if(**choice **==** 1 **&&** square**[**1**]** **==** '1'**)** // Checking that the square has not already been chosen

**{**

square**[**1**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 2 **&&** square**[**2**]** **==** '2'**)**

**{**

square**[**2**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 3 **&&** square**[**3**]** **==** '3'**)**

**{**

square**[**3**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 4 **&&** square**[**4**]** **==** '4'**)**

**{**

square**[**4**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 5 **&&** square**[**5**]** **==** '5'**)**

**{**

square**[**5**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 6 **&&** square**[**6**]** **==** '6'**)**

**{**

square**[**6**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 7 **&&** square**[**7**]** **==** '7'**)**

**{**

square**[**7**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 8 **&&** square**[**8**]** **==** '8'**)**

**{**

square**[**8**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 9 **&&** square**[**9**]** **==** '9'**)**

**{**

square**[**9**]** **=** mark**;**

getch**();**

**}**

**else** **if** **(**choice **==** 0**)**

**{**

stop **=** 0**;**

**break;**

**}**

/\*else if(choice == 10)

{

int \*i = NULL;

int \*j = NULL;

i = pop(&undoTracker); //Remove last item from stack, replacing the square with the number

if(i)

{

square[\*i] = \*i+'0';

push(&redoChoice, \*i); //Update the redo stack

j = pop(&undoSymbol); //Pop from the symbol tracking stack, and copy the symbol that has been removed to the redo stack

if(j)

{

push(&redoSymbol, \*j);

}

redo = 1; //Flag showing we need to offer a redo

printf("Undo move: %d\n", \*i);

}

getch();\*/

**else** **if(**choice **==** 10**)** //Undo action if choice is 10

**{**

int **\***i **=** **NULL;**

//Remove last item from stack, replacing the square with the number

i **=** pop**(&**undoTracker**);**

**if(**i**)**

**{**

square**[\***i**]** **=** **\***i**+**'0'**;**

printf**(**"Undo move: %d\n"**,** **\***i**);**

**}**

//Propt user for key insert for continue

getch**();**

**}**

**else** //any other number would be invalid

**{**

printf**(**"Invalid move "**);**

player**--;**

getch**();**

**}**

i **=** checkwin**();**

player**++;**

**if(**choice **!=** 10**)**

**{**

push**(&**undoTracker**,**choice**);**

**}**

**}while** **((**c **=** fgetc**(**fp**))** **!=** EOF**);**

board**();**

**if** **(**i **==** 1**)**

printf**(**"==>\aPlayer %d win "**,** **--**player**);**

**else** **if** **(**stop **=** 1**)**

printf**(**"==>\aStopped Game\n"**);**

**else**

printf**(**"==>\aGame draw"**);**

**do**

**{**

int **\***i **=** **NULL;**

i **=** pop**(&**undoTracker**);** //Remove last item from stack, replacing the square with the number

**if(**i**)**

**{**

square**[\***i**]** **=** **\***i**+**'0'**;**

**}**

**}while(**isempty**(&**undoTracker**)** **==** 0**);**

getch**();**

**return** 0**;**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

END OF PROJECT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Tic-Tac-ToeMenu.c

#include <stdio.h>

#include "tic-tac-toe.c"

#include "sinlgePlayerVsAI.c"

int choice**;**

int SinglePlayerMenu**();**

int main**()**

**{**

**do**

**{**

printf**(**"\nMenu\n\n"**);**

printf**(**"1. Multiple Player\n"**);**

printf**(**"2. Single Player vs AI\n"**);**

printf**(**"3. Replay Multiple player game\n"**);**

printf**(**"4. Exit\n"**);**

scanf**(**"%d"**,&**choice**);**

**switch** **(**choice**)**

**{**

**case** 1**:** MultiplePlayer**();**

**break;**

**case** 2**:** SinglePlayerMenu**();**

**break;**

**case** 3**:** ReplayMulti**();**

**break;**

**case** 4**:** exit**(**0**);**

**break;**

**default:** printf**(**"Wrong Choice. Enter again\n"**);**

**break;**

**}**

**}** **while** **(**choice **!=** 4**);**

**}**

int SinglePlayerMenu**()**

**{**

int choice1**;**

**do**

**{**

printf**(**"\nMenu\n\n"**);**

printf**(**"1. Easy\n"**);**

printf**(**"2. Unbeatable\n"**);**

printf**(**"3. Back\n"**);**

scanf**(**"%d"**,&**choice1**);**

**switch** **(**choice1**)**

**{**

**case** 1**:** SinglePlayer**(**choice1**);**

**break;**

**case** 2**:** SinglePlayer**(**choice1**);**

**break;**

**case** 3**:** main**();**

**break;**

**default:** printf**(**"Wrong Choice. Enter again\n"**);**

**break;**

**}**

**}** **while** **(**choice1 **!=** 4**);**

**return** 3**;**

**}**