TIC TAC TOE 2003

version 1.0

Dicelords Interactive

we know the rules of the game

University Department of Information Technology

University of Mumbai, Kalina, Mumbai 400 098.

The Academic Council of UDIT having duly examined the performance of

Mr. Pushkar Modi

currently studying in the IVth Semester of the Bachelor of Science in Information Technology course, have great pleasure in awarding this

Certificate of Merit

for his work in the development of the project titled "Tic Tac Toe 2003"

as specified by the curriculum for the subject of Software Engineering.

Date: Course Coordinator Faculty Faculty

Index

I. Goals & Deliverables

Expectations:

Tic Tac Toe 2003 is a venture on its way to become a success story in the gaming industry. It will focus on students & professionals, the young and the old who enjoy having a small break between stressful hours. By developing itself as a viable source of entertainment, it will generate first fame and later revenue. This game should increase Dicelords Interactive's market share, promote name recognition, and maximize efficiency.

The TTT 2003 project is the natural evolution from the classic paper-based game of "Knots & Crosses" also known as "X & O". This simple game is played on a 3 x 3 grid. Each player takes a choice of either X or O, then using only wit, tries to create a simple line of 3 of their respective symbol. Whoever does so first, wins. If all the squares are filled, but no lines created, a draw is the result.

Mission:

The short-term objective is to start selling quickly and inexpensively, with a minimum of capital and maintenance. The long-term objective is to grow the product into a winner – an extremely stable and profitable entity that ultimately satisfies the soul at Dicelords Interactive.

Vision:

Stress must be laid on making it bug-free and maximizing customer satisfaction. We also scanned the market for similar games but most of them were either web-based or too easy. By introducing attracting features like 1 & 2 Player games, Quick game & Tournament, Difficulty Levels and Artificial Intelligence that would be tough to beat we are sure to create a market where the game would be very popular.

The game must be easy to use and understand so that all the features of the game are exploited properly. For the same reason features like Graphical and Mouse based environments need to be implemented.

Purpose:

Tic Tac Toe 2003 should be a fast loading, feature-rich game that the players can enjoy. Most of the games today are either too complex or have heavy system requirements. By scoring on both these fronts and taking advantage of the fact that it is already a very popular game we aim at giving the customers a quick, enjoyable & effective means of relaxation.

Features:

1. Single player game against the computer or two player game.
2. Artificial Intelligence designed to depict an intelligent human player.
3. 4 difficulty levels of AI for single player game.
4. Choice between playing a Quick game or Tournament of games.
5. Complete statistics report at the end of every tournament.
6. GUI and mouse support for convenient game play.
7. Standalone website for marketing and downloading the game.
8. Single player, Quick game Demo for FREE download.
9. Self-extracting file for hassle-free installation procedure.
10. Windows 9x, Me, NT, 2000 and XP compliant.

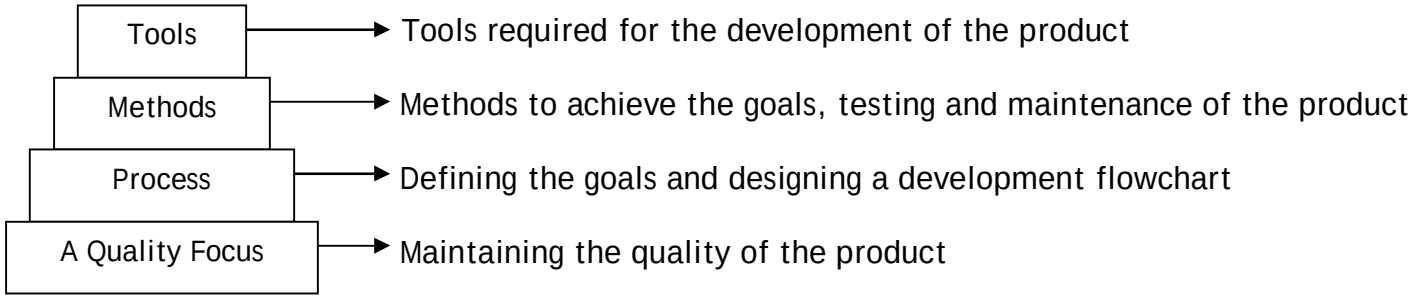
Projected, Minimum System Requirements:

- Pentium I MMX – 166 MHz or higher based system
- Windows 9x, Me, NT, 2000 or XP
- VGA, SuperVGA or compatible video adapter
- 200KB or more of free hard-drive space
- Mouse & Keyboard

II. Software Engineering

Design Implementation:

Before starting with the development of the software, we treated the software engineering process as a layered approach and chalked out four steps or guidelines to work on.



During the entire process we made sure that all of the team-members followed these rules no matter what part of the game they were working on. Below is a table of the various phases we went through with a description of how we distributed the work amongst ourselves. At the end of each working day we had a habit of discussing the things we had worked on to bring each other up-to-date on all aspects of the program. The modules handled by each one of us were:

DIPIKA CHAWLA

- 1 Player Game & AI
- Difficulty Levels
- Mouse Environment
- Website & Style-Guide

PUSHKAR MODI

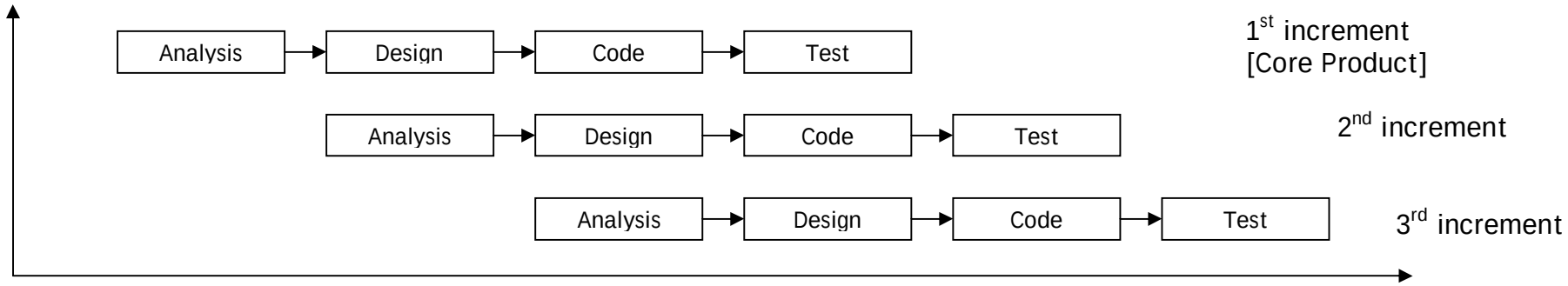
- 2 Player Game
- Quick Game
- Tournament & Alternate Game Play
- Documentation & Packaging

PREETI GOYAL

- Graphical User-Interface
- Research & Development
- Code Integration
- Testing & De-bugging

Software Process Model:

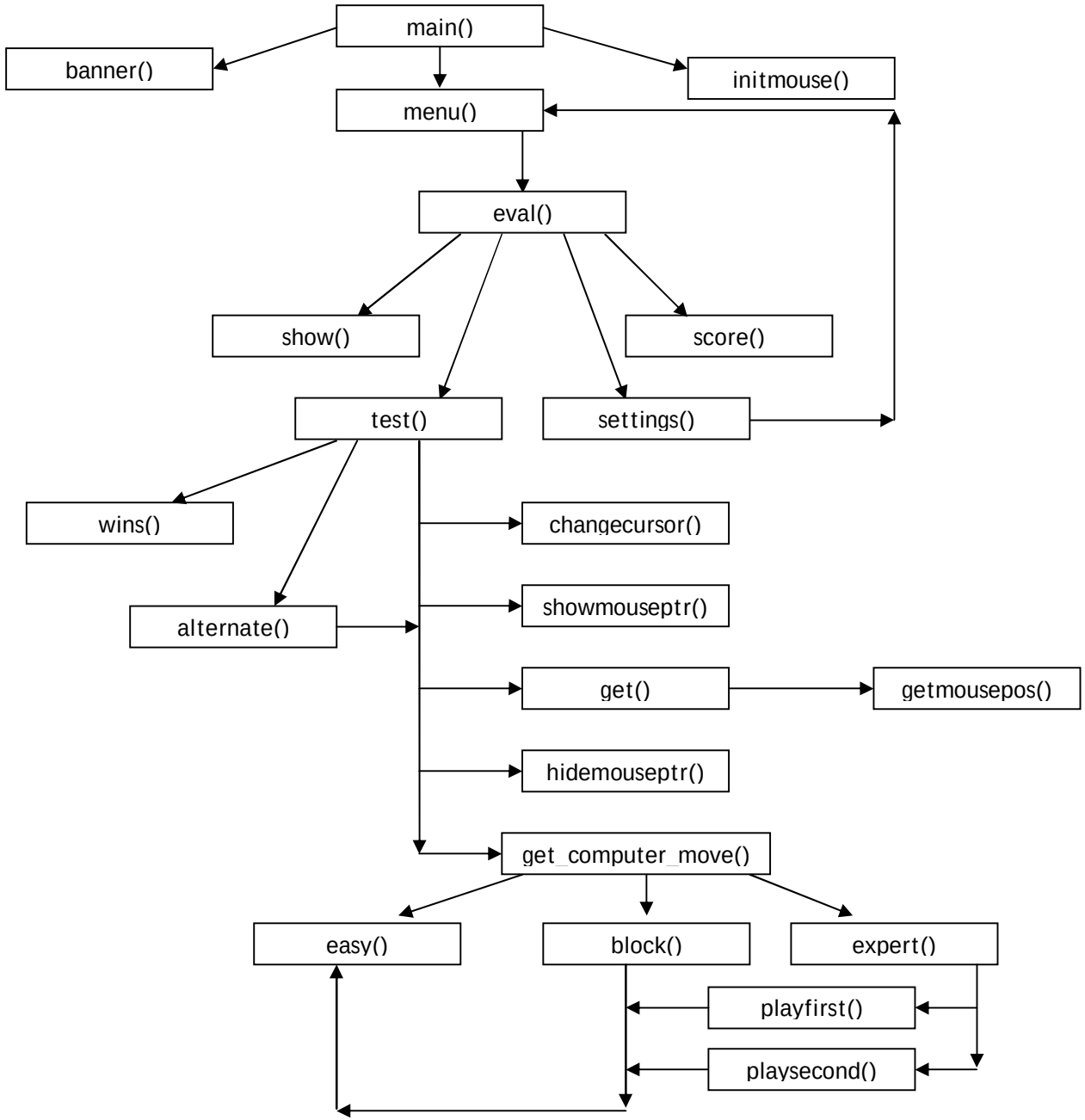
From the various Software Process Models available, our software is based on the Incremental Model. We chose this model for the development for our software because with this model we were able to work at one step at a time i.e. analyze what each step does, design implementations, code and test to check whether it performs the required task and if it contains any errors.



In the first increment, we got the core product working i.e. the basic game with a numbered grid from 1 to 9 and as the Player played, the corresponding positions in the grid being changed to X or O. In the second increment, we got the single player game working. In this we defined the functions for the moves of the Computer. In the third increment, the graphics and mouse functionality was added.

Whatever enhancements we wish to make in the future, will also be added as increments.

Control Flow Diagram:



Language Specifications:

In spite of having the option to use C or Java; we decided to make use of the ANSI C++ standards in the development of this game since we wanted to have an application that interacted directly with the hardware to invoke the graphics & mouse drivers and also followed the principles of object-oriented programming for superior reliability and data integrity. A major disadvantage of not using Java was that our product would not be platform independent, a temporary handicap which we were sure to get rid of in the future releases. We decided to use C++ over C because of the additional flexibilities it provided like dynamic initialization, polymorphism, data-hiding, encapsulation, etc.

Product Release:

To make our software as reliable as possible we decided to have 2 test-releases before the final product was introduced to the market.

- Alpha Release – This version was released internally, within our team and to other software developers. A lot of bugs were found and based on various suggestions and comments we have managed to involve some great features into the game.

- Beta Release – The Beta release was published onto the internet as a freeware for the general public to experience. Some tremendous feedback has helped us in reducing some over-looked areas and the game is becoming stronger and stronger ever since.

- Final Release v1.0 – The final version of our game was released on the website 15 days after the Beta version. A Demo version was also introduced at the same time for the user's to enjoy before they invested in the full game, especially since this is the first release of the game.

- Future Releases – We would like to continue production for the next release. The two main features we wish to incorporate are:
  - Game play across a network
  - Support for other popular platforms like UNIX, Linux & Macintosh

III. Function Descriptions, Variable Descriptions & Mouse Bitmaps

Here's a complete list of all the functions & variables used in the making of this game.

User Defined Functions	
`ttt(void)`	Constructor-initializes the data members
`void get(void)`	Gets the move from the player
`void show(char)`	Clears screen & throws output
`int test(int, char, int)`	Checks for a winner or continues the game
`void alternate(int, char, int)`	For alternate games in a tournament
`void playfirst(int)`	Computer plays first
`void playsecond(int)`	Computer plays second
`void easy(void)`	Plays randomly
`void expert(int, int)`	Play in expert level
`void block(char)`	Checks for a blocking condition
`void get_computer_move(int, int)`	Get the computer's move
`void eval(char)`	Evaluates which function should be called
`void menu(void)`	Main Menu
`void banner(void)`	Displays the TTT banner
`void initmouse(void)`	Initializes mouse drivers
`void showmouseptr(void)`	Displays the mouse pointer
`void hidemouseptr(void)`	Hides the mouse pointer
`void changecursor(int *)`	Changes the cursor shape
`void getmousepos(int *, int *, int *)`	Gets the position of the mouse
`void settings(void)`	Settings of the game
`void wins(int)`	Increases the scores
`void score(char, int)`	Displays the score card

Library Functions

`initgraph(int, int, "")`	Initializes graphics drivers, part of graphics.h
`exit(int)`	Terminates the program, part of stdlib.h
`getche(void)`	Takes a character and echoes it on screen, part of conio.h
`closegraph(void)`	Closes graphics mode, part of graphics.h
`restorecrtmode(void)`	Restores CRT mode, part of graphics.h
`return(expression)`	Returns a value
`setfillstyle(int, int)`	Sets the fill pattern and color, part of graphics.h
`bar(int, int, int, int)`	Draws a bar, part of graphics.h
`rectangle(int, int, int, int)`	Draws a rectangle, part of graphics.h
`line(int, int, int, int)`	Draws a line, part of graphics.h
`gotoxy(int, int)`	Positions cursor in window, part of conio.h
`cleardevice(void)`	Clears the graphics screen, part of graphics.h
`int86(int, union, union)`	Executes an 8086 software interrupt
`settextstyle(int, int, int)`	Sets the text style, part of graphics.h
`outtextxy(int, int, char)`	Displays a string at the specified location, part of graphics.h
`delay(unsigned)`	Suspends execution for an interval, part of dos.h
`random(int)`	Selects a random number, part of stdlib.h
`atoi(char)`	Converts string to integer, part of stdlib.h
`getch(void)`	Takes a character from screen, part of conio.h

Variable Names	Description
`int gd`	Numeric value of the graphics driver
`int gm`	Numeric value of the graphics mode
`int button`	Captures whether the left button was clicked or not
`int x`	X co-ordinate of the mouse click
`int y`	Y-co-ordinate of mouse click
`int p1wins`	Stores no. of games won by Player 1 in a tournament
`int p2wins`	Stores no. of games won by Player 2 in a tournament
`int draw`	Stores no. of tied games in a tournament
`char difficulty`	Difficulty level chosen
`char alterplay`	Alternate play - ON or OFF
`int prevgame`	No. Of the previous game played by the computer
`int cursorX[32]`	Bitmap for cursor X
`int cursor0[32]`	Bitmap for cursor 0
`int ip`	Captures the position of the next move
`char t[11]`	Array for showing the User's moves on the grid
`int b[10]`	Buffer for storing all the moves
`int move`	First move of the player in single player game
`int move_no`	Stores which turn no. it is
`int flag`	Checks if the game has ended or not
`int playone`	Move of the player in even games of tournament
`int playtwo`	Move of the player in odd games of tournament
`int newgame`	Stores 1 if the game is a new game
`int select`	Game to be played by the computer
`union REGS i,o`	Variables required for mouse functions
`struct SREGS s`	Variable required for mouse functions
`char set_choice`	Captures user's choice in the Settings Menu
`char menu_choice`	Captures user's choice in the Main Menu
`int count`	Counts the turns (each game has 9 turns)
`char tour[2]`	No. of games to be played in the tournament
`char *tour_ptr`	Pointer to string tour[]
`int no_of_games`	No. of games to be played in the tournament
`int game_no`	Game no. Of the tournament

Cursor ‘X’ Bitmap

Mouse Pointer Bitmap

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0
0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Screen Mask

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1
1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1
1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	0	0	1	1	0	0	1	1	1	1	1
1	1	1	1	0	0	1	1	1	1	0	0	1	1	1	1
1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Cursor ‘0’ Bitmap

Mouse Pointer Bitmap

0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0
0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	1	1	1	1	0	1	0	0	0	0
0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0

Screen Mask

1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1
1	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1
1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1
1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	1
1	1	1	1	0	1	0	0	0	0	1	0	1	1	1	1
1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1

IV. The Final Code

For the sake of convenient programming we have broken up the code into 10 files. Here's a list of the filenames with a list of the modules it holds.

1. TTT.CPP
   - 1.1 Preprocessor Directives
   - 1.2 Global Variables
   - 1.3 Data Members
   - 1.4 Member Functions
   - 1.5 File Includes
   - 1.6 main()
2. MENU.CPP
   - 2.1 menu()
   - 2.2 banner()
3. EVAL.CPP
   - 3.1 eval()
4. GETCONST.CPP
   - 4.1 ttt()
   - 4.2 get()
5. TESTSHOW.CPP
   - 5.1 alternate()
   - 5.2 test()
   - 5.3 show()
6. TOUR.CPP
   - 6.1 wins()
   - 6.2 score()
7. AI.CPP
   - 7.1 easy()
   - 7.2 block()
   - 7.3 get_computer_move()
8. EXPERT.CPP
   - 8.1 playfirst()
   - 8.2 playsecond()
   - 8.3 expert()
9. MOUSE.CPP
   - 9.1 initmouse()
   - 9.2 showmouseptr()
   - 9.3 hidemouseptr()
   - 9.4 changecursor()
   - 9.5 getmousepos()
10. SETTINGS.CPP
    - 10.1 settings()

1. TTT.CPP


```
//----------------------------
// 1.1 Preprocessor Directives
//----------------------------
#include<iostream.h>                          //Required for cout, cin, etc.
#include<conio.h>                             //Required for getch(), etc.
#include<stdlib.h>                            //Required for atoi(), etc.
#include<graphics.h>                          //Required for graphics mode and all related functions
#include<dos.h>                               //Required for mouse initialization


//---------------------
// 1.2 Global Variables
//---------------------
union REGS i,o;                               //Used by mouse functions
struct SREGS s;                               //Used by changecusor() function
int gd=DETECT,gm,button,x,y;                  //Used for graphics and mouse initialization
int p1wins=0, p2wins=0, draw=0;               //Records numbers of wins/draws in a tournament

char difficulty='2';                          //Checks default difficulty level
char alterplay='5';                           //Checks alternate game play ON/OFF
int prevgame=0;                               //Stores the no. of the previous game played by the computer

//The following 2 arrays hold what the 2 cursors look like.
int cursorX[32]={                             //cursor 1: X
                                              //screen mask
          0xffff,0x3ffc,0x9ff9,0xcff3,
          0xe7e7,0xf3cf,0xf99f,0xfc3f,
          0xfc3f,0xf99f,0xf3cf,0xe7e7,
          0xcff3,0x9ff9,0x3ffc,0xffff,
```

```
                                                    //mouse pointer bitmap
          0x0000,0xc003,0x6006,0x300c,
          0x1818,0x0c30,0x0660,0x03c0,
          0x03c0,0x0660,0x0c30,0x1818,
          0x300c,0x6006,0xc003,0x0000,
          };

int cursor0[32]={                                   //cursor 2: 0
                                                    //screen mask
          0xfc3f,0xf42f,0xe7e7,0xcff3,
          0x9ff9,0xffff,0x3ffc,0x3ffc,
          0x3ffc,0x3ffc,0xffff,0x9ff9,
          0xcff3,0xe7e7,0xf42f,0xfc3f,

                                                    //mouse pointer bitmap
          0x03c0,0x0bd0,0x1818,0x300c,
          0x6006,0x0000,0xc003,0xc003,
          0xc003,0xc003,0x0000,0x6006,
          0x300c,0x1818,0x0bd0,0x03c0,
          };



class ttt
{
//-----------------
// 1.3 Data-members
//-----------------
private:
 int ip;                                            //Move Input
 char t[11];                                        //Tic Tac Toe Main Board
 int b[10];                                         //Buffer of all inputs
 int move;                                          //Move of Player in single player game
 int move_no;                                       //Records move number
 int flag;                                          //Stops the game if someone wins
 int playone, playtwo;                              //Move of player in odd and even games
 int newgame;                                       //New game
 int select;                                        //Selects which game should be played in AI
```

```
//---------------------
// 1.4 Member Functions
//---------------------
public:
 ttt(void);                                  //Constructor for initializing data members
 void get(void);                             //Gets Player's move
 void show(char);                            //Displays the board
 int test(int, char, int);                   //Checks for a winner or continues game
 void alternate(int, char, int);             //For alternate games in a tournament
 void playfirst(int);                        //When the player has to play first
 void playsecond(int);                       //When the player has to play second
 void easy(void);                            //Computer picks a random number
 void expert(int, int);                      //Computer plays according to strategy
 void block(char);                           //Checks for winning possibilities
 void get_computer_move(int, int);           //Gets Computer's move
};


//------------------
// 1.5 File Includes
//------------------
#include"AI.CPP"                             //Holds Computer AI related functions
#include"TOUR.CPP"                           //Holds functions related to tournament gameplay
#include"MOUSE.CPP"                          //Holds functions for working of the mouse
#include"TESTSHOW.CPP"                       //Holds functions for playing, winning & the gameboard. etc.
#include"GETCONST.CPP"                       //Holds the constructor and the get() function
#include"SETTINGS.CPP"                       //Holds the functions related to the settings of the game.
#include"EVAL.CPP"                           //Holds eval() that evaluates input by user in the main menu.
#include"MENU.CPP"                           //Holds the functions for showing the banner & the menu
#include"EXPERT.CPP"                         //Holds the functions that select the computer’s moves by strategy.
```

```
//-----------
// 1.6 main()
//-----------

void main()
{
 initgraph(&gd,&gm,"d:\\tc\\bgi");              //Graphics drivers initialized, Check the path of the bgi directory

 if(initmouse()==0)                             //If Mouse drivers NOT initialized enter block
     {
     closegraph();                              //Close graphics mode
     cout << " ERROR: Unable to initialize mouse drivers.";
     exit(1);                                   //Exit from the game
     }

 banner();                                      //Introduction Banner
 char getc='0';

 do
 {
     menu();                                    //Makes the user select something from the menu
     cout << "\n\n Press 'M' to go back to the menu or any other key to exit.";
     getc=getche();                             //Gets user's choice
  }
 while (getc=='M' || getc=='m');                //Repeats do-while loop if true else continues

 closegraph();                                  //Closes graphics mode
 restorecrtmode();                              //Restores CRT mode
}

/*****************************************************************************/
```

2. MENU.CPP

2.1 menu()
2.2 banner()

```
//-----------
// 2.1 menu()
//-----------
void menu(void)                                 //Main Menu
{
cleardevice();                                  //Clears Screen
gotoxy(1,2);                                    //Goes to char 1 on line 2

cout  <<"  ** MAIN MENU **\n"
      <<"\n\n 1: 1 Player - Quick Game"
      <<"\n\n 2: 1 Player - Tournament"
      <<"\n\n\n 3: 2 Player - Quick Game"
      <<"\n\n 4: 2 Player - Tournament"
      <<"\n\n\n 5: Settings"
      <<"\n\n 0: Exit the game"
      <<"\n\n\n Please enter an option number from above: ";

char menu_choice;                               //Variable for capturing valid menu input
menu_choice=getche();

if (menu_choice < '0' || menu_choice > '5')     //If input not valid
menu();                                         //Re-call menu() function

eval(menu_choice);                              //else pass the value of menu_choice to eval()
}

/*****************************************************************************/
```

```
//-------------
// 2.2 banner()
//-------------
void banner(void)                               //Displays huge TTT banner!
{
cleardevice();                                  //Clears screen
settextstyle(11, HORIZ_DIR, 0);                 //Sets font-type=11, alignment as horizontal and font-size=0
outtextxy(240,230,"Dicelords Interactive");     //Goes to co-ordinate (x, y and outputs “text”)
outtextxy(285,250,"presents");                  //Goes to co-ordinate (x, y and outputs “text”)
delay(2000);                                    //Waits for 2 seconds before proceeding

cleardevice();                                  //Clears screen
line(210,149,410,149);                          //Horizontal-Top
line(210,213,410,213);                          //Horizontal-Bottom
line(273,90,273,275);                           //Vertical-Left
line(342,90,342,275);                           //Vertical-Right
delay(500);                                     //Waits for 0.5 seconds before proceeding

settextstyle(10, HORIZ_DIR, 1);                 //Sets font-type=10, alignment as horizontal and font-size=1
outtextxy(230,95,"T");
outtextxy(300,95,"I");
outtextxy(370,95,"C");
delay(500);                                     //Waits for 0.5 seconds before proceeding
outtextxy(230,155,"T");
outtextxy(300,155,"A");
outtextxy(370,155,"C");
delay(500);                                     //Waits for 0.5 seconds before proceeding
outtextxy(230,220,"T");
outtextxy(300,220,"O");
outtextxy(370,220,"E");
delay(500);                                     //Waits for 0.5 seconds before proceeding

outtextxy(272,275,"2003");                      //Goes to co-ordinate (x, y and outputs “text”)
settextstyle(11, HORIZ_DIR, 1);                 //Sets font-type=11, alignment as horizontal and font-size=1
outtextxy(272,320,"v1.0 Beta");                 //Goes to co-ordinate (x, y and outputs “text”)
delay(3000);                                    //Waits for 3 seconds before exiting
}

/*****************************************************************************/
```

3. EVAL.CPP

3.1 eval()

```
//-----------
// 3.1 eval()
//-----------
void eval(char menu_choice)
{
if (menu_choice=='0')                           //If user selects “0: Exit” from menu
exit(0);                                        //Exits from the game


else if (menu_choice=='5')                      //If user selects “5: Settings” from menu
settings();                                     //call the settings() function


else if (menu_choice=='1' || menu_choice=='3')  //If user selects “1 OR 3: Quick Game” from menu enter block
   {
     ttt game;                                  //Creates object ‘game’ of class ‘ttt’
     for (int count=1; count<=10; count++)      //Repeat the block below 10 times, count declared for the same
     {
          cleardevice();                        //Clears Screen
          gotoxy(3,2);    cout << "  ** QUICK GAME **";

          if (menu_choice=='1')                 //If user has selected a 1 Player game
               {                                //Show output
               gotoxy(60,1);   cout << "X = Player 1";
               gotoxy(60,2);   cout << "O = Computer";
               }
          else                                  //Else if user has selected a 2 Player game
               {                                //Show output
               gotoxy(60,1);   cout << "X = Player 1";
               gotoxy(60,2);   cout << "O = Player 2";
               }

          game.show(menu_choice);          //Calls function show() of object game by passing value of menu_choice
          int flag=game.test(count, menu_choice, 0);  //Assigns return value of game.test() to ‘flag’
```

```
          if (flag==1)                          //If flag=1 it means that the game is over so exit for-loop
          break;
     }

   }
else if (menu_choice=='2' || menu_choice=='4')   //If user selects “2 OR 4: Tournament” from menu enter block
   {
     char tour[2]="00";                          //Variable for taking valid tournament no
     char *tour_ptr=tour;                        //Pointer to string tour[]

     do
     {
          cleardevice();                         //Clears screen
          gotoxy(2,3);
          tour[1]='a';                           //Assign invalid value to variable
          cout << "How many games would you like to have in the tournament?"
               << "\n Minimum 1, Maximum 9 : ";
          tour[1]=getche();                      //Get single character and assign to ‘tour[1]’
     }
     while(tour[1]<'1'||tour[1]>'9');            //If invalid choice repeat do-while loop

     int no_of_games=atoi(tour_ptr);             //Convert string ‘tour[2]’ & assign to int ‘no_of_games’

     extern int p1wins, p2wins, draw;            //Variables for storing tournament wins / draws
     p1wins=0; p2wins=0; draw=0;                 //Values when tournament begins

     for (int game_no=1; game_no<=no_of_games; game_no++)  //Repeat block below for ‘no_of_games’
     {
          ttt game;                              //Creates object ‘game’ of class ‘ttt’
          for (int count=1; count<=10; count++) //Repeat the block below 10 times, count declared for the same
          {
          cleardevice();                         //Clears screen
          gotoxy(1,2);    cout << "  ** TOURNAMENT GAME "<<game_no<<" OF "<<no_of_games<<" **";

          if (menu_choice=='2')                  //If user has selected a 1 Player game
               {                                 //Show output
               gotoxy(60,1);   cout << "X = Player 1";
               gotoxy(60,2);   cout << "O = Computer";
               }
```

```
          else                                  // Else if user has selected a 2 Player game
               {                                //Show output
               gotoxy(60,1);   cout << "X = Player 1";
               gotoxy(60,2);   cout << "O = Player 2";
               }
          game.show(menu_choice);          //Calls function show() of object game by passing value of menu_choice
          int flag=game.test(count, menu_choice, game_no); //Assigns return value of game.test() to 'flag'
          if (flag==1)                          //If flag=1 it means that the game is over so exit for-loop
          break;
          }

          if (game_no != no_of_games)           //If not the last game in tournament enter block
          {
          cout << "\n\n Press any key to continue with game " << game_no+1;
          getch();                              //Wait for the user to hit a key before playing next game
          }
     }

     cout << "\n\n Press any key to see the Final Tournament Scorecard";
     getch();                                   //Wait for the user to hit a key before showing scorecard
     score(menu_choice, no_of_games);           //calls score() and passes value 'menu_choice' & 'no_of_games'
   }

}

/*****************************************************************************/
```

4. GETCONST.CPP

4.1 ttt()
4.2 get()

```
//----------
// 4.1 ttt()
//----------
ttt::ttt(void)                                  //Constructor for initializing values of data members
{
ip=0; move_no=0; move=0; flag=0; newgame=0; playone=0; playtwo=0; select=1;

b[0]=0; b[1]=0; b[2]=0; b[3]=0; b[4]=0;
b[5]=0; b[6]=0; b[7]=0; b[8]=0; b[9]=0;

t[0]=' '; t[1]=' '; t[2]=' '; t[3]=' '; t[4]=' ';
t[5]=' '; t[6]=' '; t[7]=' '; t[8]=' '; t[9]=' ';
}

/*****************************************************************************/

//----------
// 4.2 get()
//----------
void ttt::get(void)
{
gotoxy(1,23);
cout << " Please click on the appropriate position.";
ip=0;
     while(ip==0)
     {
     getmousepos(&button, &x, &y);              //Gets position & button of clicking

     if((button & 1)==1)                        //If left-click enter block below
          {
               if(x>199 && x<267)               //If click between 200 & 266 of X co-ordinate enter block
               {
                    if(y>84 && y<143)           //If click between 85 & 142 of Y co-ordinate
                    ip=1;                       //Player wants to play at position 1
```

```
                    else if(y>149 && y<207)    //If click between 150 & 206 of Y co-ordinate
                    ip=4;                      //Player wants to play at position 4

                    else if(y>213 && y<281)    //If click between 214 & 280 of Y co-ordinate
                    ip=7;                      //Player wants to play at position 7
               }
               else if(x>273 && x<336)         //If click between 274 & 335 of X co-ordinate enter block
               {
                    if(y>84 && y<143)          //If click between 85 & 142 of Y co-ordinate
                    ip=2;                      //Player wants to play at position 2

                    else if(y>149 && y<207)    //If click between 150 & 206 of Y co-ordinate
                    ip=5;                      //Player wants to play at position 5

                    else if(y>213 && y<281)    //If click between 214 & 280 of Y co-ordinate
                    ip=8;                      //Player wants to play at position 8
               }
               else if(x>342 && x<421)         //If click between 343 & 420 of X co-ordinate enter block
               {
                    if(y>84 && y<143)          //If click between 85 & 142 of Y co-ordinate
                    ip=3;                      //Player wants to play at position 3

                    else if(y>149 && y<207)    //If click between 150 & 206 of Y co-ordinate
                    ip=6;                      //Player wants to play at position 6

                    else if(y>213 && y<281)    //If click between 214 & 280 of Y co-ordinate
                    ip=9;                      //Player wants to play at position 9
               }
          }
     }

     for (int i=0; i<=9; i++)             //Check if position already occupied for values 0-9 in buffer 'b[i]'
     {
     if (ip==b[i])                             //Check if ip=b[i] which means it is an invalid move
          get();                               //Thus, re-calls the get() function
     }
}

/*****************************************************************************/
```

5. TESTSHOW.CPP

5.1 alternate()
5.2 test()
5.3 show()

```
//----------------
// 5.1 alternate()
//----------------
void ttt::alternate(int count, char menu_choice, int game_no)
{
     if (game_no%2 != 0)                        //For odd games Player 1 plays first
     {
          if (count%2 != 0)                     //For odd turns Player 1 plays first
          {
               cout << " Player 1's turn.";
               changecursor(cursorX);           //Make the mouse pointer look like an X
               showmouseptr();                  //Show the mouse pointer
               get();                           //Get the player's move
               hidemouseptr();                  //Hide the mouse pointer
               t[ip]='X';                       //Assign the position selected by the player an 'X'
               if(count==1)                     //If it's the first turn enter block
                { move=ip;}                     //Assign the player's move to 'move'
               playtwo=ip;                      //Assign the player's move to  'playtwo'
          }

          else                                  //For even turns Player 2 plays
          {
               if (menu_choice=='1' || menu_choice=='2')   //If it's a 1 Player game
               {
               get_computer_move(count,game_no);           //Get the computer's move
               }
               else                             //If it's a 2 Player game
               {
               cout << " Player 2's turn.";
               changecursor(cursor0);           //Make the mouse pointer look like an 'O'
               showmouseptr();                  //Show the mouse pointer
               get();                           //Get the player's move
```

```
                hidemouseptr();                 //Hide the mouse pointer
                }
                t[ip]='O';                      //Assign the position selected by the player an ‘O’
          }
     }

     else                                       //For even games Player 2 goes first
     {
          if (count%2 != 0)                     //For odd turns Player 2 goes first
          {
                if (menu_choice=='1' || menu_choice=='2')   //If it’s a 1 Player game
                {
                get_computer_move(count,game_no);           //Get the computer’s move
                }
                else                            //If it’s a 2 Player game
                {
                cout << " Player 2's turn.";
                changecursor(cursor0);          //Make the mouse pointer look like an ‘O’
                showmouseptr();                 //Show the mouse pointer
                get();                          //Get the player’s move
                hidemouseptr();                 //Hide the mouse pointer
                }
                t[ip]='O';                      //Assign the position selected by the player an ‘O’
          }

          else                                  //For even turns Player 1 goes first
          {
                cout << " Player 1's turn.";
                changecursor(cursorX);          //Make the mouse pointer look like an ‘X’
                showmouseptr();                 //Show the mouse pointer
                get();                          //Get the player’s move
                hidemouseptr();                 //Hide the mouse pointer
                t[ip]='X';                      //Assign the position selected by the player an ‘X’
                playone=ip;                     //Assign the player’s move to ‘playone’
          }
     }
}

/*****************************************************************************/
```

```
//-----------
// 5.2 test()
//-----------
int ttt::test(int count, char menu_choice, int game_no)     //Checks for a winner or continues the game
{
gotoxy(1,22);
//X's winning conditions
if (
     (t[1] == 'X' && t[2] == 'X' && t[3] == 'X') ||
     (t[4] == 'X' && t[5] == 'X' && t[6] == 'X') ||
     (t[7] == 'X' && t[8] == 'X' && t[9] == 'X') ||
     (t[1] == 'X' && t[4] == 'X' && t[7] == 'X') ||
     (t[2] == 'X' && t[5] == 'X' && t[8] == 'X') ||
     (t[3] == 'X' && t[6] == 'X' && t[9] == 'X') ||
     (t[1] == 'X' && t[5] == 'X' && t[9] == 'X') ||
     (t[3] == 'X' && t[5] == 'X' && t[7] == 'X')
   )
     {
     cout << " PLAYER 1 WINS!!";
     if (menu_choice=='2'||menu_choice=='4')     //If tournament
          wins(1);                               //Increment Player 1's victories
     return(1);                                  //End game
     }


//O's winning conditions
else if (
     (t[1] == 'O' && t[2] == 'O' && t[3] == 'O') ||
     (t[4] == 'O' && t[5] == 'O' && t[6] == 'O') ||
     (t[7] == 'O' && t[8] == 'O' && t[9] == 'O') ||
     (t[1] == 'O' && t[4] == 'O' && t[7] == 'O') ||
     (t[2] == 'O' && t[5] == 'O' && t[8] == 'O') ||
     (t[3] == 'O' && t[6] == 'O' && t[9] == 'O') ||
     (t[1] == 'O' && t[5] == 'O' && t[9] == 'O') ||
     (t[3] == 'O' && t[5] == 'O' && t[7] == 'O')
    )
     {
     if (menu_choice=='1' || menu_choice=='2')   //If 1 Player game
          cout << " THE COMPUTER WINS!!";
```

```
     else                                        //Else if 2 Player game
          cout << " PLAYER 2 WINS!!";
     if (menu_choice=='2'||menu_choice=='4')     //If tournament
          wins(2);                               //Increment Player 2's victories
     return(1);                                  //End game
     }

//Draw conditions
else if (count == 10)                            //If it's the 10th turn in the game
     {
     cout << " The game was a Draw!";
     if (menu_choice=='2'||menu_choice=='4')     //If tournament
          wins(0);                               //Increment no. of ties
     return(1);                                  //End game
     }

//If alternate game play is on & if it's a tournament, enter the block
else if ((alterplay=='5') && (menu_choice=='2'||menu_choice=='4'))
     {
     alternate(count, menu_choice, game_no);
     return(0);                                  //Game not over yet
     }


else if (count%2 != 0)
          {
          cout << " Player 1's turn.";
          changecursor(cursorX);                 //Make the mouse pointer look like an X
          showmouseptr();                        //Show the mouse pointer
          get();                                 //Get the player's move
          hidemouseptr();                        //Hide the mouse pointer
          t[ip]='X';                             //Assign the position selected by the player an 'X'
          if(count==1)                           //If it's the first turn enter block
           { move=ip;}                           //Assign the player's move to 'move'
          playtwo=ip;                            //Assign the player's move to  'playtwo'
          return(0);                             //Game not over yet
          }
```

```
else
     {
     if (menu_choice=='1' || menu_choice=='2')   //If it's a 1 Player game
          {
          get_computer_move(count,1);            //Get the computer's move
          }
     else                                        //If it's a 2 Player game
          {
          cout << " Player 2's turn.";
          changecursor(cursor0);                 //Make the mouse pointer look like an 'O'
          showmouseptr();                        //Show the mouse pointer
          get();                                 //Get the player's move
          hidemouseptr();                        //Hide the mouse pointer
          }

          t[ip]='O';                             //Assign the position selected by the player an 'O'
          return(0);                             //Game not over yet
     }
}

/*****************************************************************************/
```

```
//-----------
// 5.3 show()
//-----------
void ttt::show(char menu_choice)                //Clears screen & throws output
{
if (menu_choice=='1' || menu_choice=='2')       //If 1 Player game
     setfillstyle(9,9);                         //Set fill-style to pattern=9 & colour=9
else
     setfillstyle(9,12);                        //Else set fill-style pattern=9 & colour=12
bar(0,65,639,300);                              //Fill around grid

setfillstyle(1,0);    bar(190,75,430,290);      //Background colour of grid
setfillstyle(1,7);                              //Thick White bar
bar(0,50,639,64);                               //Top
bar(0,301,639,315);                             //Bottom

rectangle(190,75,430,290);                      //Border around grid
line(210,149,410,149);                          //Horizontal-Top
line(210,213,410,213);                          //Horizontal-Bottom
line(273,90,273,275);                           //Vertical-Left
line(342,90,342,275);                           //Vertical-Right

gotoxy(30,8);   cout << t[1];                   //Show position 1 on grid
gotoxy(39,8);   cout << t[2];                   //Show position 2 on grid
gotoxy(48,8);   cout << t[3];                   //Show position 3 on grid

gotoxy(30,12);  cout << t[4];                   //Show position 4 on grid
gotoxy(39,12);  cout << t[5];                   //Show position 5 on grid
gotoxy(48,12);  cout << t[6];                   //Show position 6 on grid

gotoxy(30,16);  cout << t[7];                   //Show position 7 on grid
gotoxy(39,16);  cout << t[8];                   //Show position 8 on grid
gotoxy(48,16);  cout << t[9];                   //Show position 9 on grid

b[move_no]=ip;                                  //Buffering input
move_no++;                                      //Incrementing buffer for next time
}

/*****************************************************************************/
```

6. TOUR.CPP

6.1 wins()
6.2 score()

```
//-----------
// 6.1 wins()
//-----------
void wins(int i)
{
extern int p1wins, p2wins, draw;                //Variables for storing tournament wins
if (i==1)                                       //If Player 1 has won
    p1wins++;                                   //Increments player 1's no. of wins by 1
else if (i==2)                                  //If Player 2 or the computer has won
    p2wins++;                                   //Increments player 2's no. of wins by 1
else if (i==0)                                  //If the game was a draw
    draw++;                                     //Increments no. of draws by 1
}

/*****************************************************************************/


//------------
// 6.2 score()
//------------
void score(char menu_choice, int tour_no)
{
extern int p1wins, p2wins, draw;                //Variables for storing tournament wins
cleardevice();                                  //Clears screen
gotoxy(1,1);

/*The following code displays a small cartoon of a boy holding a scorecard which displays the number of wins Player
1 & Player 2 had in the tournament*/
cout << endl << endl
     << endl << "\t\t                    \\\\\\\///          "
     << endl << "\t\t                   (  oo  )      "
     << endl << "\t\t    +--------o00o----(..)----o00o--------+  "
     << endl << "\t\t    |                                     |  "
```

```
     << endl << "\t\t    |     ** TOURNAMENT SCORECARD **     |   "
     << endl << "\t\t    |                                    |   "

     << endl << "\t\t    |  Player 1: "<<p1wins<<"          ";
     if(menu_choice=='2')cout<<"Computer: "; else cout << "Player 2: ";
     cout<<p2wins<<"  |    "

     << endl << "\t\t    |                                    |   "
     << endl << "\t\t    +---------------ooo0-----------------+   "
     << endl << "\t\t                   (  )    0ooo        "
     << endl << "\t\t                    \\ (    (  )"
     << endl << "\t\t                     \\_)   ) /        "
     << endl << "\t\t                          (_/          "
    << "\n\n The tournament consisted of "<<tour_no<<" game(s) out of which "<<draw<<" game(s) were ties.\n\n";

if(p1wins>p2wins)                                   //If Player 1 beat Player 2 enter block
     {
          cout << " Thus PLAYER 1 beat ";
          if(menu_choice=='2')                      //If 1 player game
               cout<<"the COMPUTER, ";
          else                                      //Else if 2 Player game
               cout <<"PLAYER 2, ";
          cout << p1wins << " - " << p2wins << ".\n\n";
     }

else if (p2wins>p1wins)                             //If Player 2 beat Player 1 enter block
     {
          if(menu_choice=='2')                      //If 1 Player game
               cout<<" Thus the COMPUTER";
          else                                      //Else if 2 Player game
               cout <<" Thus PLAYER 2";
          cout << " beat PLAYER 1, " << p2wins << " - " << p1wins << ".\n\n";
     }

else if (p1wins==p2wins)                            //If the tournament was a tie
cout << " Thus the tournament was a draw as both the players won " << p1wins << " game(s).\n\n";
}

/*****************************************************************************/
```

7. AI.CPP

7.1 easy()
7.2 block()
7.3 get_computer_move()

```
//-----------
// 7.1 easy()
//-----------
void ttt::easy(void)                              //For picking a move randomly
{
     do
     {
          ip = random(10);                        //Assigns a random number between 0-9 to 'ip'
     }
     while(ip==0);                                //If 'ip'=0 recall do-while loop
}

/*****************************************************************************/

//------------
// 7.2 block()
//------------
void ttt::block(char check)                       //Checks for a blocking or winning condition
{
 if( (flag==0) && (t[1]==check) )                 //When 't[1]' = X or O enter block
     {
          if(   (t[2]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[3]==check) && (t[2]==' ')     )    {ip=2; flag=1;}
     else if(   (t[4]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[7]==check) && (t[4]==' ')     )    {ip=4; flag=1;}
     else if(   (t[5]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[9]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

 if( (flag==0) && (t[2]==check) )                 //When 't[2]' = X or O
     {
          if(   (t[1]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[3]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
```

```
     else if(   (t[5]==check) && (t[8]==' ')     )    {ip=8; flag=1;}
     else if(   (t[8]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

if(  (flag==0) && (t[3]==check) )               //When 't[3]' = X or O
     {
          if(   (t[1]==check) && (t[2]==' ')     )    {ip=2; flag=1;}
     else if(   (t[2]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
     else if(   (t[6]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[9]==check) && (t[6]==' ')     )    {ip=6; flag=1;}
     else if(   (t[5]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[7]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

if(  (flag==0) && (t[4]==check) )               //When 't[4]' = X or O
     {
          if(   (t[1]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[7]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
     else if(   (t[5]==check) && (t[6]==' ')     )    {ip=6; flag=1;}
     else if(   (t[6]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

if(  (flag==0) && (t[5]==check) )               //When 't[5]' = X or O
     {
          if(   (t[1]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[2]==check) && (t[8]==' ')     )    {ip=8; flag=1;}
     else if(   (t[3]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[4]==check) && (t[6]==' ')     )    {ip=6; flag=1;}
     else if(   (t[6]==check) && (t[4]==' ')     )    {ip=4; flag=1;}
     else if(   (t[7]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[8]==check) && (t[2]==' ')     )    {ip=2; flag=1;}
     else if(   (t[9]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
     }

if(  (flag==0) && (t[6]==check) )               //When 't[6]' = X or O
     {
          if(   (t[3]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[9]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[5]==check) && (t[4]==' ')     )    {ip=4; flag=1;}
```

```
     else if(   (t[4]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

 if( (flag==0) && (t[7]==check) )               //When 't[7]' = X or O
     {
          if(   (t[4]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
     else if(   (t[1]==check) && (t[4]==' ')     )    {ip=4; flag=1;}
     else if(   (t[8]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[9]==check) && (t[8]==' ')     )    {ip=8; flag=1;}
     else if(   (t[5]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[3]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

 if( (flag==0) && (t[8]==check) )               //When 't[8]' = X or O
     {
          if(   (t[7]==check) && (t[9]==' ')     )    {ip=9; flag=1;}
     else if(   (t[9]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[5]==check) && (t[2]==' ')     )    {ip=2; flag=1;}
     else if(   (t[2]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

 if( (flag==0) && (t[9]==check) )               //When 't[9]' = X or O
     {
          if(   (t[3]==check) && (t[6]==' ')     )    {ip=6; flag=1;}
     else if(   (t[6]==check) && (t[3]==' ')     )    {ip=3; flag=1;}
     else if(   (t[7]==check) && (t[8]==' ')     )    {ip=8; flag=1;}
     else if(   (t[8]==check) && (t[7]==' ')     )    {ip=7; flag=1;}
     else if(   (t[5]==check) && (t[1]==' ')     )    {ip=1; flag=1;}
     else if(   (t[1]==check) && (t[5]==' ')     )    {ip=5; flag=1;}
     }

 if( (check=='X') && (flag==0)  )               //If win or block not possible play a random number
     {
          if(difficulty=='1' || difficulty=='2' || difficulty=='3')
          easy();
     }
}

/*****************************************************************************/
```

```
//------------------------
// 7.3 get_computer_move()
//------------------------
void ttt::get_computer_move(int count, int game_no)   // Get the computer's move
{
flag=0;                                          //Will be set to 1 when computer plays proper move

if (difficulty=='1')                             //If Difficulty set to Easy
easy();

else if (difficulty=='2')                        //If Difficulty set to Medium
block('X');                                      //Try and stop Player from winning

else if (difficulty=='3' )                       //If Difficulty set to Advanced
{
     block('O');                                 //Check if winning is possible
     if(flag==0)                                 //If no win possible
     block('X');                                 //Try and stop Player from winning
}

else if(difficulty=='4')                         // If Difficulty set to Expert
  expert(count,game_no);

for (int i=0; i<=9; i++)                         //Check if position exists for values 0-9 in buffer ‘b[i]’
     {
     if (ip==b[i])                               //Check if ip=b[i]; if true it is an invalid move
     get_computer_move(count, game_no);          //Thus ask the computer to play again
     }
}

/*****************************************************************************/
```

8. EXPERT.CPP

8.1 playfirst()
8.2 playsecond()
8.3 expert()

```
//----------------
// 8.1 playfirst()
//----------------
void ttt::playfirst(int count)             //Decides which game the computer should play when it is playing first.
{
if(flag==0)
     {
          if(newgame==0)                        //If it is a new game then, enter block
          {
          do                                    //Selects randomly the game to be played
               {
               select=random(5);                //assigns the selected game no to select
               }
          while(select==0 || select==prevgame); //Continue till no. selected is 0 or equal to previous game

          newgame=1;                            //This block should not be executed for the other moves
          prevgame=select;                      //Used for the next game
          }


     switch(select)                             //chooses from the five games using select
          {
          //1st game
          case 1:    if (count==1)                                          ip=7;
               //When count is 1, play at position 7

               else if(count==3)
                    {
                    if(t[1]=='X' || t[9]=='X')                            ip=3;
                    else if(t[3]=='X' || t[4]=='X')                       ip=9;
                    else if(t[2]=='X' || t[8]=='X' || t[6]=='X')          ip=1;
                    else if(t[5]=='X')                                    ip=6;
```

```
      else easy();
      }

else if(count==5)
      {
      switch(playone)            //playone contains the move that the player played

            {
            case 5: if(t[1]=='X')                              ip=9;
                  else if(t[9]=='X')                           ip=1;
                  else easy();
                  break;

            case 8: if(t[3]=='X')                              ip=1;
                  else if(t[4]=='X')                           ip=3;
                  else easy();
                  break;

            case 4: if(t[2]=='X' || t[6]=='X' || t[8]=='X')    ip=5;
                  else if(t[5]=='X')                           ip=9;
                  else easy();
                  break;

            case 1: ip=9;
                  break;

            default:easy();       //Pick a random number
                  break;

            }
      }

else if (count==7)
      easy();                    //Pick a random number

else if(count==9)
      {
      for (int i=1;i<=9;i++)
            {
```

```
                if(t[i]==' ')        //Play at whichever position is empty
                     {
                     ip=i;
                     break;
                     }
                }
           }
      break;



//2nd game
case 2: if (count==1)                                                ip=2;
      //When count is 1, play at position 2

      else if(count==3)
           {
           if(t[1]=='X' || t[3]=='X' || t[5]=='X' || t[8]=='X')   ip=9;
           else if(t[4]=='X' || t[6]=='X')                         ip=5;
           else if(t[7]=='X')                                      ip=1;
           else if(t[9]=='X')                                      ip=3;
           else easy();
           }

      else if(count==5)
           {
           switch(playone)           //playone contains the move that the player played

                {
                case 7: if(t[1]=='X' || t[3]=='X')                 ip=5;
                     else if(t[5]=='X' || t[8]=='X')               ip=3;
                     else easy();
                     break;

                case 8: if(t[4]=='X' || t[6]=='X' || t[5]=='X')   ip=3;
                     else if(t[3]=='X')                            ip=5;
                     else easy();
                     break;
```

```
            case 3: if(t[7]=='X')                              ip=5;
                  else if(t[1]=='X')                           ip=8;
                  else easy();
                  break;

            case 1: if(t[3]=='X')                              ip=8;

                  else easy();
                  break;

            case 4: if(t[8]=='X')                              ip=3;
                  else if(t[3]=='X')                           ip=8;
                  else easy();
                  break;

            case 5: if(t[8]=='X')                              ip=3;
                  else easy();
                  break;

            case 6: if(t[8]=='X')                              ip=1;
                  else if(t[3]=='X' || t[1]=='X')              ip=8;
                  else easy();
                  break;

            default: easy();      //Pick a random number
                   break;
            }
      }

else if (count==7)
      {
      switch(playone)            //playone contains the move that the player played

            {
            case 8: if(t[5]=='X')                             ip=4;
                  else easy();
                  break;

            default:easy();       //Pick a random number
```

```
                    break;
               }
          }


     else if (count==9)
          {
          for (int i=1;i<=9;i++)
               {
               if(t[i]==' ')        //Play at whichever position is empty
                    {
                    ip=i;
                    break;
                    }
               }
          }
     break;



//3rd game
case 3: if (count==1)                                        ip=9;
     //When count is 1, play at position 9

     else if(count==3)
          {
          if(t[1]=='X' || t[2]=='X' || t[4]=='X' || t[8]=='X')  ip=3;
          else if(t[3]=='X' || t[6]=='X')                    ip=7;
          else if(t[5]=='X' || t[7]=='X')                    ip=1;
          else easy();
          }

     else if(count==5)
          {
          switch(playone)           //playone contains the move that the player played

               {
               case 6: if(t[1]=='X' || t[2]=='X')            ip=7;
                    else if(t[4]=='X')                       ip=5;
```

```
                else if(t[8]=='X')                          ip=1;
                else easy();
                break;

          case 8: if(t[3]=='X' || t[6]=='X')                ip=1;
                else easy();
                break;

          case 7: if(t[5]=='X')                             ip=3;
                else easy();
                break;

          case 5: if(t[7]=='X')                             ip=3;
                else easy();
                break;

          default: easy();      //Pick a random number
                 break;
          }
     }

else if (count==7)
     easy();                    //Pick a random number

else if(count==9)
     {
     for (int i=1;i<=9;i++)
          {
          if(t[i]==' ')         //Play at whichever position is empty
                {
                ip=i;
                break;
                }
          }
     }
break;
```

```
//4th game
case 4:    if (count==1)                                          ip=1;
     //When count is 1, play at position 1

     else if(count==3)
          {
          if(t[3]=='X' || t[7]=='X')                             ip=9;
          else if(t[2]=='X' || t[8]=='X')                        ip=7;
          else if(t[4]=='X' || t[6]=='X' || t[9]=='X')           ip=3;
          else if(t[5]=='X')                                     ip=6;
          else easy();
          }

     else if(count==5)
          {
          switch(playone)            //playone contains the move that the player played

               {
               case 4: if(t[2]=='X')                             ip=9;
                    else if(t[8]=='X')                           ip=3;
                    else if(t[4]=='X')                           ip=3;
                    else easy();
                    break;

               case 2: if(t[4]=='X')                             ip=9;
                    else if(t[6]=='X' || t[9]=='X')              ip=7;
                    else easy();
                    break;

               case 9: if(t[5]=='X')                             ip=2;
                    else easy();
                    break;

               default: easy();      //Pick a random number
                     break;
               }
          }

     else if (count==7)
```

```
            easy();                     //Pick a random number

      else if(count==9)
            {
            for (int i=1;i<=9;i++)
                  {
                  if(t[i]==' ')         //Play at whichever position is empty
                        {
                        ip=i;
                        break;
                        }
                  }
            }
      break;



//5th game

case 5:    if (count==1)                                          ip=3;
      //When count is 1, play at position 1

      else if(count==3)
            {
            if(t[1]=='X' || t[2]=='X' || t[4]=='X')               ip=9;
            else if(t[5]=='X' || t[9]=='X')                       ip=7;
            else if(t[6]=='X' || t[8]=='X' || t[7]=='X')          ip=1;
            else easy();
            }

      else if(count==5)
            {
            switch(playone)             //playone contains the move that the player played

                  {
                  case 6: if(t[1]=='X' || t[2]=='X')              ip=7;
                        else easy();
                        break;
```

```
                    case 2: if(t[6]=='X')                         ip=7;
                            else if(t[7]=='X')                    ip=9;
                            else easy();
                            break;

                    default:easy();       //Pick a random number
                            break;
                    }
                }

            else if (count==7)
                easy();                   //Pick a random number

            else if(count==9)
                {
                for (int i=1;i<=9;i++)
                    {
                    if(t[i]==' ')         //Play at whichever position is empty
                        {
                        ip=i;
                        break;
                        }
                    }
                }
            break;
        }
    }
}

/*****************************************************************************/
```

```
//-----------------
// 8.2 playsecond()
//-----------------
void ttt::playsecond(int count)            //Decides which game the computer should play when it is playing second.
{
if(flag==0)
     {
     switch(move)                                 //slects the game to be played based on what the computer has
played
          {
          //1st game
          case 1: if (count==2)                                                    ip=5;
               //When count is 2, play at 5

               else if(count==4)
                    {
                    if(t[3]=='X' || t[6]=='X')                                   ip=2;
                    else if(t[4]=='X')                                           ip=7;
                    else if(t[7]=='X' || t[9]=='X')                              ip=4;
                    else if(t[2]=='X')                                           ip=3;
                    else if(t[8]=='X')                                           ip=6;
                    else easy();
                    }

               else if(count==6)
                    {
                    switch(playtwo)           //playtwo contains the move that the player played

                         {
                         case 8: if(t[1]=='X' || t[6]=='X')                      ip=7;
                              else easy();
                              break;

                         case 6: if(t[7]=='X')                                   ip=2;
                              else if(t[9]=='X')                                 ip=3;
                              else easy();
                              break;

                         case 4: if(t[8]=='X')                                   ip=7;
```

```
                    else easy();
                    break;

               default:easy();       //Pick a random number
                    break;
               }
          }

     else if (count==8)
          {
          switch(playtwo)            //playtwo contains the move that the player played
               {
               case 6: if(t[2]=='X')                          ip=8;
                    else easy();
                    break;

               case 8: if(t[4]=='X')                          ip=9;
                    else easy();
                    break;
               default: easy();      //Pick a random number
                     break;
               }
          }
     break;



//2nd game
case 2: if (count==2)                                         ip=5;
     //When count is 2, play at 5

     else if(count==4)
          {
          if(t[1]=='X' || t[6]=='X' || t[8]=='X' || t[9]=='X')  ip=3;
          else if(t[4]=='X' || t[3]=='X' || t[7]=='X')          ip=1;
          else easy();
          }

     else if(count==6)
```

```
            {
            switch(playtwo)            //playtwo contains the move that the player played

                  {
                  case 9: if(t[4]=='X')                              ip=3;
                        else easy();
                        break;

                  case 7: if(t[6]=='X')                              ip=1;
                        else easy();
                        break;

                  default:easy();       //Pick a random number
                        break;
                  }
            }
      else if (count==8)
            {
            switch(playtwo)            //playtwo contains the move that the player played

                  {
                  case 6: if(t[1]=='X')                              ip=8;
                        else easy();
                        break;

                  case 4: if(t[3]=='X')                              ip=7;
                        else easy();
                        break;

                  default:easy();       //Pick a random number
                        break;
                  }
            }
      break;



//3rd game
case 3: if (count==2)                                                ip=7;
```

```
//When count is 2, play at 7

else if(count==4)
     {
     if(t[2]=='X' || t[5]=='X' || t[8]=='X')               ip=1;
     else if(t[1]=='X')                                    ip=2;
     else if(t[6]=='X')                                    ip=9;
     else if(t[4]=='X')                                    ip=5;
     else if(t[9]=='X')                                    ip=6;
     else easy();
     }

else if(count==6)
     {
     switch(playtwo)            //playtwo contains the move that the player played

          {
          case 4: if(t[2]=='X')                            ip=9;
               else if(t[8]=='X')                          ip=5;
               else if(t[1]=='X')                          ip=8;
               else easy();
               break;

          case 8: if(t[6]=='X')                            ip=1;
               else easy();
               break;

          default:easy();       //Pick a random number
               break;
          }
     }

else if (count==8)
     easy();                    //Pick a random number

break;
```

```
//4th game
case 4:  if (count==2)                                          ip=6;
     //When count is 2, play at 6

     else if(count==4)
          {
          if(t[1]=='X' || t[3]=='X' || t[9]=='X')               ip=7;
          else if(t[2]=='X' || t[8]=='X')                       ip=9;
          else if(t[5]=='X')                                    ip=3;
          else if(t[7]=='X')                                    ip=1;
          else easy();
          }

     else if(count==6)
          {
          switch(playtwo)           //playtwo contains the move that the player played

               {
               case 5: if(t[3]=='X')                            ip=7;
                    else if(t[7]=='X')                          ip=3;
                    else easy();
                    break;

               case 8: if(t[3]=='X')                            ip=1;
                    else if(t[1]=='X')                          ip=3;
                    else easy();
                    break;

               case 3: if(t[8]=='X')                            ip=7;
                    else easy();
                    break;

               case 2: if(t[7]=='X')                            ip=9;
                    else if(t[9]=='X')                          ip=5;
                    else easy();
                    break;

               default:easy();      //Pick a random number
                    break;
```

```
                }
          }
     else if (count==8)
          easy();                    //Pick a random number

     break;



//5th game
case 5: if (count==2)                                          ip=1;
     //When count is 2, play at 1

     else if(count==4)
          {
          if(t[9]=='X')                                        ip=7;
          else easy();
          }

     else if(count==6)
          {
          switch(playtwo)            //playtwo contains the move that the player played
               {
               case 9: if(t[2]=='X')                           ip=7;
                    else easy();
                    break;

               default: easy();      //Pick a random number
                     break;
               }
          }
     else if(count==8)
          easy();                    //Pick a random number


     break;
```

```
//6th game
case 6: if (count==2)                                                ip=3;
     //When count is 2, play at 3

     else if(count==4)
          {
          if(t[1]=='X' || t[2]=='X' || t[4]=='X' || t[7]=='X')   ip=5;
          else if(t[5]=='X')                                         ip=4;
          else if(t[8]=='X' || t[9]=='X')                            ip=1;
          else easy();
          }

     else if(count==6)
          {
          switch(playtwo)            //playtwo contains the move that the player played
               {
               case 7: if(t[2]=='X')                                 ip=9;
                    else easy();
                    break;

               case 2: if(t[9]=='X')                                 ip=7;
                    else if(t[7]=='X')                               ip=9;
                    else easy();
                    break;

               default:easy();       //Pick a random number
                    break;
               }
          }

     else if (count==8)
          easy();                    //Pick a random number

     break;



//7th game
case 7: if (count==2)                                                ip=6;
```

```
      //When count is 2, play at 6

      else if(count==4)
            {
            if(t[1]=='X')                                          ip=4;
            else if(t[2]=='X' || t[3]=='X')                        ip=5;
            else if(t[4]=='X')                                     ip=1;
            else if(t[5]=='X')                                     ip=3;
            else if(t[8]=='X')                                     ip=9;
            else if(t[9]=='X')                                     ip=8;
            else easy();
            }

      else if(count==6)
            {
            switch(playtwo)            //playtwo contains the move that the player played
                  {
                  case 2: if(t[4]=='X')                            ip=9;
                        else if(t[9]=='X')                          ip=5;
                        else easy();
                        break;

                  default: easy();     //Pick a random number
                         break;
                  }
            }

      else if (count==8)
            easy();                    //Pick a random number

      break;



//8th game
case 8: if (count==2)                                              ip=2;
      //When count is 2, play at 2

      else if(count==4)
```

```
      {
      if(t[1]=='X' || t[4]=='X' || t[9]=='X')               ip=7;
      else if(t[3]=='X' || t[6]=='X' || t[7]=='X')          ip=9;
      else if(t[5]=='X')                                    ip=1;
      else easy();
      }

else if(count==6)
      {
      switch(playtwo)            //playtwo contains the move that the player played
            {
            case 3: if(t[6]=='X')                           ip=5;
                  else easy();
                  break;

            case 4: if(t[1]=='X')                           ip=5;
                  else if(t[9]=='X')                        ip=3;
                  else easy();
                  break;

            case 6: if(t[3]=='X')                           ip=5;
                  else if(t[7]=='X')                        ip=1;
                  else easy();
                  break;

            case 1: if(t[4]=='X')                           ip=5;
                  else easy();
                  break;

            default:easy();       //Pick a random number
                  break;
            }
      }

else if (count==8)
      easy();                    //Pick a random number

break;
```

```
          //9th game
          case 9: if (count==2)                                                ip=4;
                //When count is 2, play at 4

                else if(count==4)
                      {
                      if(t[1]=='X' || t[2]=='X')                               ip=5;
                      else if(t[3]=='X')                                       ip=6;
                      else if(t[5]=='X')                                       ip=1;
                      else if(t[6]=='X')                                       ip=3;
                      else if(t[7]=='X')                                       ip=8;
                      else if(t[8]=='X')                                       ip=7;
                      else easy();
                      }

                else if(count==6)
                      {
                      switch(playtwo)            //playtwo contains the move that the player played
                            {
                            case 2: if(t[6]=='X')                              ip=7;
                                  else easy();
                                  break;

                            default: easy();     //Pick a random number
                                   break;
                            }
                      }

                else if(count==8)
                      easy();                    //Pick a random number
                break;
          }
     }
}

/*****************************************************************************/
```

```
//-------------
// 8.3 expert()
//-------------
void ttt::expert(int count, int game_no)
{

block('O');                                     //Check if winning is possible
if(flag==0)                                     //If no win possible
block('X');                                     //Try and stop Player from winning

if (alterplay=='5')                             //If alternate game play is on
     {
     if (game_no%2 != 0)                        //Odd game
        playsecond(count);
     else                                       //Even game
        playfirst(count);
     }

else                                            //If alternate game play is off
        playsecond(count);
}

/*****************************************************************************/
```

9. MOUSE.CPP


```
/* Please refer to Yashwant Kanitkar's book, "Let Us C" for a detailed description of
how these readymade functions work, what are the various interrupts & services
related to a mouse, etc. */



//----------------
// 9.1 initmouse()
//----------------
initmouse()                                     //Initializes the mouse drivers
{
i.x.ax=0;
int86(0x33,&i,&o);
return(o.x.ax);
}

/*****************************************************************************/


//-------------------
// 9.2 showmouseptr()
//-------------------
void showmouseptr()                             //Shows the mouse pointer
{
i.x.ax=1;
int86(0x33,&i,&o);
}

/*****************************************************************************/


//-------------------
// 9.3 hidemouseptr()
//-------------------
```

```
void hidemouseptr()                             //Hides the mouse pointer
{
i.x.ax=2;
int86(0x33,&i,&o);
}

/*****************************************************************************/


//-------------------
// 9.4 changecursor()
//-------------------
void changecursor(int *shape)                   //Changes the mouse pointer to a different cursor
{
i.x.ax=9;
i.x.bx=0;
i.x.cx=0;
i.x.dx=(unsigned)shape;
segread(&s);
s.es=s.ds;
int86x(0x33,&i,&i,&s);
}

/*****************************************************************************/


//------------------
// 9.5 getmousepos()
//------------------
void getmousepos(int *button, int *x, int *y)    //Checks the x & y co-ords. & which button was clicked
{
i.x.ax=3;
int86(0x33,&i,&o);
*button=o.x.bx;
*x=o.x.cx;
*y=o.x.dx;
}

/*****************************************************************************/
```

10. SETTINGS.CPP

10.1 settings()

```
//----------------
// 10.1 settings()
//----------------
void settings(void)
{
void menu(void);                                    //Forward Declaration
cleardevice();                                      //Clears screen

gotoxy(1,2);    cout << "  ** LEVEL OF DIFFICULTY **";
gotoxy(50,2);   cout << "Current Status: ";

     if (difficulty=='1') cout << "EASY";
else if (difficulty=='2') cout << "MEDIUM";
else if (difficulty=='3') cout << "ADVANCED";
else if (difficulty=='4') cout << "EXPERT";

cout  <<"\n\n 1: Easy"
      <<"\n\n 2: Medium"
      <<"\n\n 3: Advanced"
      <<"\n\n 4: Expert";

gotoxy(1,14);   cout <<"  ** ALTERNATE GAME PLAY **";
gotoxy(50,14);  cout << "Current Status: ";
     if (alterplay=='5') cout << "ON";
else if (alterplay=='6') cout << "OFF";

cout  <<"\n\n 5: On"
      <<"\n\n 6: Off"
      <<"\n\n\n\n 0: Save & Exit"
      <<"\n\n\n Please enter an option number from above: ";

char set_choice;                                    //Variable for capturing valid settings input
set_choice=getche();                                //Gets user's choice
```

```
if (set_choice < '0' || set_choice > '6')        //If input is in-valid
settings();                                      //Re-call the settings() function

else if (set_choice >= '1' && set_choice <='4')  //If input for changing difficulty levels
     {
     difficulty=set_choice;                      //Assign the selection to ‘difficulty’
     settings();                                 //Re-call the settings() function
     }

else if (set_choice >= '5' && set_choice <='6')  //If input for switching alternate play ON or OFF
     {
     alterplay=set_choice;                       //Assign the selection to ‘alterplay’
     settings();                                 //Re-call the settings() function
     }

else if (set_choice=='0')                        //If input is to save & exit
menu();                                          //Re-call the menu() function
}

/*****************************************************************************/
```

V. In-Depth Working

Getting Started:

The first thing that we needed to take care of when we began coding is include the header files into the program. Since the game was to be developed in C++, we needed the 'iostream.h' file for using 'cout', 'cin', etc. We also included the file 'conio.h' since we extensively used the functions - gotoxy(), getch(), getche(), etc. We saved this file in a folder by the name of TTT.CPP. The first thing we wanted when the game started was to display a banner showing the words "Dicelords Interactive presents" in the center of the screen. So we started writing the main() function.

Initializing the Graphics Drivers:

Since main() doesn't return a value we made it's return-type void. This was a good time to invoke the graphics drivers and so we included the 'graphics.h' header file and declared 3 global variables, int gd=DETECT,gm. The EGAVGA.BGI file in the BGI directory of the TURBO C++ 3.0 compiler can be used to automatically invoke the graphics drivers of the system. We referred to Yashwant Kanitkar's book, "Let Us C" and got the graphics drivers initialized. There is a readymade function, initgraph() that needs to be passed the values of the 2 variable we defined by reference along with the path of the EGAVGA.BGI file. So the final code that got the graphics drivers initialized looked like this:

```
initgraph(&gd,&gm,"d:\\tc\\bgi");                    //Graphics drivers initialized
```

After some research we found out that the screen was at a resolution of 640 x 480 by default although that could be changed. We also added the functions closegraph() and restorecrtmode() as the last 2 lines of main().

Introducing the banner():

Since we wanted to show 2 banners, one for "Dicelords Interactive presents" and the other mentioning the name of the game we decided to create a new function to avoid the clutter in main() and to stick with the true concept of modularity. The function called banner() is placed in a file called BANNER.CPP in the same folder and included in main(). The first thing that we wanted to do was to clear the screen whenever the banner was called. Since clrscr() has its own problems with the graphic mode we were forced to cleardevice() in all the places where we wished to clear the screen. After a lot of trial and error in calculating the right co-ordinates we came up with the code in module 2.2. We'll try and explain some of the functions here:

- The settextstyle() function is part of the 'graphics.h' package and is basically used to select the style, alignment and the size of the font to be displayed. The results don't come with the normal output but only special function; outtextxy() is one such function.
  Syntax: settextstyle(font-style, alignment, font-size);

- outtextxy() also part of 'graphics.h' is another such function that just outputs the text at the given x & y co-ordinates.
  Syntax: outtextxy(x co-ordinate, y co-ordinate, "text");

- delay() is part of the 'dos.h' header file and is therefore included in the main file. It is used to make a pause in the program, necessary in our case so that the user has the time to read the output properly before it jumps to the next screen.
  Syntax: delay(time in milliseconds);

- line() is part of the 'graphics.h' file and is used to draw a line from (x1,y1) to (x2,y2). Remember that the resolution of the screen is set to 640 x 480 so the values of the co-ordinates need to be within those limits to prevent errors.
  Syntax: line(x1, y1, x2, y2);

The rest of the code is written so that the 1st screen is followed by something like this with the version of the game clearly specified.

T	I	C
T	A	C
T	O	E

The Main Menu:

After the completion of the banner() function the control returns to main() and we need to show the Main Menu to the user. Another function menu() is defined in the BANNER.CPP file. This file is later renamed to MENU.CPP since the menu() function is the more important of the two functions that reside in the file. Now the menu() function simply needs to show a list of possible options to the user. We want to have 1 & 2 Player games along with Quick Game and Tournament options. For the time being we will consider only these 5 options:

1. 1 Player - Quick Game
2. 1 Player - Tournament
3. 2 Player - Quick Game
4. 2 Player – Tournament
0. Exit the game

We started off by cleardevice() and a popular function gotoxy() part of the ‘conio.h’ file. It is used to move the cursor to a particular point of the screen for the desired output. A common mis-conception about this function is that the x & y co-ordinates have to passed when in fact the character no. and the line number need to be passed.

Syntax: gotoxy(Char number, Line number);

To capture the user's choice we declared a variable 'int menu_choice' expecting the user to enter the decimal number of the option that he wished to select. We faced 2 major problems at this point. What if the user entered a number greater than or less than the desired values. For this reason we assigned the value to the variable with the help of getche() instead of the obvious 'cin' method. This made sure that only a number between 0-9 was assigned to menu_choice. We also re-called the menu() function using an 'if statement' incase the values were not 0, 1, 2, 3 or 4. The second problem was more devastating. If the user happened to type in anything other than a number, say an alphabet the program entered an infinite loop! To work around this we came up with the idea of changing the data type to char. So by keeping everything else the same and accepting the input as a character the program was working fine.

Game Basics:

We had to start working on the actual game now. This was the point where we had to be the most careful.

After some delicate calculations and long-term planning we decided to represent the characters of the game as part of a single-dimensional array. If we defined this array as 'char t[11]' for the 9 positions of the grid starting from the top-left to the bottom-right, the grid would end up looking like this.

t[1]	t[2]	t[3]
t[4]	t[5]	t[6]
t[7]	t[8]	t[9]

This means that if we assigned t[1], t[2], t[3] & t[8] ='X' and the rest as 'O', the grid would be:

X	X	X
O	O	O
O	X	O

So we had figured out a way to show the user's move on the screen, i.e. if he wanted to place his move at say t[1] and it was Player 1's turn (X) we simply had to assign 'X' to the char t[1] and show the array on the grid. We couldn't have defined t[] as an array of int since it would be unable to display the characters 'X' & 'O'. In order to take the right input we declared a variable 'int ip' i.e. if the user wanted his move to be placed in say cell 5, ip would be assigned 5. Then t[ip]='X' would place X in the 5^{th} cell. This is the main reason why t[] is an array of 11 and not 10 elements since we do not use t[0].

Another important thing that we had to take care of was to prevent the user from inputting a move into a position where a character was already present i.e. 'O' should not be allowed to overwrite a cell that already had 'X'. For this we declared another data member b[10] to buffer the 9 moves that each game would have. We could then check if the value of ip existed in the buffer. If it did then the move was invalid. Just like ip helps the array t[] get the right values, we needed a variable for the array b[]. We decided to declare this variable 'int move_no'.

Object Oriented Design:

We wanted to protect this data from the onset so it was necessary to apply the principles of OOP from now. We declared a class in the main file by the name "ttt". Inside the private section we defined the variables. Simultaneously we declared a constructor ttt() under public: and defined it in a file called GETCONST.CPP. We initialized all the elements of the array t[10] to ' ' (blank spaces) since we didn't want any

characters to show up on the screen when the game began. All the elements of b[] were also initialized to 0. Besides this ip and move_no were initialized to 0.

Evaluation of input:

Once the right value was assigned to menu_choice we had to get the right module called. So we made a function called eval() to evaluate which module to call and placed this in a file called EVAL.CPP. The function eval() accepts a char variable menu_choice by value. The 1st 2 lines of eval() look like this:

```
if (menu_choice=='0')
exit(0);
```

All this does is that if the user has selected 0 the readymade function exit() is called. exit() is part of the STDLIB.H header file so it is also included in the main file. This function is basically designed to quit the main() program when called, although we need to pass an integer variable to it. Similarly if 1 was selected from the menu it would mean that the user wants to play a 1 Player – Quick Game.

Quick Game:

We had to now prepare eval() for a quick game. So if the user selected the 1st or the 3rd option from the menu control would be delivered to this type of code:

```
if (menu_choice=='1' || menu_choice=='3')
{
ttt game;                              //Creates object 'game' of class 'ttt'

     //Repeat the block below 10 times, count declared for the same
     for (int count=1; count<=10; count++)
     {
```

```
            cleardevice();              //Clears Screen
            gotoxy(3,2);    cout << "  ** QUICK GAME **";

            gotoxy(60,1);   cout << "X = Player 1";
            gotoxy(60,2);   cout << "O = Player 2";

            game.show(menu_choice);          //Calls function show() of object game by passing value of menu_choice
            int flag=game.test(count, menu_choice, 0);            //Assigns return value of game.test() to 'flag'
            if (flag==1)                                 //If flag=1 it means that the game is over so exit for-loop
            break;
      }
}
```

The first thing done is the object of the class ttt. This user-defined variable was called 'ttt game'. One more rule of the game is that it can have a maximum of 9 moves. So we want the game to do the following steps 10 times:

1. Show the grid
2. Check for a win / draw condition. If true then finish the game.
3. Move on to take the next move if game not over.
4. Accept a valid move

For this reason we have a for-loop that defines a variable 'int count' to keep track of the move number. This loop is carried out 10 times and important thing to note is that if count=10 was true than it meant that the game was a draw. The first couple of lines after the loop are for displaying some data only. The important bit is in the last 4 lines. First a function called show() is called followed by a function test(). Both these belong to the class ttt. A variable 'int flag' is assigned the return value of the function test(). If the value returned is 1 then the control breaks and exits the for-loop.

Each function is discussed in detail in the coming pages.

Showing the grid:

We had to show the grid repeatedly to the player's after each move, so we put this piece of code in a member function called show() and put it in the TESTSHOW.CPP file. The code is fairly related to the graphical display with a lot of readymade functions being implemented. We wanted to have different pattern fills around the grid for 1 & 2 Player games and have thus passed the value of menu_choice whenever it has been called. Some other functions that have been used are:

- The setfillstyle() function is a part of the 'graphics.h' file and is used to set a fill pattern and colour. The colour and pattern are actually applied to the screen using the bar() function discussed below. C++ supports 16 inbuilt colours and patterns numbered from 0 to 15.
  Syntax: setfillstyle(pattern number, colour number);

- The bar() function is also a part of the 'graphics.h' file and is used to create a rectangle that is filled with a selected pattern and colour set by the setfillstyle() function. Here (x1, y1) is the top-left edge and (x2, y2) is the bottom-right corner. Remember that the resolution of the screen is set to 640 x 480 so the values of the co-ordinates need to be within those limits to prevent errors.
  Syntax: bar(x1, y1, x2, y2);

- The rectangle() function is used to draw the borders of a rectangle on the screen. Here (x1, y1) is the top-left edge and (x2, y2) is the bottom-right corner.
  Syntax: rectangle(x1, y1, x2, y2);

Besides the functions above a couple of gotoxy() statements help in displaying the elements of the array t[] in the right places on the grid. The last 2 lines of the show() function are used for buffering the moves of the game so that no duplication of moves takes place. After this the control returns to eval() and the test() function is called.

Winning Conditions:

The test() function had to be capable of 3 things:

1. Check for a win / draw condition. If true then finish the game.
2. Move on to take the next move if game not over.
3. Accept a valid move

We placed this code in a member function called test(). The member function show() discussed previously was placed in a file called TESTSHOW.CPP along with test().

Here's what the test() function looked like:

```
if (
      (t[1] == 'X' && t[2] == 'X' && t[3] == 'X') ||
      (t[4] == 'X' && t[5] == 'X' && t[6] == 'X') ||
      (t[7] == 'X' && t[8] == 'X' && t[9] == 'X') ||
      (t[1] == 'X' && t[4] == 'X' && t[7] == 'X') ||
      (t[2] == 'X' && t[5] == 'X' && t[8] == 'X') ||
      (t[3] == 'X' && t[6] == 'X' && t[9] == 'X') ||
      (t[1] == 'X' && t[5] == 'X' && t[9] == 'X') ||
      (t[3] == 'X' && t[5] == 'X' && t[7] == 'X')
   )
      {cout << " PLAYER 1 WINS!!";
      return(1);}
```

This bit of code was designed to check if Player 1 had his marker 'X' in any winning combination. Translating the first line of code says that if t[1], t[2] & t[3] are all equal to 'X' than the if statement is true. We also return(1) to indicate that a result has been achieved. This is assigned to flag and the for-loop in eval() breaks. Similar code was also duplicated to check O's winning conditions. Now if neither had won the game it could mean 2 things. Either all the 9 moves aren't complete or if they are complete than the game was a draw. To check if the

game was a draw we simply had to check if count=10 was true. This is the reason count is passed to test(). The final code that checks for a draw looks like this:

```
else if (count == 10)                    //If it's the 10th turn in the game
     {
     cout << " The game was a Draw!";
     return(1);
     }
```

So if the game has still not been completed we need to accept moves from the player's alternately. Consider this code:

```
else if (count%2 != 0)
          {
          cout << " Player 1's turn.";
          changecursor(cursorX);              //Make the mouse pointer look like an X
          showmouseptr();                     //Show the mouse pointer
          get();                              //Get the player's move
          hidemouseptr();                     //Hide the mouse pointer
          t[ip]='X';                          //Assign the position selected by the player an 'X'
          return(0);                          //Game not over yet
          }
```

The first if condition checks if it's an odd turn or not since Player 1 will play turns 1, 3, 5, 7 & 9 and the rest will be played by Player 2 or the computer. We first incorporate the readymade mouse functions from Yashwant Kanitkar's book 'Let Us C' and include 'dos.h' as a header file. We also globally declare the bitmaps of 2 mouse cursors – one that looks like an 'X' and the other an 'O'. What we'd basically like to do is change the cursor to an X or an O depending on whose turn it is so that the player can simply move the cursor and click on the right place to stick his respective character to the grid. A detailed explanation of how to make these cursors is also defined in the same book.

Mouse Functions:

Mouse functions can be accessed by setting up the AX register with different values i.e. service numbers and issuing interrupt 33h. initmouse() function is used to initialize the mouse. It checks if the mouse driver has been loaded or not by issuing interrupt 33h, service number 0 and reports the status to main. It returns 0 if the drivers have not been loaded and returns FFFFh if the drivers have been loaded. If the mouse is not initialized successfully then the closegraph() function is called which unloads the graphics drivers and issues an error saying "ERROR: Unable to initialize mouse drivers" and finally exits the program. If the mouse drivers have been loaded successfully then the mouse would be successfully initialized.

showmouseptr() function is used to display the mouse pointer on the screen. It uses service number 1. This function returns nothing. hidemouseptr() is used to hide the mouse pointer. It uses service number 2. This function returns nothing. changecursor() function is used to change the mouse cursor. It uses service number 9.This function accepts, as parameter, a pointer to the array of cursor shape (cursorX or cursor0 in this case).

The mouse cursor in graphics mode occupies a 16 by 16 pixel box. By highlighting or de-highlighting some of the pixels in this box, a mouse cursor of the desired shape can be made. The 1's in the mouse pointer bitmap indicate that the pixel would be drawn, whereas the 0's indicate that the pixel would stand erased. To change the pointer shape, a 64 byte bitmap is needed. The first 32 bytes contain a bit mask which is first ANDed with the screen image, and then the second 32 bytes bit mask is XORed with the screen image.

getmousepos() function is used to get the position of the mouse, i.e. the x and y coordinates of the mouse pointer, and status of the mouse buttons, i.e. it determines whether the button has been clicked or not. It uses service number 3. This function accepts, as parameters, pointers to the integer variables x, y and button. bx returns the mouse button status, 0 when left button is down, 1 when right button is down, 2 when center button is down. cx returns the value of the x coordinate and dx returns the value of the y coordinate.

After changecursor() and showmouseptr() we need to create a member function that can access the ip variable from the private section of the class to capture the player's move. Please refer to the get() function defined in module 4.2 for better understanding the following matter. This function defines the code for capturing the Player's input for his move. The 3rd line of code assigns 0 to the variable ip. The control then passes to a while loop which is designed to repeat itself till the user makes a valid input. The getmousepos() function monitors the X & Y co-ordinates of the mouse and checks if the left mouse button is in the 'up' or 'down' position.

The next set of statements in the same loop state that if the left mouse button is clicked within a certain range of X or Y coordinates then depending on the combination 'ip' is assigned a value. For example:

```
if(x>199 && x<267)              //If click between 200 & 266 of X co-ordinate enter block
                {
                     if(y>84 && y<143)          //If click between 85 & 142 of Y co-ordinate
                     ip=1;                      //Player wants to play at position 1

                     else if(y>149 && y<207)    //If click between 150 & 206 of Y co-ordinate
                     ip=4;                      //Player wants to play at position 4

                     else if(y>213 && y<281)    //If click between 214 & 280 of Y co-ordinate
                     ip=7;                      //Player wants to play at position 7
                }
```

The code above is applicable only to the shaded blocks in the diagram below.

1	2	3
4	5	6
7	8	9

Please note that the x1 and x2 co-ordinates for all the 3 blocks remain the same. Only the y1 & y2 coordinates will change based on their distance from the top of the screen. To explain in a gist, the code defines that if a mouse-click occurs between the co-ordinates (x1,y1) & (x2,y2) then assign 'ip' the block number. All the 9 blocks have absolutely unique combination so the code is safe to use.

After the get() function is over, the control passes back to the test() function and the hidemouseptr() function is called. After this t[ip] is assigned the character 'X' or 'O' depending on who played last. The final line of code is return(0) to satisfy 'flag' in the eval() function. This is to indicate that the game is not yet over.

1 Player Games:

The main difference between a 1 & a 2 player game is that in a 1 player game the second player's role is fulfilled by the computer. To begin with we will define a small function get_computer_move() that will be called instead of get() if it's a 1 player game. What goes on inside the get_computer_move() is not important right now. Let us just assume for the time being that it somehow assigns a random value to 'ip'. Consider the last part of the test() function:

```
else
     {
     if (menu_choice=='1' || menu_choice=='2')   //If it's a 1 Player game
          {
          get_computer_move(count,1);            //Get the computer's move
          }
     else                                        //If it's a 2 Player game
          {
          cout << " Player 2's turn.";
          changecursor(cursor0);                 //Make the mouse pointer look like an 'O'
          showmouseptr();                        //Show the mouse pointer
          get();                                 //Get the player's move
          hidemouseptr();                        //Hide the mouse pointer
          }
     }
```

This piece of code is for Player 2's turn. By checking the value of 'menu_choice' we can decide whether it's a 1 or a 2 player game. If it's a 1 player game then the get_computer_move() function is called or else the get() function is called just like in Player 1's turn the only difference being that this time changecursor(cursor0); makes sure that the cursor will look like a O.

After the test() function finishes execution, control passes back to eval() and then to main().

Tournament:

After the user selects to play a tournament from the main menu he is take to another menu, where he must enter the number of games he would like to have in the tournament. To make sure that a valid input is accepted we have declared some variables. In a tournament the basic methodology remains the same as the quick game. However, we need to put the existing code of a quick game in an additional for-loop so that the entire process repeats for the number of games in the tournament. The code resides in eval() itself and is fairly easy to understand; only thing to remember is that the object is re-defined every time a new game is played so the constructor reinitializes all the variables of the class to their default values.

wins() & score():

We wanted a feature whereby the user could be given a scorecard at the end of his tournament stating the number of wins, draws the players had amongst them. For this we defined 2 functions wins() and score() and placed them inside a file called TOUR.CPP. We defined 3 variables p1wins, p2wins & draws globally. The job of wins() was to increment each of these variables whenever Player 1 won, Player 2 won or a draw was the result. Obviously we wanted to make sure this happened as soon as any of the mentioned scenarios occurred. So we placed the statements in the proper places in the test() function and passed an integer to wins() at the point of a result. Depending on the number wins() received it incremented one of the variables. We also had to re-initialize these variables to 0 as soon as a new tournament was to begin because of which we redefined them in the tournament module in eval() as extern variables. The score function is called by eval at the end of a tournament. It simply displays some statistics based on the number of wins, draws, etc.

get_computer_move():

We wanted a function to get the Computer's move. So we developed a function called get_computer_move() and placed it in a file called AI.CPP which we then included in main(). Since the values or the variables count and game_no were needed in this function, we passed them as arguments. This function, according to the level of difficulty, selected by the user, calls the appropriate functions. Since the game includes 4 levels of difficulty, we wanted to have a function for each level. Also we wanted to make sure that the move, selected by any of the functions, is not played before (i.e. the position is not already taken). If that move has already been played, the get_computer_move() function is called again.

easy():

When difficulty level is 1, the game is in easy mode and so a function called easy() was defined. In this function, random(), part of 'stdlib.h', was called to generate a random number between 1 and 9.

block():

We wanted a function which would enable the Computer to block the Player from winning and also play a winning move. For this a block() function was defined. The first thing we wanted to do was to have a variable which would be set to 1 whenever a move was played in the block() function so that no other function would be called. For this a variable called 'int flag' was declared in the private section of the class and was initialized to 0 in get_computer_move(). Also a variable 'char check' was declared in private section of the class which accepted, as parameters, 'X' or 'O'.

Here's what the block() function looks like:

```
if(   (flag==0) && (t[1]==check) )
      {
        if( (t[2]==check) && (t[3]==' ')     )     {ip=3; flag=1;}
```

```
else if(   (t[3]==check) && (t[2]==' ')     )     {ip=2; flag=1;}
else if(   (t[4]==check) && (t[7]==' ')     )     {ip=7; flag=1;}
else if(   (t[7]==check) && (t[4]==' ')     )     {ip=4; flag=1;}
else if(   (t[5]==check) && (t[9]==' ')     )     {ip=9; flag=1;}
else if(   (t[9]==check) && (t[5]==' ')     )     {ip=5; flag=1;}
}
```

This bit of code is used to check for a blocking condition. If the value of check is 'X', then the above code checks that if t[1] and t[2] are equal to 'X' and t[3] is blank, then the Computer should play at t[3] and block the Player from winning. If the value of check is 'O', then the above code checks that if t[1] and t[2] are equal to 'O' and t[3] is blank, then the Computer should play at t[3] and win the game. Similarly, the blocking condition is checked for each line. Whenever a move is made, the value of flag changes to 1.

When difficulty level is 2, the game is in medium mode. In this mode, we wanted the Computer to only block the Player from winning and not win itself. So, we called the block() function by passing 'X' as the parameter, i.e. block('X').
If no move was made to block the Player then easy() function is called to play a random move.

When difficulty level is 3, the game is in advanced mode. In this mode, we wanted the Computer to block the Player as well as win games. So we first call the block('O') function which checks whether there is a win condition for the Computer (i.e. two O's in a line) and plays the move to make the Computer win (i.e. the third O). If this happens then the value of variable flag changes to one and no other function is called. However, if the value of flag is equal to zero, the block('X') function is called which blocks the moves of the player. If no move was made to block the Player then easy() function is called to play a random move.

expert(), playfirst() & playsecond():

When difficulty level is 4, the game is in expert mode. In this mode, we wanted the Computer to play with some strategy. So we developed an expert() function which defined the Computer's move. In this function the Computer's moves are dependent on the Player's moves. In a tournament, if the alternate game play is ON (i.e. the value of the variable alterplay is 5), then the Player and the Computer play

alternately. In odd games, the Player plays first and in even games, the Computer plays first. If alternate game play is OFF (i.e. the value of the variable alterplay is 6) then, then the Computer always plays second. So we developed two separate functions, playfirst() which defined the moves of the Computer when it has to play first and playsecond() which defined the moves of the Computer when it has to play second. To determine whether the playfirst() or playsecond() functions have to be called we needed the value of the variable game_no. Also to determine the turn, we needed the count variable. So we decided to accept these two variables as parameters. As with other difficulty modes, in this function also the Computer blocks the moves of the Player and also tries to win. So first, we called the block('O') function which checked for a winning condition for the Computer. If this happened then the value of variable flag changed to one and no other function was called. However, if the value of flag was equal to zero, the block('X') function was called which blocked the moves of the player. If no move was made to block the Player then, depending on whether it is an odd game or an even the game, the playfirst() and playsecond() functions are called. playfirst() and playsecond() functions need the value of the variable count to determine the turn. So we pass this value as a parameter.

The expert() function looks like this:

```
void ttt::expert(int count, int game_no)
{
block('O');                              //Checks for a blocking condition
if(flag==0)
block('X');                              //Checks for a blocking condition
if (alterplay=='5')                      //If alternate game play is on
      {
      if (game_no%2 != 0)                //Odd game
      playsecond(count);
      else                               //Even game
      playfirst(count);
      }
else                                     //If alternate game play is off
      playsecond(count);
}
```

playfirst() function is executed only if flag=0, i.e. no move has been made in the block() function. We did not want the Computer to play the same moves again and again. So we defined 5 different game plays for the Computer. We used the random() function, part of stdlib.h, to choose from these 5 game plays. We defined a variable ‘int select’ which stored the value of the game play selected randomly and then using switch selected that particular game play. We also did not want the Computer to play the same moves in consecutive games. So the random number was selected until it was equal to 0 or equal to the previous game number stored in the variable ‘int prevgame’. However, we wanted this to happen only when a new game started. For this a variable called ‘int newgame’ was declared which was initialized to 0 only when a new game started.

So the code,

```
if(newgame==0)                              //Selects randomly the game to be played
      {
      do
            {
            select=random(5);               //Assigns the selected game no. to select
            }
      while(select==0 || select==prevgame);   //0 & the no. selected in the previous game is not allowed
      newgame=1;
      prevgame=select;
      }
```

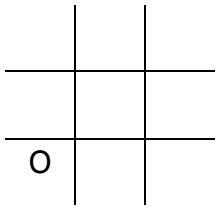
selected a new game play only when the game is a new game. Once a proper number has been selected, we set the variable newgame to 1 and assign the value selected in this game to the variable prevgame. This will be used in the next game. We wanted to store each move of the Player. So we defined a variable called ‘int playone’ which is assigned the moves of the Player in the test() function.

Example: We assume that the Computer selects the number 1, randomly. Thus, the variable select is assigned 1 and using switch the first case is selected.

```
switch(select)                                  //Chooses from the five games using select
          {
          //1st game
          case 1:    if (count==1)                                  ip=7;

               else if(count==3)
                    {
                    if(t[1]=='X' || t[9]=='X')                     ip=3;
                    else if(t[3]=='X' || t[4]=='X')                ip=9;
                    else if(t[2]=='X' || t[8]=='X' || t[6]=='X')ip=1;
                    else if(t[5]=='X')                             ip=6;
                    else easy();
                    }

               else if(count==5)
                    {
                    switch(playone)
                         {
                         case 5: if(t[1]=='X')                     ip=9;
                              else if(t[9]=='X')                   ip=1;
                              else easy();
                              break;

                         case 8: if(t[3]=='X')                     ip=1;
                              else if(t[4]=='X')                   ip=3;
                              else easy();
                              break;

                         case 4: if(t[2]=='X' || t[6]=='X' || t[8]=='X')  ip=5;
                              else if(t[5]=='X')                   ip=9;
                              else easy();
                              break;

                         case 1: ip=9;
                              break;

                         default:easy();
                              break;
```

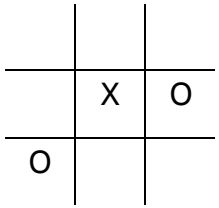
```
            }
      }

else if (count==7)
      {
      easy();
      }

else if(count==9)
      {
      for (int i=1;i<=9;i++)
            {
            if(t[i]==' ')
                  {
                  ip=i;
                  break;
                  }
            }
      }
break;
```

When the value of count is equal to 1, ip is assigned 7. This means that the Computer plays at the position number 7 in its first move.

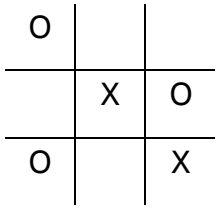


When count is equal to 2, it is the Player's turn. We assume that the player plays at position number 5. This value is stored in the variable playone and the value of count is incremented. When count becomes 3, the if statement checks which move the Player has played

and accordingly selects the Computer's move. The statement if(t[5]=='X') evaluates to true and ip is assigned 6. The game now looks like this:



Count is now 4. We assume that the Player plays at position 9. This value is stored in variable playone and count is incremented. Call to the block() function will make the Computer block the Player from winning by playing ay position 1. If in some case, the block() function does not make any move, then the control comes to the statement else if(count==5) and from the switch block chooses the case whose value is equal to the value of playone. By checking the conditions in that particular case, an appropriate move is played. The game now looks like this:



We assume that the player now tries to block the Computer by playing at position 4. The Computer now has the choice of playing at positions 2, 3 or 8. Since not much winning choices are available, call to the easy() is made and a random number from 2, 3, 8 is selected. If the number selected is 3, then the game looks like this:

O		O
X	X	O
O		X

The value of count is now equal to 8, and we assume that the player blocks the Computer by playing at position 2. Count is incremented to 9 and now, since only position 8 is available, the Computer plays at position 8 and the game ends in a Draw.

O	X	O
X	X	O
O	O	X

playsecond() function is executed only if flag=0, i.e. no move has been made in the block() function. When the Computer is playing second, we want the Computer to play the moves depending on the moves of the Player. So we defined 9 different game plays for the Computer, one for each position. Depending on the first move of the Player, a particular game play is selected. So we required a variable which would store the first move of the player. Thus, we defined the variable 'int move' and in the test function, this variable is assigned the first move of the Player. This is needed to make sure that in all the turns of a game, moves are made from a single game play. Using switch, we select the game play depending on the Player's move. As the Computer's move is based on the Player's move, we required a variable to store each move of the Player. So, we defined the variable 'int playtwo' which is assigned the moves of the Player in the test() function.

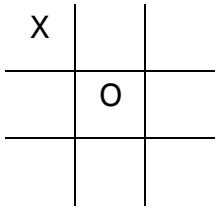
We have explained the working of this function with the help of an example as below.

Example: We assume that the Player plays at position 1 in his first turn, i.e. when count = 1. Therefore, move is assigned one and from the switch block, the first case, i.e. case 1, is selected.

```
case 1: if (count==2)                                          ip=5;
     else if(count==4)
          {
          if(t[3]=='X' || t[6]=='X')                           ip=2;
          else if(t[4]=='X')                                   ip=7;
          else if(t[7]=='X' || t[9]=='X')                      ip=4;
          else if(t[2]=='X')                                   ip=3;
          else if(t[8]=='X')                                   ip=6;
          else easy();
          }

     else if(count==6)
          {
          switch(playtwo)
               {
               case 8: if(t[1]=='X' || t[6]=='X')               ip=7;
                    else easy();
                    break;

               case 6: if(t[7]=='X')                           ip=2;
                    else if(t[9]=='X')                         ip=3;
                    else easy();
                    break;

               case 4: if(t[8]=='X')                           ip=7;
                    else easy();
                    break;

               default:easy();
                    break;
               }
          }

     else if (count==8)
```

```
        {
        switch(playtwo)
              {
              case 6: if(t[2]=='X')                ip=8;
                    else easy();
                    break;

              case 8: if(t[4]=='X')                ip=9;
                    else easy();
                    break;
              default: easy();
                     break;
              }
        }
break;
```

When the value of count is equal to 2, the Computer plays at position 5. The game now looks like this:



When count is equal to 3, it is the Player's turn. We assume that the player plays at position number 9. This value is stored in the variable playtwo and the value of count is incremented. When count becomes 4, the if statement checks which move the Player has played and accordingly selects the Computer's move. The statement else if(t[7]=='X' || t[9]=='X') evaluates to true and ip is assigned 4. The game now looks like this:

X		
O	O	
		X

Count is now 5. We assume that the Player blocks the Computer's move by playing at position 6. This value is stored in variable playone and count is incremented. Call to the block() function will make the Computer block the Player from winning by playing ay position 3. If in some case, the block() function does not make any move, then the control comes to the statement else if(count==6) and from the switch block chooses the case whose value is equal to the value of playtwo. By checking the conditions in that particular case, an appropriate move is played. The game now looks like this:

X		O
O	O	X
		X

We assume that the player now tries to block the Computer by playing at position 7. The Computer now blocks the player by playing at position 8. Then the game looks like this:

X		O
O	O	X
X	O	X

The value of count is now equal to 9, and since only position 2 is available, the Player plays at position 2 and the game ends in a Draw.

X	X	O
O	O	X
X	O	X

Alternate Game Play:

Our current game always gave Player 1 the first chance to play in a tournament. This was a little unfair to the second player especially since playing first gave you one extra turn. This was a very good advantage and to make the game fair we decided to add feature whereby the player's would play first alternately in a tournament. The new function alternate() placed in TESTSHOW.CPP works on the same principles as the original game but has some minor changes here and there to make the feature come alive. Although this feature can be switched OFF it is turned ON by default whenever the game starts.

Settings:

We had to provide a place where the players could change the difficulty levels and switch alternate game play on or off. So we added another option in the main menu called Settings. Necessary changes were made in eval() to call the function settings() placed in SETTINGS.CPP. This function called another menu-driven program where the user could select from a list of options. Again we made sure that the user entered only valid values.

VI. Marketing, Advertisement, Packaging & Technical Support

Here are a few steps taken by us to make sure the game reached out to the customer.

Marketing:

- A demo version has been developed, with limited functionality, so that the users get a feel of the game before investing in the full version.
- A webpage has been developed for downloading the demo version of the game. The URL is www.dicelords.com/ttt
- Minimal charges so that the game is affordable to everyone.
- Information on future releases to be provided on the webpage.

Advertising:

- The Internet is the best medium possible to reach out to the type of audience that would play such a game. For the same reason, we are in the process of collaborating with major game sites to help us distribute TTT's Demo version.
- We have already submitted the website to popular search-engines like Google, MSN, Yahoo!, Excite, Indiatimes & Rediff although it may take a while for us to get listed.
- A lot of banner-exchange programs are also available on the internet. We wish to tie-up with one of these and are on the lookout for the best deal.
- Advertisements are also the primary source of income for most of the websites. However they require that the website attracts a lot of hits each day. We wish to recruit some space-selling personnel once we start getting the right number of visitors.
- We have sent e-mails to various local computer/gaming magazines like Digit, Computer's @ Home, and PC Quest to feature a demo version of our game on the CD that goes with their monthly issues. We are yet to receive replies from all of them.

Packaging:

- The game was packaged in a self-extractor for easy installation at the user-level.
- The packages are extremely small in size and can be conveniently downloaded directly from the website. This avoids the cost of distribution media like 1.44 Mb Floppy-disks or CD-ROM's which ultimately helps us provide the software even cheaper to the customers.
- Since we don't have the infrastructure or the budget to accept online payment right now, we will have to rely on the customer's mailing us a Cheque or a Demand Draft to our head-office in India. We will start accepting credit-cards on our website as soon as we break-even.
- Proper documentation was provided with the game so that the user could answer most of his queries by reading them. A ReadMe.txt file explained the Overview, System Requirements, Uninstallation, Bug Report, Disclaimer of Warranty & Technical Support.
- A License.txt file explained the 'End User License Agreement' (EULA) to explain to the user the legalities behind using this software. The same document was also presented as the first step of the installation procedure.
- A FAQ's.txt file answered some of the most Frequently Asked Questions that we came across in the Alpha & Beta Release.

Technical Support:

- A dedicated e-mail account for TTT has been created where the customer's can mail us their queries, comments, suggestions, comments, etc. The e-mail address is ttt@dicelords.com and it has been repeatedly mentioned in all the packaged documents. We want to make all efforts possible to make sure that no e-mail remains un-attended and that all queries are answered satisfactorily.
- If there are any problems with the understanding of the code, the developers can be contacted personally at the addresses listed below:

  Dipika Chawla – dipikachawla@hotmail.com

  Pushkar Modi - ronik@eth.net

  Preeti Goyal – goyal_preeti@hotmail.com

VII. Cost Analysis

Here's a brief idea of how much development time the project required, how much it cost us and how soon we hope to break even.

Time Required:

Number of days required	: 13 days
Hours worked per day	: 08 hours / day
Number of employees	: 03 employees
Total number of hours worked	: 13 days x 8 hours x 3 employees = 312 hours

Project Expenditure:

Wage per day	: Rs. 200/-
Total wage budget	: Rs. 200 x 13 days x 3 employees = Rs. 7800/-

Break Even Estimates:

Cost of a single license of the game	: Rs. 50/- for India US $5 for outside India ($1 USD = Rs. 48 INR)
No. of licenses to be sold in India	: Rs. 7800 / Rs. 50 = 156 licenses
No. of licenses to be sold outside India	: Rs. 7800 / (Rs. 48 x $5) = 32.5 ~ 33 licenses

VIII. References & Tools

Books:

1. Let Us C, 3rd Edition – by Yashwant Kanitkar
2. Object-Oriented Programming with C++, 2nd Edition - by E Balagurusamy
3. Software Engineering, – by Roger Pressman

Websites:

1. www.brackeen.com
2. www.programmersheaven.com
3. www.free2code.net
4. www.devpro.netfirms.com
5. www.funducode.com
6. www.codeproject.com

Tools:

1. Turbo C++ 3.0
2. Adobe Photoshop 7.0
3. CuteFTP 5.0 XP
4. Microsoft Word 2002
5. WinRAR 7.0

IX. Conclusion

"I feel like a man who planned to fix the porch steps of his house, then decided to work on the porch door, then the porch ceiling and finally ended up rebuilding his whole house." - A famous quote by Mr. Yashwant Kanitkar – author of a number of C & C++ books. At the end of this project we feel very much like him, having started to write a simple 2 player game based on the classic X & O we ended up with a game that boasted of options like 1 & 2 Player game, Quick Game or Tournament, 4 difficulty levels of AI, GUI, Mouse support and various other features.

We never imagined that our minds could think the way they did or that our fingers would type code at the speed of thought... well, almost **J**. Staring at the computer for endless hours was definitely not easy, especially when our compiler showered us with errors that seemed hopelessly unexplainable. We realized the importance of teamwork but most importantly we realized that a man's best friend can only be his determination.

We have a lot of people to thank for this successful venture; especially Dr. S. S. Mantha, our course coordinator for implementing such an up-to-date curriculum for the B.Sc. IT course due to which we got an opportunity to exploit our undiscovered potential in the field of Software Engineering. We would like to thank Prof. Prasad Padalkar for helping us choose the right kind of project; this project wouldn't have been possible in the absence of his constant effort to convert us into thorough professionals. We are also very grateful to Prof. Sonal Mahajan for sowing the seeds of C++ & Object Oriented Programming in our minds; we owe her all that we know about these wonderful concepts. Last but not the least our parents for being there when we needed them… always & forever.

Best Regards,
Pushkar Modi