Project 1

Title

Tic-Tac-Toe V1.0

“The classic strategy game of old”

Class

CSC 17A, Section 47975

Author:

John Palacios

Outline of contents

1 Introduction

Tic-Tac-Toe is a simple game. As a human, you do not need to know much about anything to play. You draw a three by three table, you mark your space with an “X” or an “O” and then it is your friend’s turn to make his mark. As human beings, it is easy to forget the complexity of the underlying logic of such a simple game.

I chose Tic-Tac-Toe because this game presents some inherent challenges, and offers opportunities for greater challenges; especially in programming the game logic. There are many decisions to be made; many design philosophies to consider when writing any game. I decided long ago that I would focus on these decisions by writing the simplest game coming to mind, applying as many programming concepts as I could; making the logic much more complicated than need be, but providing the challenges I need to overcome if I am to grow.

In short: I wrote Tic-Tac-Toe as a challenge to myself.

2 Summary

Statistics at a glance

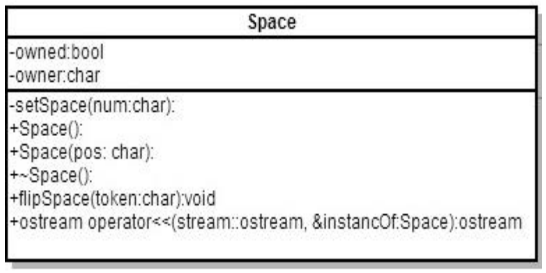
Number of variables	**18**	**Number of ADTs**	**2**
Single for loops	8	Nested for loops	8
If or if/else statements	40	Do-while loops	11
Data members	28	Methods	35
Static members	8	Classes	3
Menus	2	files	9
Lines of code	396	Hours spent in development	~16+

Most complex variable: the Game class’ field pointer which is a pointer to an instance of the Board class, which has a dynamic two dimensional array of Space objects.

This game can be best described as the aggregate of three classes. The Game object has a Board object which has a two dimensional dynamic array of Space objects. The Space class is the simplest unit of my game, and so I will start there.

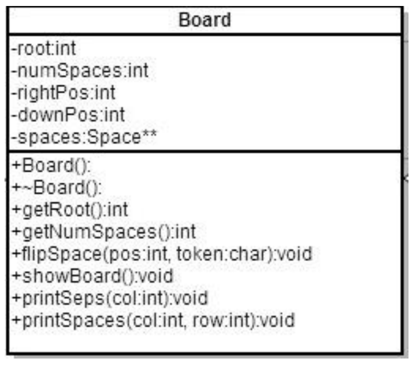
Classes in more detail

2.1 The Space class


Space
-owned:bool
-owner:char
-setSpace(num:char):
+Space():
+Space(pos: char):
+~Space():
+flipSpace(token:char):void
+ostream operator<<(stream::ostream, &instancOf:Space):ostream

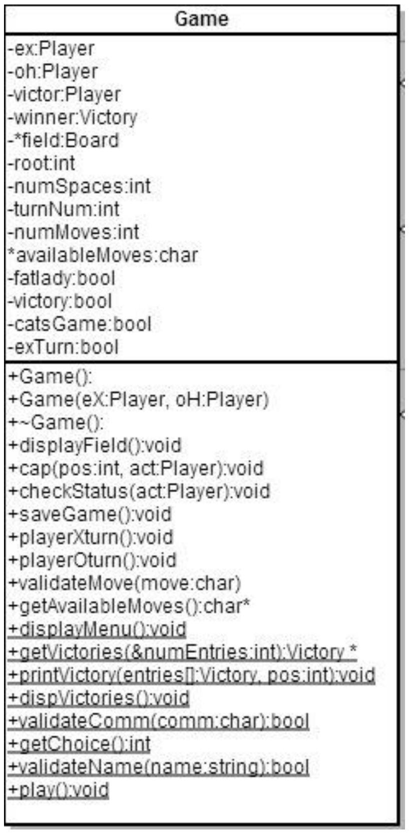
This class is the basic element of my game. It encapsulates a character and a Boolean value to signify its owned state (to prevent players from capturing another player's square). Aside from the default constructor and standard destructor, this class has a private setSpace method to be used only during initial set up of the board, a flipSpace method used to set up the board or capture that space. This class seemed too simple to me, so I also overloaded the ostream class's stream extraction operator to specifically return the owner of the space's character (instead of writing a simple "getOwner" method). Overloading the ostream operator was a bigger challenge than I originally anticipated; it hadn't been covered in class and I've never tried it before, but a bit of trial and error saw me through. I declared the Game class a friend since it orchestrates the behavior of the space class through its own dynamically allocated Board object.

2.2 The Board class



The board is the heart of my game. It keeps a few integral statistics dealing with its size and printing and a dynamically allocated two dimensional array of Space objects to track player moves. In addition to constructor and destructor methods, this class returns its root and numSpaces values to the calling game object, flips a given space object in its **spaces data member to a given character, prints the spaces in a familiar format to the screen for human players (using a couple of support methods: printSeps and printSpaces). The Game class is a friend of the Board to better facilitate control of the dynamically allocated Board object.

2.3 The Game class



The brain of my game is the Game class. The Game class has: two Player data structures, one Victory data structure, a pointer to the dynamically allocated Board object, the Board objects' root and numSpaces, the turn number, the number of available moves, a dynamic char array of available moves, and four Boolean variables: fatlady (it's over when the fatlady), victory, catsGame, and exTurn. The Game class has: two constructors; one overloaded for named two player play, a displayField method which calls the field Board objects' showBoard method, a cap method to capture the space specified by a player, the checkStatus method which searches the board for patterns indicating victory, saveGame method to save the Victory structure to a binary file, a getExTurn method which returns the exTurn Boolean value, two player turn methods (one for each player), support method validateMove, getAvailableMoves which returns a pointer to a dynamically allocated char array representing the spaces that haven't been captured yet, and several static members: displayMenu which presents a menu interface for the user, getVictories which returns a pointer to a dynamically allocated array of Victory structures read in from the binary Victories.dat file(created by the saveGame method), printVictories to print a given element of the aforementioned Victory array, displayVictories which offers the user a menu to browse the Victories array one element at a time, validateComm is used in displayVictories to validate the users' input, getChoice is as advertised, validateName is used in game set up, and the most important member: play, from which the entirety of the game is run from.

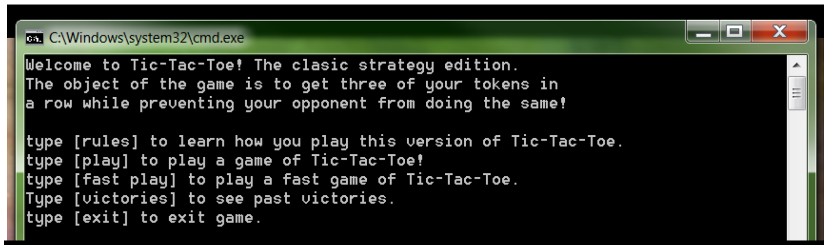
Final thoughts on this project

I foolishly thought that this project was going to be simple, even building it out of an aggregate of classes with dynamic and variable cardinality. I introduced complexity in two ways: I wanted to make the board size scalable and I wanted to include some of the more advanced topics not covered in class yet. To facilitate a variable board size in the future, I coded all board allocations and display functions in terms of a variable called "root" which is passed between the Board and Game class objects. Originally; I thought I might allow the player to choose how many squares were to be on the board. As the deadline grew close, I resolved to stick to regular old tic-tac-toe. I started project 1 the week of the midterm, and at that point class had not gone past chapter 13, but I envisioned a game of tic-tac-toe built from an aggregate of objects; calling the methods of dynamically allocated Space objects through a Board object in an instance of a Game object run from static methods of the Game class itself. Ultimately that is what the core of my project is; something which is not covered in the text book, though admittedly, it would only require a little imagination to expand on pointers and classes to be able to create such a mad scheme.

3 Description

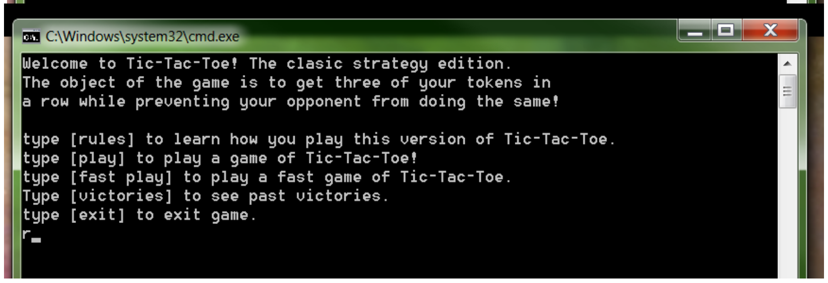
3.1 General description of user experience.

The entirety of my solution to tic-tac-toe runs from a single line in main which calls the “play();” static member of my game class. This method presents the user with a greeting and a menu of choices.



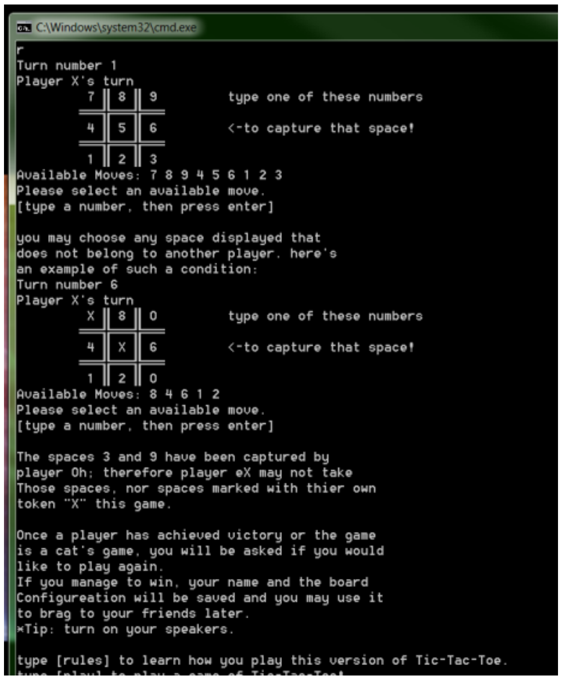
```
C:\Windows\system32\cmd.exe
Welcome to Tic-Tac-Toe! The clasic strategy edition.
The object of the game is to get three of your tokens in
a row while preventing your opponent from doing the same!

type [rules] to learn how you play this version of Tic-Tac-Toe.
type [play] to play a game of Tic-Tac-Toe!
type [fast play] to play a fast game of Tic-Tac-Toe.
Type [victories] to see past victories.
type [exit] to exit game.
```

Unlike textbook menu systems, this one accepts either single letter or partial phrase entries from the user.



```
C:\Windows\system32\cmd.exe
Welcome to Tic-Tac-Toe! The clasic strategy edition.
The object of the game is to get three of your tokens in
a row while preventing your opponent from doing the same!

type [rules] to learn how you play this version of Tic-Tac-Toe.
type [play] to play a game of Tic-Tac-Toe!
type [fast play] to play a fast game of Tic-Tac-Toe.
Type [victories] to see past victories.
type [exit] to exit game.
r_
```

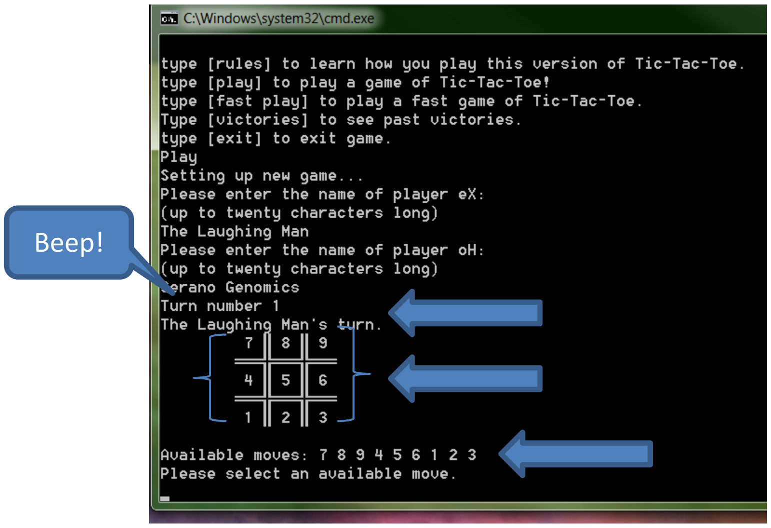
For example: user types “r” or “rules” and the system prints the rules.



```
C:\Windows\system32\cmd.exe
r
Turn number 1
Player X's turn
         7 ║ 8 ║ 9          type one of these numbers
        ═══╬═══╬═══
         4 ║ 5 ║ 6          <-to capture that space!
        ═══╬═══╬═══
         1 ║ 2 ║ 3
Available Moves: 7 8 9 4 5 6 1 2 3
Please select an available move.
[type a number, then press enter]

you may choose any space displayed that
does not belong to another player. here's
an example of such a condition:
Turn number 6
Player X's turn
         X ║ 8 ║ 0          type one of these numbers
        ═══╬═══╬═══
         4 ║ X ║ 6          <-to capture that space!
        ═══╬═══╬═══
         1 ║ 2 ║ 0
Available Moves: 8 4 6 1 2
Please select an available move.
[type a number, then press enter]

The spaces 3 and 9 have been captured by
player Oh; therefore player eX may not take
Those spaces, nor spaces marked with thier own
token "X" this game.

Once a player has achieved victory or the game
is a cat's game, you will be asked if you would
like to play again.
If you manage to win, your name and the board
Configureation will be saved and you may use it
to brag to your friends later.
*Tip: turn on your speakers.

type [rules] to learn how you play this version of Tic-Tac-Toe.
```

Though limited to character driven interface and graphics, I found a couple of interesting ways to spice up the game, as demonstrated in this screenshot of the [rules] option.

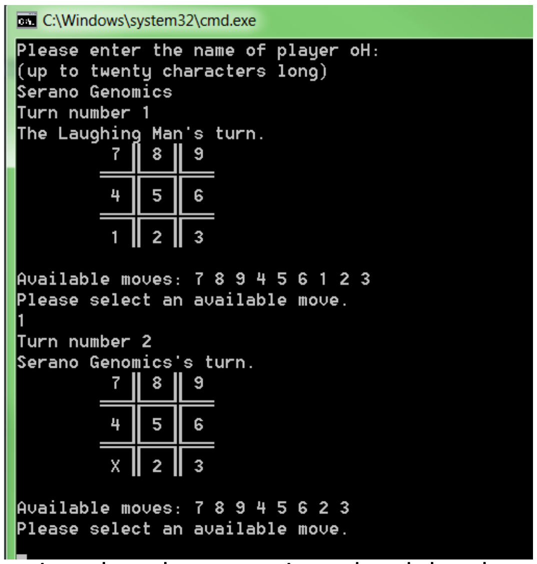
There are two options available for game play: [play], and [fast play]. Either one of these options allows the user to play, but [fast play] is just a generic tic-tac-toe game.



```
type [rules] to learn how you play this version of Tic-Tac-Toe.
type [play] to play a game of Tic-Tac-Toe!
type [fast play] to play a fast game of Tic-Tac-Toe.
Type [victories] to see past victories.
type [exit] to exit game.
Play
Setting up new game...
Please enter the name of player eX:
(up to twenty characters long)
The Laughing Man
Please enter the name of player oH:
(up to twenty characters long)
Serano Genomics
Turn number 1
The Laughing Man's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         4 ║ 5 ║ 6
        ═══╬═══╬═══
         1 ║ 2 ║ 3

Available moves: 7 8 9 4 5 6 1 2 3
Please select an available move.
```

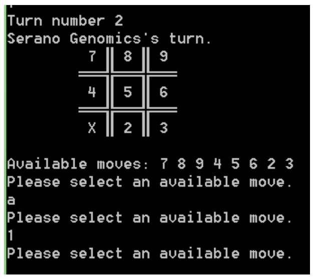
Option [play] allows the user to play a game of tic-tac-toe which keeps track of the player's names and saves the name of the victorious player and that board configuration in a binary data file I have included with the project called "victories.dat." These saved games are available for viewing via option [victories].

Notice several features: 1. when the system starts the game, a beep is issued. 2. The display shows the turn number and the player's name. 3. The Board is constructed using a set of nested loops which print an extended set of ASCII characters (the frame of the board is actually characters 186, 205, and 206 in the extended ascii table I found at http://www.asciitable.com/). 4. A list of available moves is displayed for convenience.



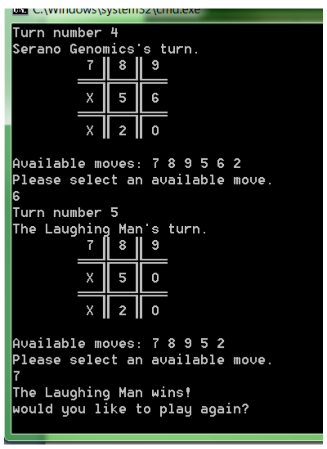
```
Please enter the name of player oH:
(up to twenty characters long)
Serano Genomics
Turn number 1
The Laughing Man's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         4 ║ 5 ║ 6
        ═══╬═══╬═══
         1 ║ 2 ║ 3

Available moves: 7 8 9 4 5 6 1 2 3
Please select an available move.
1
Turn number 2
Serano Genomics's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         4 ║ 5 ║ 6
        ═══╬═══╬═══
         X ║ 2 ║ 3

Available moves: 7 8 9 4 5 6 2 3
Please select an available move.
```

The player simply types in a number to capture a space, say [1] for example. The board's space is manipulated, the player's turn ends, the turn number is incrimented, the next player's name is displayed and the board is reprinted. Game play is the expected alternation between player X and player O. Players are not allowed to capture spaces owned by another player, and invalid inputs cause the input request to loop.



```
Turn number 2
Serano Genomics's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         4 ║ 5 ║ 6
        ═══╬═══╬═══
         X ║ 2 ║ 3

Available moves: 7 8 9 4 5 6 2 3
Please select an available move.
a
Please select an available move.
1
Please select an available move.
```

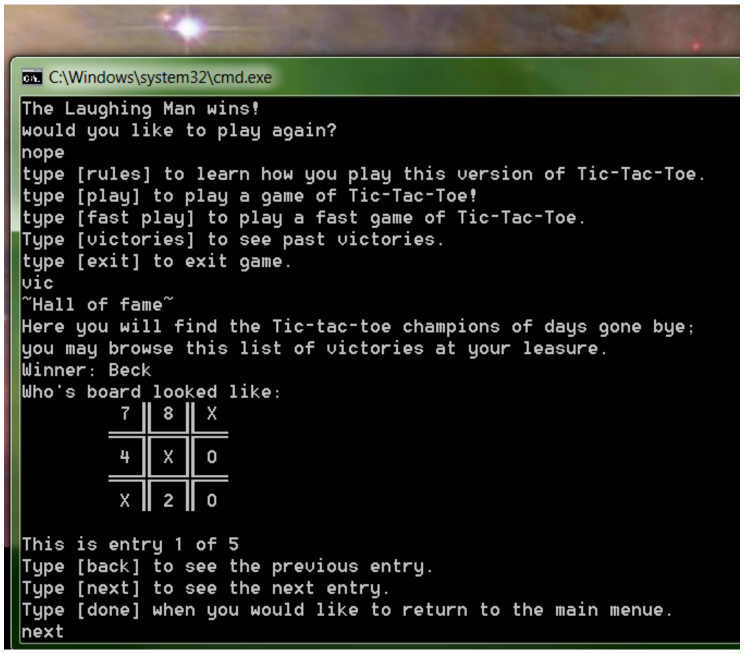
When a player wins in this option, three beeps are issued and the player's name is displayed to the screen while the game quietly saves the player's name and board configureation in victories.dat. At this point the player is offered an oprotunity to loop back to name selection and play again.



```
Turn number 4
Serano Genomics's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         X ║ 5 ║ 6
        ═══╬═══╬═══
         X ║ 2 ║ O

Available moves: 7 8 9 5 6 2
Please select an available move.
6
Turn number 5
The Laughing Man's turn.
         7 ║ 8 ║ 9
        ═══╬═══╬═══
         X ║ 5 ║ O
        ═══╬═══╬═══
         X ║ 2 ║ O

Available moves: 7 8 9 5 2
Please select an available move.
7
The Laughing Man wins!
would you like to play again?
```

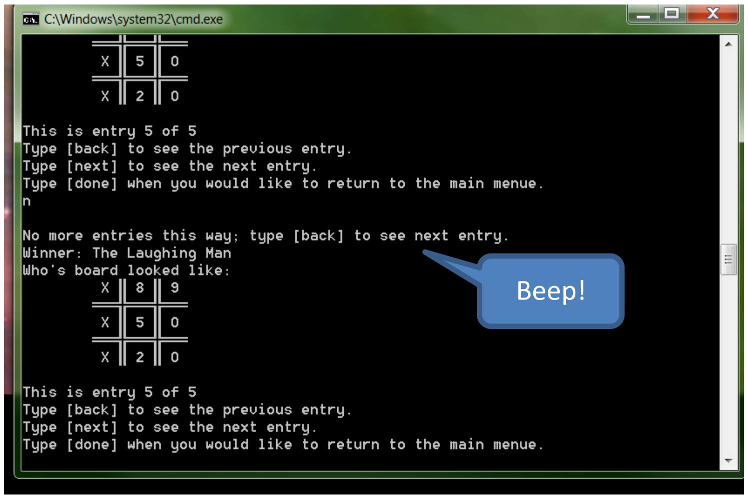
Next let's look at option [victories]:

Here we have a display similar to the player turn display, but we now have browsing options. This menu allows the user to look at elements of a dynamic array of Victory structures read from the victory.dat file. If the user chooses back while at the zeroth element, the system issus a beep and displays a message telling the user that there are no more entries that way, type next. A similar beep and message are displayed when the user tries to look beyond the last element.



```
The Laughing Man wins!
would you like to play again?
nope
type [rules] to learn how you play this version of Tic-Tac-Toe.
type [play] to play a game of Tic-Tac-Toe!
type [fast play] to play a fast game of Tic-Tac-Toe.
Type [victories] to see past victories.
type [exit] to exit game.
vic
~Hall of fame~
Here you will find the Tic-tac-toe champions of days gone bye;
you may browse this list of victories at your leasure.
Winner: Beck
Who's board looked like:
           7 ║ 8 ║ X
          ═══╬═══╬═══
           4 ║ X ║ O
          ═══╬═══╬═══
           X ║ 2 ║ O

This is entry 1 of 5
Type [back] to see the previous entry.
Type [next] to see the next entry.
Type [done] when you would like to return to the main menue.
next
```

For example:

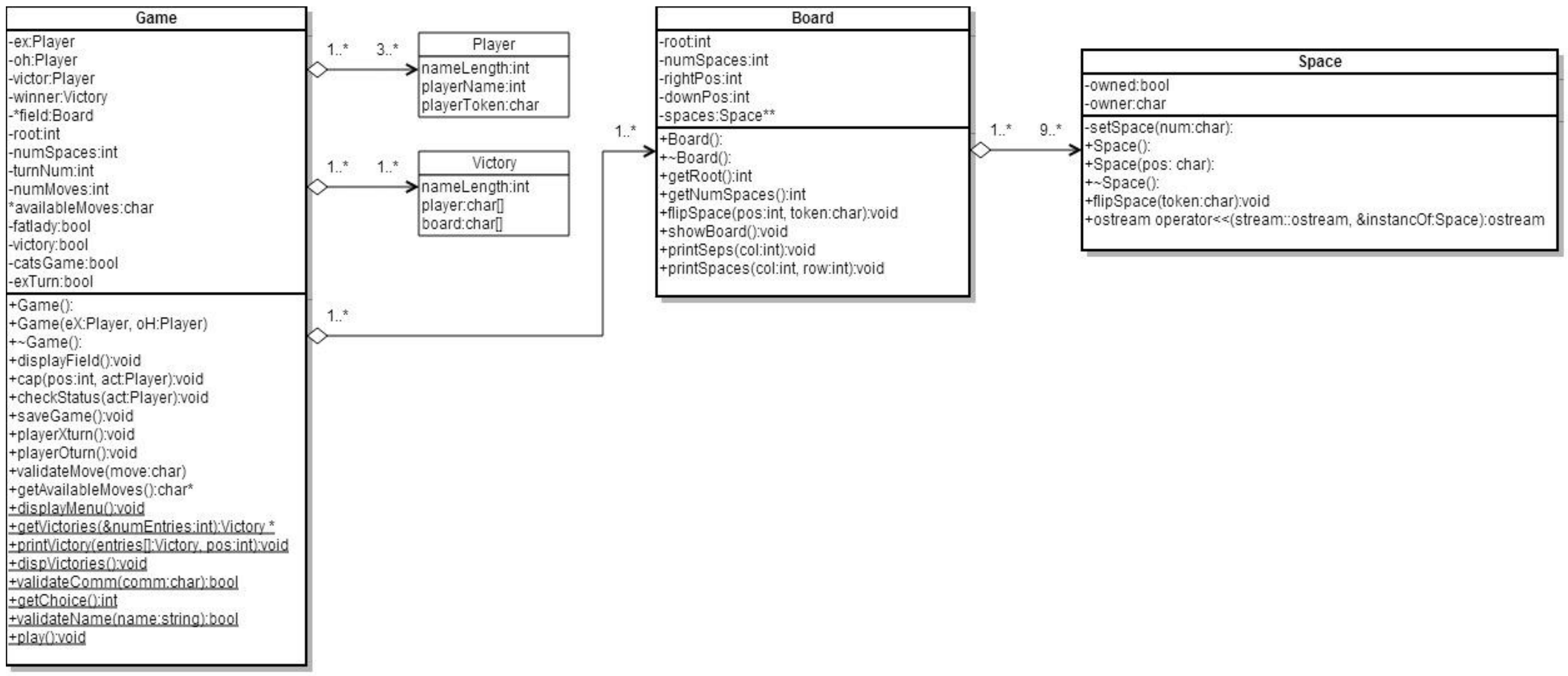


```
           X ║ 5 ║ O
          ═══╬═══╬═══
           X ║ 2 ║ O

This is entry 5 of 5
Type [back] to see the previous entry.
Type [next] to see the next entry.
Type [done] when you would like to return to the main menue.
n

No more entries this way; type [back] to see next entry.
Winner: The Laughing Man
Who's board looked like:
           X ║ 8 ║ 9
          ═══╬═══╬═══
           X ║ 5 ║ O
          ═══╬═══╬═══
           X ║ 2 ║ O

This is entry 5 of 5
Type [back] to see the previous entry.
Type [next] to see the next entry.
Type [done] when you would like to return to the main menue.
```

Truth be told, when developing this algorithm, I jumped right into coding the classes and testing instances in main because I already had a rough idea of what I wanted to do and how I was going to do it. I know the value of documenting my code and so I have also included a UML class diagram and a flow chart which gives and overview of program flow.

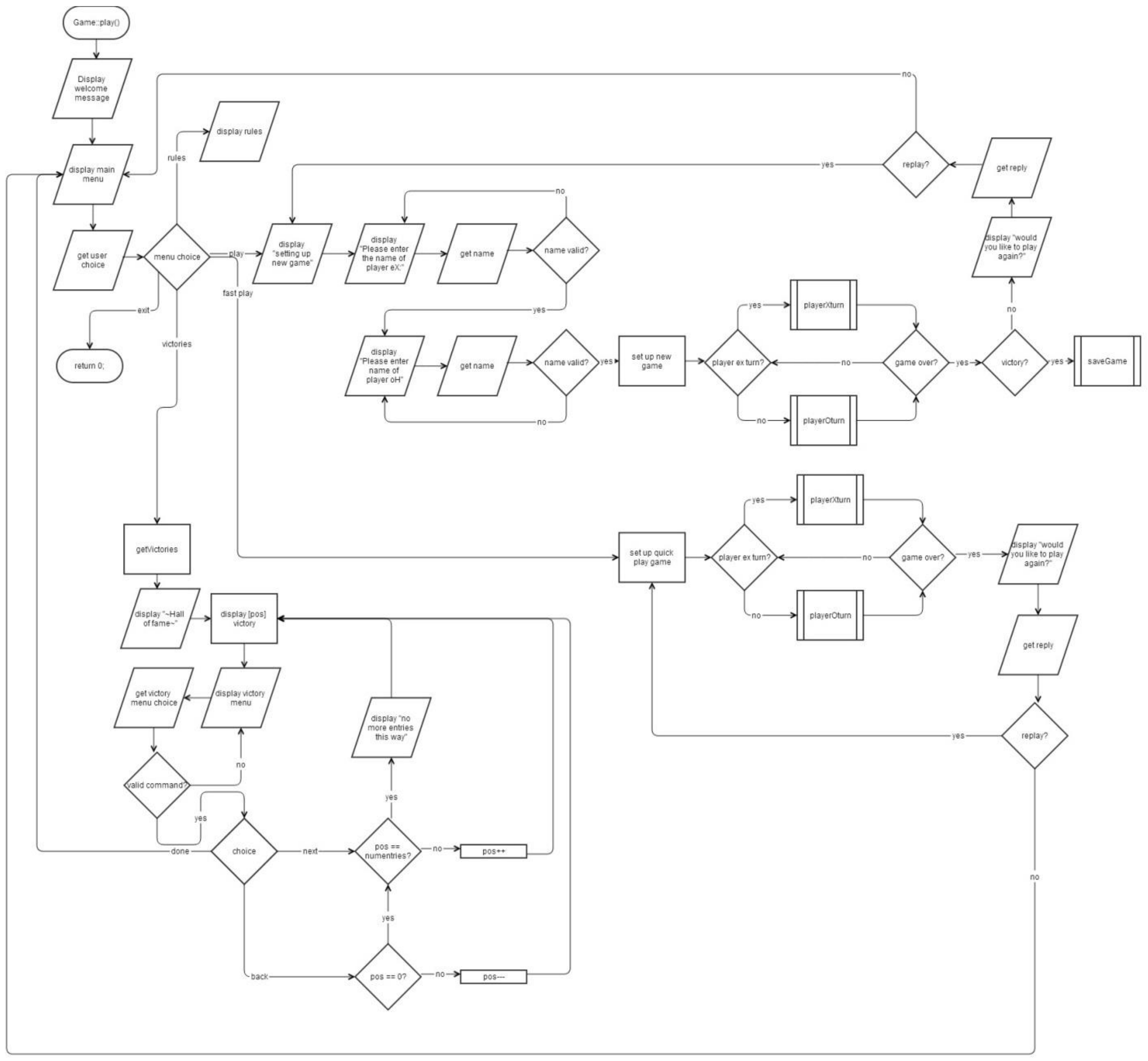
3.2 Visual documentation of code.

Because I know that this is rather hard to see in detail I will also include the jpg with my source code.



This diagram not only details my classes and structures it also shows the relations and cardinality of object instances of those classes. (continues on next page)

Next we see a flow chart which outlines the higher level flow of this program.



3.3 Major variables

Aside from the primative types such as int, char, bool, structures, and standard arays scattered throughout my code; my application also utilizes a dynamically alocated two dimensional array of Space objects stored in a Space pointer field in an instance of the Board class which itself is dynamically created and stored in the data member of a Game class object. (See UML class diagram above). I utilize a dynamic array of Victory objects in the [victories] option accessable from the main menu which is read from the fstream object dataFile.

3.4 Concepts

This program demonstrates many concepts from chapters 1-6 of “Starting out with C++” seventh edition by Tony Gaddis throughout my solution, with particular emphasis on decision making such as if statements, using Boolean values as flags, using logical operators, validating user input, various loops, and menus. A lot of my Board class is set up to process and display a dynamically allocated two dimensional array of Space objects, using the indirection or dot operator to access the Space object’s class methods. The Game object is has a data member which is a pointer to an instance of the Board class, which has a pointer to an array of Space objects, so the Game class is an aggregate of classes, and has several static methods. The Game class’s saveGame() function calls the open function of an fstream object called dataFile with the arguments “victories.dat”, ios::app and ios::bin to store an abstract data type called Victory in a binary file using the fstream object’s write method. Later that same file is opened back up and the getVictories function uses the fstream seekg and tellg methods to return the number of Victory structures in the file, and then creates an allocates memory for an array of Victory structures, before reading data using the fstream object’s read member. Naturally the Victory structure must be reinterpret cast as a char pointer, and it’s size is provided courtesy of the sizeof operator. See UML class diagram and flow chart above.

4 References

I developed most of the code on the fly, however I used “Starting out with C++” Seventh edition by Tony Gaddis as a reference when I wasn’t sure of how to proceed. I used the ascII table found at http://www.asciitable.com/ for the extended character set used in the display and bell sounding of my program. http://www.cs.bsu.edu/homepages/pvg/misc/uml/ provided me with almost everything I needed to know about UML and I used Gliffy to create my UML class diagrams and flow chart.

5 Program

```
//Data file structure.
//Author: John Palacios

#ifndef VICTORY_H
#define VICTORY_H

struct Victory
{
	int nameLength;			//The number of characters used in 
player.
	char player[21];		//The name of the victorious player.
	char board[9];			//The board at time of victory.
};
```

```
#endif /* VICTORY_H */

//Basic information pertinant to the player is stored in this struct.
//Author: John Palacios

#include <string>
using namespace std;

#ifndef PLAYER_H
#define PLAYER_H

struct Player
{
      int nameLength;                              //The number of characters in
playerName.
      string playerName;                           //The Name of player ex,
player oh, or of the winner.
      char playerToken;                      //This is the character used to
mark spaces captured.
};

#endif /* PLAYER */

//Header file for the Space class: an integral part of the Board class.
//Author: John Palacios.

#include <iostream>
using namespace std;

#ifndef SPACE_H
#define SPACE_H

class Space;
      ostream &operator << (ostream &, const Space &);            //Forward
declaration of overloaded stream extraction operator.

class Space
{
private:
      bool owned;                                  //Spaces state in respect to
being owned.
      char owner;                                  //owner's character if owned,
or location if not owned.
      void setSpace(char);                   //Used as initial set up of board.
public:
      Space()
      {owned = false; owner = ' ';}
      //Default constructor.
      Space(char);
      //Test constructor.
      ~Space();
      //destructor.
      void flipSpace(char);
      //Sets the owner and ownership based on alpha numeric character passed
into this function.
      friend ostream &operator << (ostream & stream, const Space &instanceOf)
```

```
	{
		stream << instanceOf.owner;
		return stream;
	}
	//Overloaded ostream sream extraction operator used to display owner's
character to screen.
	/*friend class Board;*/
	//Because the Board contains a dynamic array of this class, Board must
be able to manipulate space objects.
	friend class Game;
	//The Game obj. must manipulate the Board obj, which must manipulate
space objects.
};

#endif /* SPACE_H */

//Implementation of the Space class.
//File: Space.cpp
//Author: John Palacios.

#include "Space.h"
using namespace std;

Space::Space(char pos)
{
	owned = false;
	owner = pos;
}

Space::~Space()
{
}

void Space::setSpace(char num)
{
	owner = num;
}

void Space::flipSpace(char token)
{
	if(owned == false && (!isdigit(token))){
		owner = token;
		owned = true;
	}
	if(owned == false && isdigit(token)){
		setSpace(token);
	}
}

//Header file for the Board class; an integral part of the Game class.
//Author: John Palacios.

#include "Space.h"
//using namespace std;

#ifndef BOARD_H
#define BOARD_H
```

```
class Board
{
private:
      int root;                //The root of the number of spaces, used for 
victory determination.
      int numSpaces;           //The total number of spaces.
      int rightPos;            //Keeps track of the element position when 
printing board.
      int downPos;             //Keeps track of the element position when 
printing board.
protected:
      Space **spaces;          //pointer to two dimensional dynamic array of 
space objects.
public:
      Board();
      //Default constructor.
      ~Board();
      //Destructor.
      int getRoot() const
      { return root;}
      //returns the square root of the number of spaces.
      int getNumSpaces() const
      { return numSpaces;}
      //Tells the game how many spaces there are.
      void flipSpace(int, char);
      //Change the owner and possibly state of the space obj.
      void showBoard();
      //Prints the current familiar hash style playing field to the screen.
      void printSeps(int);
      //prints the horizontal seperation characters.
      void printSpaces(int, int);
      //Alternately prints space obj owner character or pipe char.
      friend class Game;
      //Game has a Board, and Game needs to acces all of board's members.
};

#endif /* BOARD_H */

//Implementation of the Board class.
//File: Board.cpp
//Author: John Palacios.

#include "Board.h"
#include "Space.h"
#include <iostream>
using namespace std;

Board::Board()
{
      int ch = 55;  //Interger base to assigned count character.

      //Default board is 3*3
      root = 3;
      numSpaces = root * root;
      //creat rows of space objects.
      spaces = new Space* [root];
```

```
	for(int i = 0; i < root; i++)
	{
		//creat collumns of space objects.
		spaces[i] = new Space[root];
	}

	for(int i = 0; i < root; i++)
	{
		for(int j = 0; j < root; j++)
		{
			//Set spaces to thier respective positions
			//in the game field based on standard
			//num pad configuration.
			ch -= i * root;
			ch += j;
			spaces[i][j].flipSpace(ch);
			//Reset char base.
			ch = 55;
		}
	}
}

Board::~Board()
{
	for(int i = 0; i < root; i++)
	{
	delete [] spaces[i];
	}

	delete [] spaces;
}

void Board::flipSpace(int pos, char token)
{
	int i;
	int j;
	if(pos >= numSpaces - 2 && pos <= numSpaces)
	{
		i = 0;
		j = pos - 7;
	}
	if(pos >= numSpaces - 5 && pos <= numSpaces - root)
	{
		i = 1;
		j = pos - 4;
	}
	if(pos >= numSpaces - 8 && pos <= numSpaces - (root*2))
	{
		i = 2;
		j = pos - 1;
	}
	spaces[i][j].flipSpace(token);
}

void Board::showBoard()
{
	rightPos = 0;
	downPos = 0;

	for(int row = 0; row < root + 2; row++){
			cout << "\t";
		for(int col = 0; col < root + 2; col++){
			if(row % 2 == 0)
				printSpaces(col, row);
			else{
				printSeps(col);
			}
		}
		if(row % 2 == 1){
			downPos++;
		}
		rightPos = 0;
		cout << endl;
	}
	cout << endl;

}

void Board::printSeps(int col)
{
	char symb[2] = {205, 206};
	if(col % 2 == 0)
		cout << symb[0] << symb[0] << symb[0];
	else
		cout << symb[1];
}

void Board::printSpaces(int col, int row)
{
	char symb = 186;
	if(col % 2 == 0){
		cout << " " << spaces[downPos][rightPos] << " ";
		rightPos++;
	}
	else
		cout << symb;
}



//The Game class; this class orchestrates the constituant Board and Space
objects that make up a game.
//Author: John Palacios

#include "Space.h"
#include "Board.h"
#include "Player.h"
#include "Victory.h"
#include <iostream>
using namespace std;

#ifndef GAME_H
#define GAME_H
```

```
class Game
{
private:
	Player ex;			//Structure storing player x's name and 
token character.
	Player oh;			//Structure storing player o's name and 
token character.
	Player victor;			//The winner's details will be stored 
here.
	Victory winner;			//Data storage struct.
	Board *field;			//Pointer to a two dimensional array of 
space objects.
	int root;			//root value of Board object. used in a 
number of calclulations.
	int numSpaces;			//number of spaces on board, used in a 
number of calculations.
	int turnNum;			//Used to determine a cat's game.
	int numMoves;			//The number of spaces minus spaces owned 
by a player.
	char *availMoves;		//A pointer to a char array used to display the 
available moves.
	bool fatlady;			//End of game flag. True when a player 
wins or cat's game.
	bool victory;			//player wins flag. True when a player 
wins.
	bool catsGame;			//Draw flag. True when no one wins.
	bool exTurn;			//Player alternation flag. True when it 
is player X's turn.
public:
	Game();

	//Default constructor.
	Game(Player, Player);
	//Named player overloaded constructor.
	~Game();

	//Destructor.
	void displayField();
	//Calls Board object's showBoard() f(x).
	void cap(int, Player);
	//Calls Board object's flipSpace() f(x).
	void checkStatus(Player);
	//Reviews Board for victory conditions.
	void saveGame();
	//Called when a player wins; dumps Victory struct into dat file.
	bool getExTurn() const
	{ return exTurn;}
	//Determine's turn.
	void playerXturn();

	//Human Player x's turn protocall.
	void playerOturn();

	//Human Player o's turn protocall.
	bool validateMove(char);
	//Check move selection against available moves.
	char *getAvailableMoves();
	//Audit's field for unowned spaces; returns char array with thos chars.
	static void displayMenu();
	//Main Menu system implementation.
	static Victory *getVictories(int &);
	//creates array of victory structs and fills it with binary data from
dat file.
	static void printVictory(Victory [], int);
	//prints entry from victories array.
	static void dispVictories();
	//Secondary Menu system implementation: browse victories array.
	static bool validateComm(char);
	//validates user command in secondary Menu system.
	static int getChoice();
	//Processes user choice in main menue system.
	static bool validateName(string);
	//Ensures name will fit in victory struct member player name.
	static void play();

	//Runs Main menu, sets up games, runs games, handls victory review.

};

#endif /* GAME_H */

//Implementation of the Game class.
//File: Game.cpp
//Author: John Palacios.

#include "Space.h"
#include "Board.h"
#include "Game.h"
#include "Player.h"
#include "Victory.h"
#include <fstream>
#include <iostream>
using namespace std;

Game::Game()
{
	ex.playerName = "Player X";
	ex.playerToken = 'X';
	oh.playerName = "Player O";
	oh.playerToken = 'O';
	field = new Board;
	root = field->getRoot();
	numSpaces = field->getNumSpaces();
	turnNum = 1;
	numMoves = numSpaces;
	fatlady = false;
	victory = false;
	catsGame = false;
}

Game::Game(Player eX, Player oH)
{
	ex = eX;
	oh = oH;
	field = new Board;
	root = field->getRoot();
	numSpaces = field->getNumSpaces();
	turnNum = 1;
	numMoves = numSpaces;
	fatlady = false;
	victory = false;
	catsGame = false;
}

Game::~Game()
{
	field->~Board();
	delete [] availMoves;
}
void Game::displayField()
{
	field->showBoard();
}

void Game::cap(int pos, Player act)
{
	field->flipSpace(pos, act.playerToken);

	if(turnNum > 2 * root + 1 && victory == false)
	{
		cout << "Looks like no one wins this game.\n";
		fatlady = true;
		catsGame = true;
	}

	else if(turnNum >= root)
	{
		checkStatus(act);
	}
}

void Game::checkStatus(Player act)
{
	int numOpYeX = 0;			//Number of spaces captured by act
in line y = -x.
	int numYeqEx = 0;			//Number of spaces captured by act
in line y = x.
	int numConst = 0;			//Number of spaces captured by act
in line y = some constant.
	int numUndef = 0;			//Number of spaces captured by act
in line x = some constant.

	//Show me what victory looks like!
	//diagonal left to right
	for(int pos = 0; pos < root; pos++){
		if(field->spaces[pos][pos].owner == act.playerToken){
			numOpYeX++;
			if(numOpYeX == root){
				for(int i = 0; i < root; i++){
					cout << static_cast<char>(7);
```

```
                        }
                    cout << act.playerName << " wins!\n";
                    victor = act;
                    victory = true;
                    fatlady = true;
                }
            }
        }

        //diagonal right to left
        for(int row = 0, col = root - 1; row < root; row++, col--){
            if(field->spaces[row][col].owner == act.playerToken){
                numYeqEx++;
                if(numYeqEx == root){
                    for(int i = 0; i < root; i++){
                            cout << static_cast<char>(7);
                        }
                    cout << act.playerName << " wins!\n";
                    victor = act;
                    victory = true;
                    fatlady = true;
                }
            }
        }

        //accross.
        for(int i = 0; i < root; i++){
            numConst = 0;
            for(int j = 0; j < root; j++){
                if(field->spaces[i][j].owner == act.playerToken){
                    numConst++;
                    if(numConst == root){
                        for(int i = 0; i < root; i++){
                            cout << static_cast<char>(7);
                        }
                        cout << act.playerName << " wins!\n";
                        victor = act;
                        victory = true;
                        fatlady = true;
                    }
                }
            }
        }

        //down.
        for(int i = 0; i < root; i++){
            numUndef = 0;
            for(int j = 0; j < root; j++){
                if(field->spaces[j][i].owner == act.playerToken){
                    numUndef++;
                    if(numUndef == root){
                        for(int i = 0; i < root; i++){
                            cout << static_cast<char>(7);
                        }
                        cout << act.playerName << " wins!\n";
                        victor = act;
                        victory = true;
```

```
                                        fatlady = true;
                                }
                        }
                }
        }
}

void Game::saveGame()
{
        fstream dataFile;
        int j = 0;

        winner.nameLength = victor.nameLength;
        for(int i = 0; i < victor.nameLength; i++){
                winner.player[i] = victor.playerName[i];
        }

        for(int row = 0; row < root; row++){
                for(int col = 0; col < root; col++){
                        winner.board[j] = field->spaces[row][col].owner;
                        j++;
                }
        }

        dataFile.open("victories.dat", ios::app | ios::binary);
        dataFile.write(reinterpret_cast<char *>(&winner), sizeof(winner));
        dataFile.close();
}

void Game::displayMenu()
{
        cout << "type [rules] to learn how you play this version of Tic-Tac-
Toe.\n"
                << "type [play] to play a game of Tic-Tac-Toe!\n"
                << "type [fast play] to play a fast game of Tic-Tac-Toe.\n"
                << "Type [victories] to see past victories.\n"
                << "type [exit] to exit game.\n";
}

Victory *Game::getVictories(int &numEntries)
{
        fstream dataFile;
        Victory *entries;

        dataFile.open("victories.dat", ios::in | ios::binary);

        if(dataFile.fail()){
                cout << "Error oppening victories.dat.";
                exit(1);
        }

        dataFile.seekg(0L, ios::end);
        numEntries = dataFile.tellg() / sizeof(Victory);
        dataFile.seekg(0L, ios::beg);

        entries = new Victory [numEntries];
```

```
	for(int i = 0; i < numEntries; i++){
		dataFile.seekg(i * sizeof(Victory), ios::beg);
		dataFile.read(reinterpret_cast<char *>(&entries[i]),
sizeof(Victory));
	}
	dataFile.close();
	return entries;
}

void Game::printVictory(Victory entries[], int pos)
{
	int count = 0;
	char symb[] = {186, 205, 206};

	cout << "Winner: ";
	for(int i = 0; i < entries[pos].nameLength; i++){
	cout << entries[pos].player[i];
	}
	cout << "\nWho's board looked like:\n";
	for(int i = 0; i < 5; i++){
		cout << "\t";
		for(int j = 0; j < 5; j++){
			if(i % 2 == 0){
				if(j % 2 == 0){
					cout << " " << entries[pos].board[count] << "
";

					count++;
				}
				if(j % 2 == 1){
					cout << symb[0];
				}
			}
			if(i % 2 == 1){
				if(j % 2 == 0){
					cout << symb[1] << symb[1] << symb[1];
				}
				if(j % 2 == 1){
					cout << symb[2];
				}
			}

		}
		cout << endl;
	}
}

void Game::dispVictories()
{
	int pos = 0;
	int numEntries;
	string command;
	Victory *victories;
	victories = getVictories(numEntries);
	cout << "~Hall of fame~\n"
		<< "Here you will find the Tic-tac-toe champions of days gone
bye;\n"
		<< "you may browse this list of victories at your leasure.\n";
	do
	{
		do
		{
			printVictory(victories, pos);
			cout << endl;
			cout << "This is entry " << pos + 1 << " of " << numEntries
<< endl;
			cout << "Type [back] to see the previous entry.\n"
			     << "Type [next] to see the next entry.\n"
			     << "Type [done] when you would like to return to the
main menue.\n";

			getline(cin, command);
		} while(validateComm(tolower(command[0])) == false);
		if(tolower(command[0]) == 'b' && pos == 0){
			cout << static_cast<char>(7) << "\nNo more entries this
way; type [next] to see next entry.\n";
		}
		if(tolower(command[0]) == 'b' && pos > 0){
			pos--;
		}
		if(tolower(command[0]) == 'n' && pos == numEntries - 1){
			cout << static_cast<char>(7) << "\nNo more entries this
way; type [back] to see next entry.\n";
		}
		if(tolower(command[0]) == 'n' && pos < numEntries - 1){
			pos++;
		}
		if(tolower(command[0]) == 'd'){
			cout << "You typed [" << command << "]. Exiting hall of
fame...\n";
		}
	}while(tolower(command[0]) != 'd');
	delete [] victories;
}

bool Game::validateComm(char comm)
{
	bool goodComm = false;

	if(tolower(comm) == 'b'){
		goodComm = true;
	}
	if(tolower(comm) == 'n'){
		goodComm = true;
	}
	if(tolower(comm) == 'd'){
		goodComm = true;
	}

	return goodComm;
}

int Game::getChoice()
{
	string choice;
	int num = 0;
	getline(cin, choice);

	if(tolower(choice[0]) == 'r'){
		num = 1;
	}

	if(tolower(choice[0]) == 'p'){
		num = 2;
	}

	if(tolower(choice[0]) == 'f'){
		num = 3;
	}

	if(tolower(choice[0]) == 'v'){
		num = 4;
	}

	if(tolower(choice[0]) == 'e'){
		num = 5;
	}

	else if(num == 0){
		cout << "[" << choice << "] was not an option, please try 
again.\n";
	}

	return num;
}

bool Game::validateName(string name)
{
	bool goodName = true;		//input validity flag
	if(name.size() > 21){
		goodName = false;
	}
	return goodName;
}

void Game::playerXturn()
{
	string move;

	cout << "Turn number " << turnNum << endl;
	cout << ex.playerName << "'s turn.\n";
	availMoves = getAvailableMoves();
	displayField();
	cout << "Available moves: ";
	for(int i = 0; i < numMoves; i++){
		cout << availMoves[i] << " ";
	}
	cout << endl;
	do{
		cout << "Please select an available move.\n";
		getline(cin, move);
	} while (validateMove(move[0]) == false);
	cap(move[0] - 48, ex);
	turnNum++;
	exTurn = false;
}

void Game::playerOturn()
{
	string move;

	cout << "Turn number " << turnNum << endl;
	cout << oh.playerName << "'s turn.\n";
	availMoves = getAvailableMoves();
	displayField();
	cout << "Available moves: ";
	for(int i = 0; i < numMoves; i++){
		cout << availMoves[i] << " ";
	}
	cout << endl;
	do{
		cout << "Please select an available move.\n";
		getline(cin, move);
	} while (validateMove(move[0]) == false);
	cap(move[0] - 48, oh);
	turnNum++;
	exTurn = true;
}

bool Game::validateMove(char move)
{
	bool goodMove = false;

	for(int i = 0; i < numMoves; i++){
		if(move == availMoves[i]){
			goodMove = true;
		}
	}
	return goodMove;
}

char *Game::getAvailableMoves()
{
	int moves = 0;
	char *availableMoves;
	availableMoves = new char[numMoves];

	for(int row = 0; row < root; row++){
		for(int col = 0; col < root; col++){
			if(field->spaces[row][col].owned == false){
				availableMoves[moves] = field-
>spaces[row][col].owner;
				moves++;
			}
		}
	}
	numMoves = moves;
	return availableMoves;
}
```

```
void Game::play()
{
      bool replay = false;
      int choice;
      string move;
      string name;
      string set;
      Player eX;
      Player oH;
      char symb[3] = {186, 205, 206};

      cout << "Welcome to Tic-Tac-Toe! The clasic strategy edition.\n"
            << "The object of the game is to get three of your tokens in\n"
            << "a row while preventing your opponent from doing the
same!\n\n";

      do
      {
            do
            {
                  displayMenu();
                  choice = getChoice();
            }while (choice == 0);

            switch (choice)
            {
            case 1:
                  cout << "Turn number 1\n"
                        << "Player X's turn\n"
                        << "\t 7 " << symb[0] << " 8 "<< symb[0] << " 9
type one of these numbers\n"
                        << "\t" << symb[1] << symb[1] << symb[1]<< symb[2] <<
symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] << symb[1] <<
endl
                        << "\t 4 " << symb[0] << " 5 "<< symb[0] << " 6
<-to capture that space!\n"
                        << "\t" << symb[1] << symb[1] << symb[1]<< symb[2] <<
symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] << symb[1] <<
endl
                        << "\t 1 " << symb[0] << " 2 "<< symb[0] << " 3
\n"
                        << "Available Moves: 7 8 9 4 5 6 1 2 3\n"
                        << "Please select an available move.\n"
                        << "[type a number, then press enter]\n\n"
                        << "you may choose any space displayed that\n"
                        << "does not belong to another player. here's\n"
                        << "an example of such a condition:\n"
                        << "Turn number 6\n"
                        << "Player X's turn\n"
                        << "\t X " << symb[0] << " 8 "<< symb[0] << " O
type one of these numbers\n"
                        << "\t" << symb[1] << symb[1] << symb[1]<< symb[2] <<
symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] << symb[1] <<
endl
                        << "\t 4 " << symb[0] << " X "<< symb[0] << " 6
<-to capture that space!\n"
```

```
                        << "\t" << symb[1] << symb[1] << symb[1]<< symb[2] <<
symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] << symb[1] <<
endl
                        << "\t 1 " << symb[0] << " 2 "<< symb[0] << " O
\n"
                        << "Available Moves: 8 4 6 1 2\n"
                        << "Please select an available move.\n"
                        << "[type a number, then press enter]\n\n"
                        << "The spaces 3 and 9 have been captured by\n"
                        << "player Oh; therefore player eX may not take\n"
                        << "Those spaces, nor spaces marked with thier own\n"
                        << "token \"X\" this game.\n\n"
                        << "Once a player has achieved victory or the game\n"
                        << "is a cat's game, you will be asked if you
would\n"
                        << "like to play again.\n"
                        << "If you manage to win, your name and the board\n"
                        << "Configureation will be saved and you may use
it\n"
                        << "to brag to your friends later.\n"
                        << "*Tip: turn on your speakers.\n\n";
                  break;

            case 2:
                  do
                  {
                        cout << "Setting up new game...\n";
                        do
                        {
                              cout << "Please enter the name of player eX:\n"
                                    << "(up to twenty characters long)\n";
                              getline(cin, name);
                        }while (validateName(name) == false);
                        eX.nameLength = static_cast<int>(name.length());
                        eX.playerName = name.c_str();
                        name.clear();

                        do
                        {
                              cout << "Please enter the name of player oH:\n"
                                    << "(up to twenty characters long)\n";
                              getline(cin, name);
                        }while (validateName(name) == false);
                        oH.nameLength = static_cast<int>(name.length());;
                        oH.playerName = name.c_str();
                        name.clear();

                        eX.playerToken = 'X';
                        oH.playerToken = 'O';
                        Game *newGame;
                        newGame = new Game(eX, oH);
                        cout << static_cast<char>(7);
                        do
                        {
                              if(newGame->getExTurn()){
                                    newGame->playerXturn();
                              }
```

```
                                else
                                        newGame->playerOturn();

                        } while (newGame->fatlady == false);

                        if(newGame->victory){
                                newGame->saveGame();
                        }

                        move.clear();
                        cout << "would you like to play again?\n";
                        getline(cin, move);
                        if(tolower(move[0]) == 'y'){
                                newGame->~Game();
                                replay = true;
                        }
                        else{
                                delete newGame;
                                replay = false;
                        }

                } while (replay == true);
                break;

        case 3:
                do
                {
                        Game *newGame;
                        newGame = new Game();
                        cout << static_cast<char>(7);
                        do
                        {
                                if(newGame->getExTurn()){
                                        newGame->playerXturn();
                                }
                                else
                                        newGame->playerOturn();

                        } while (newGame->fatlady == false);

                        move.clear();
                        cout << "would you like to play again?\n";
                        getline(cin, move);
                        if(tolower(move[0]) == 'y'){
                                newGame->~Game();
                                replay = true;
                        }
                        else{
                                replay = false;
                                newGame->~Game();
                        }
                } while (replay == true);
                break;

        case 4:
                dispVictories();
                break;
```

```
            case 5:
                  cout << "Good bye.\n";
                  break;
            }
      }while(choice != 5);
}
```