

Final Project

Title

Tic-Tac-Toe V2.0

“The classic strategy game of old”

Class

CSC 17A, Section 47975

Author:

John Palacios

Outline of contents

1	Introduction		3
2	Summary		3
3	Description		7
4	References		13
5	Source code		13

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, for instance; it was found only recently (2002) that there are essentially 26,830 unique games.

I chose Tic-Tac-Toe because this game presents some inherent challenges, and offers opportunities for greater challenges; especially in programming an AI. 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

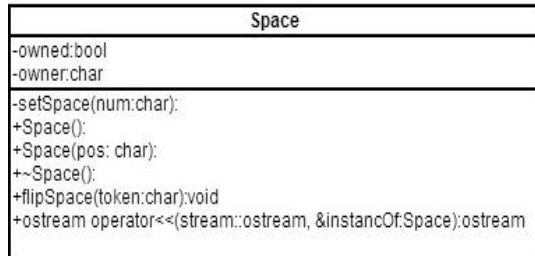
Number of variables	53+	Number of ADTs	3
Single for loops	25	Nested for loops	9
If or if/else statements	100+	Do-while loops	11
Data members	53	Methods	67
Static members	8	Classes	10
Menus	2	files	17
Lines of code	1397	Hours spent in development	~36+

Two classes are only thrown in the event of an error for exception handling. The line count does not include comments or curly braces on lines by themselves. I added 1001 lines of code to my first project.

My solution is best described as an aggregate of classes working together to perform a fairly simple task. It is not elegant code, but it does showcase many object oriented concepts.

Classes in more detail

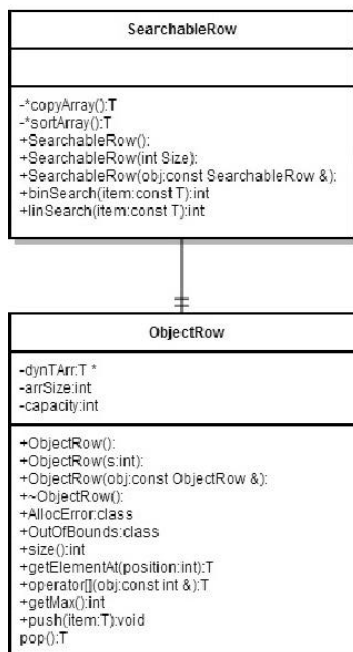
2.1 The Space class



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

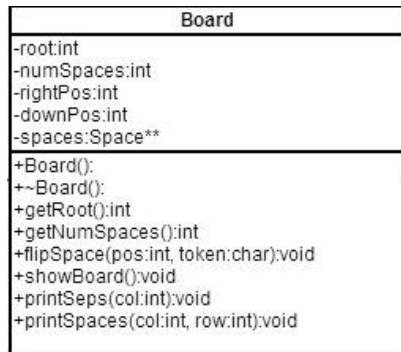
the board or capture that space. This class seemed too simple to me, so I also overloaded the ostream class's stream extraction operator to specifically return the owner of the space's character (instead of writing a simple "getOwner" method). Overloading the ostream operator was a bigger challenge than 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 SearchableRow Class



This class is a template class; it works with any data type. I use an array of these objects to keep track of my space objects; this replaces the two dimensional array of space objects included in my first project. In addition to providing my AI's with a fast way of determining if 'X' is contained in a given row; this class also handles internal memory allocation and array bounds checking using two exception classes.

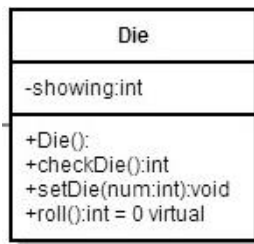
2.3 The Board class



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

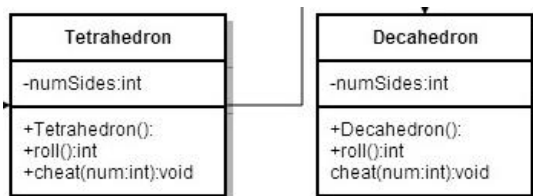
facilitate control of the dynamically allocated Board object.

2.4 The Die class



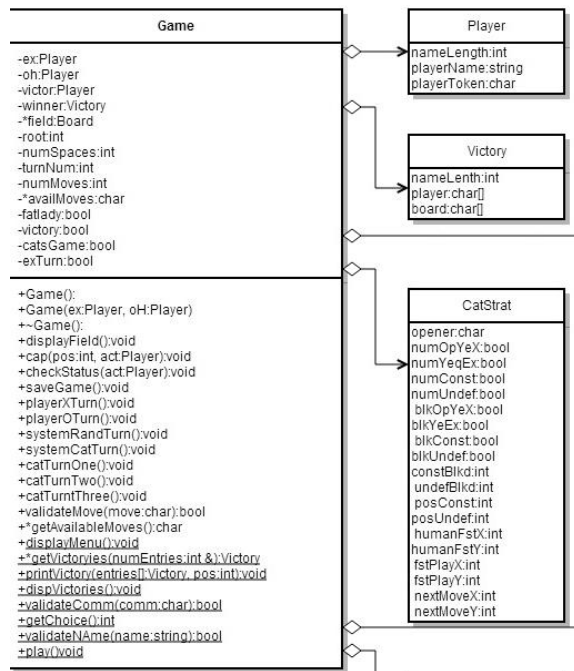
This class is a pure abstract base class from which I inherit the Tetrahedron and the Dodecahedron classes my AI uses to make decisions when more than one option is suitable.

2.5 The Tetrahedron and Decahedron Class



This class is a die who represents a four sided die or a ten sided die one of my AI uses when it needs to make a decision and more than one option is optimal or when a little chance will add a sense of variety to game play.

2.6 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 CatStrat 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 (or impending victory in the case of my AI), saveGame method to save the Victory structure to a binary file, a getExTurn method which returns the exTurn Boolean value, two

human player turn methods (one for each player), two AI turn methods (one for the random choice AI, the other for Cat who attempts to win or force a cat's game), a 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.

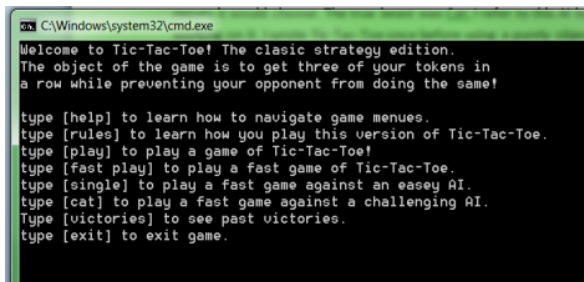
Final thoughts on this project

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 three 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. The third way, and possibly the greatest bit of complexity, were my two AI. Random choice AI was simpler than I thought it would be, and surprised me with some of the moves it would choose. The true beast was Cat. I refer to this AI by name because I spent so much time working on it. I wrote Tic-Tac-Toe once before using a purely object oriented interpreted language called ruby, but I was able to implement a simple array matching algorithm in that language in order to facilitate an AI algorithm. C++ is much more challenging. I am sorry to say that I cannot find my old code, so I had no choice but to develop my own AI and use the Array manipulation techniques I learned from C++ to implement Cat. It took a long time and A LOT of debugging, the code is not pretty, I know I that there were better ways to solve this problem, but I learned a lot from the experience.

3 Description

3.1 General description of user experience.



```

C:\Windows\system32\cmd.exe
Welcome to Tic-Tac-Toe! The classic 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 [help] to learn how to navigate game menus.
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 [single] to play a fast game against an easy AI.
type [cat] to play a fast game against a challenging AI.
type [victories] to see past victories.
type [exit] to exit game.
  
```

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.

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

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



```

C:\Windows\system32\cmd.exe
type [cat] to play a fast game against a challenging AI.
type [victories] to see past victories.
type [exit] to exit game.
ru
Turn number 1
Player X's turn
  7 8 9
  4 5 6
  1 2 3
type one of these numbers
<-to capture that space!
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 5
Player X's turn
  X 8 0
  4 X 5
  1 2 0
type one of these numbers
<-to capture that space!
Available Moves: 9 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 x may not take
these spaces. nor spaces marked with their 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
Configuration will be saved and you may use it
to brag to your friends later.
*Tip: turn on your speakers.
type [help] to learn how to navigate game menus.
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 [single] to play a fast game against an easy AI.
type [cat] to play a fast game against a challenging AI.
type [victories] to see past victories.
type [exit] to exit game.
  
```

```

C:\Windows\system32\cmd.exe
Turn number 1
Player X'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.
[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
  --+--+
  4 | X | 6
  --+--+
  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
Configuration 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.
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.

```

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 four options available for game play: [play], [fast play], [single], and [cat]. Any one of these options allows the user to play, but [fast play] is just a generic tic-tac-toe game.

```

C:\Windows\system32\cmd.exe
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.

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 incremented, the next player's name is displayed and the board is reprinted. Game play is the

```

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.

```



```

C:\Windows\system32\cmd.exe
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.

```

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.

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 configuration in victories.dat. At this point the player is offered an opportunity to loop back to name selection and play again.

```

C:\Windows\system32\cmd.exe
Turn number 4
Serano Genomics's turn.
  7 | 8 | 9
  --+--+
  X | 5 | 6
  --+--+
  X | 2 | 0

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 | 0
  --+--+
  X | 2 | 0

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 issues 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.

For example:

```

C:\Windows\system32\cmd.exe
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 leisure.
Winner: Beck
Who's board looked like:
  7 | 8 | X
  --+--+
  4 | X | 0
  --+--+
  X | 2 | 0

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 menu.
next

```

```

C:\Windows\system32\cmd.exe
  X | 5 | 0
  --+--+
  X | 2 | 0

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 menu.
n
No more entries this way; type [back] to see the previous entry.
Winner: The Laughing Man
Who's board looked like:
  X | 8 | 9
  --+--+
  X | 5 | 0
  --+--+
  X | 2 | 0

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 menu.

```

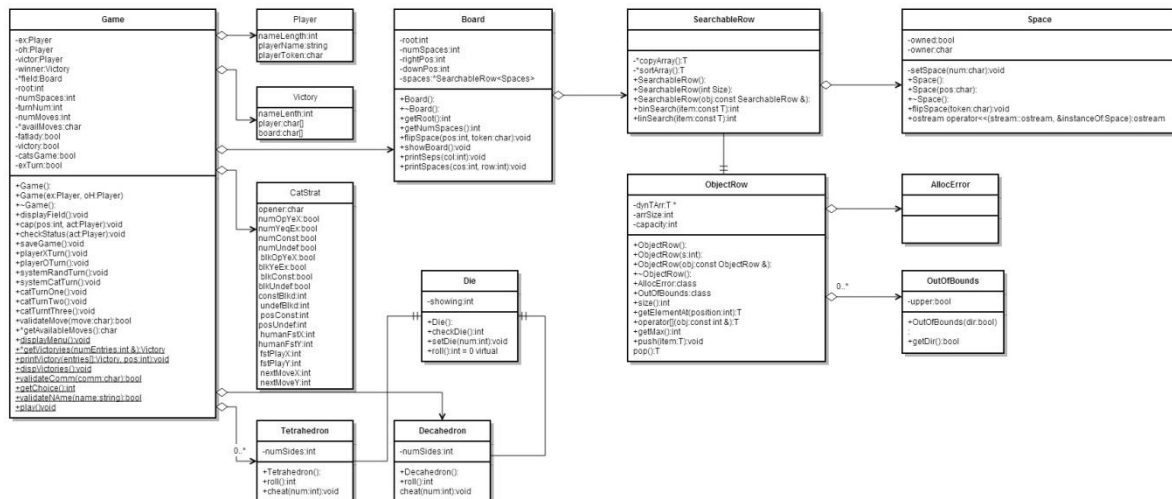
Beep!

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 did a little work on a white board to figure out some of my solution.

I know the value of documenting my code and so I have also included a UML class diagram and a flow chart which gives an overview of program flow.

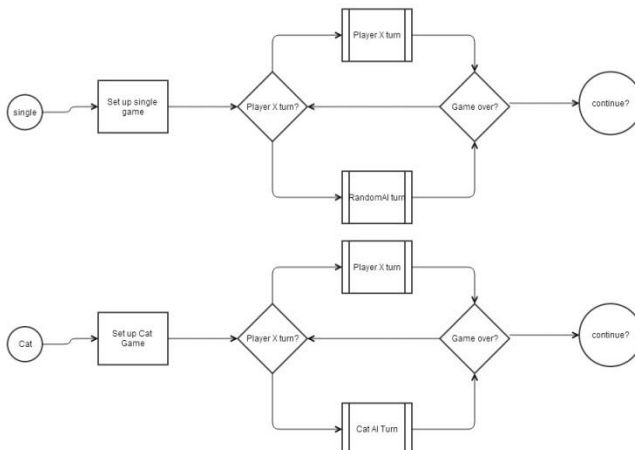
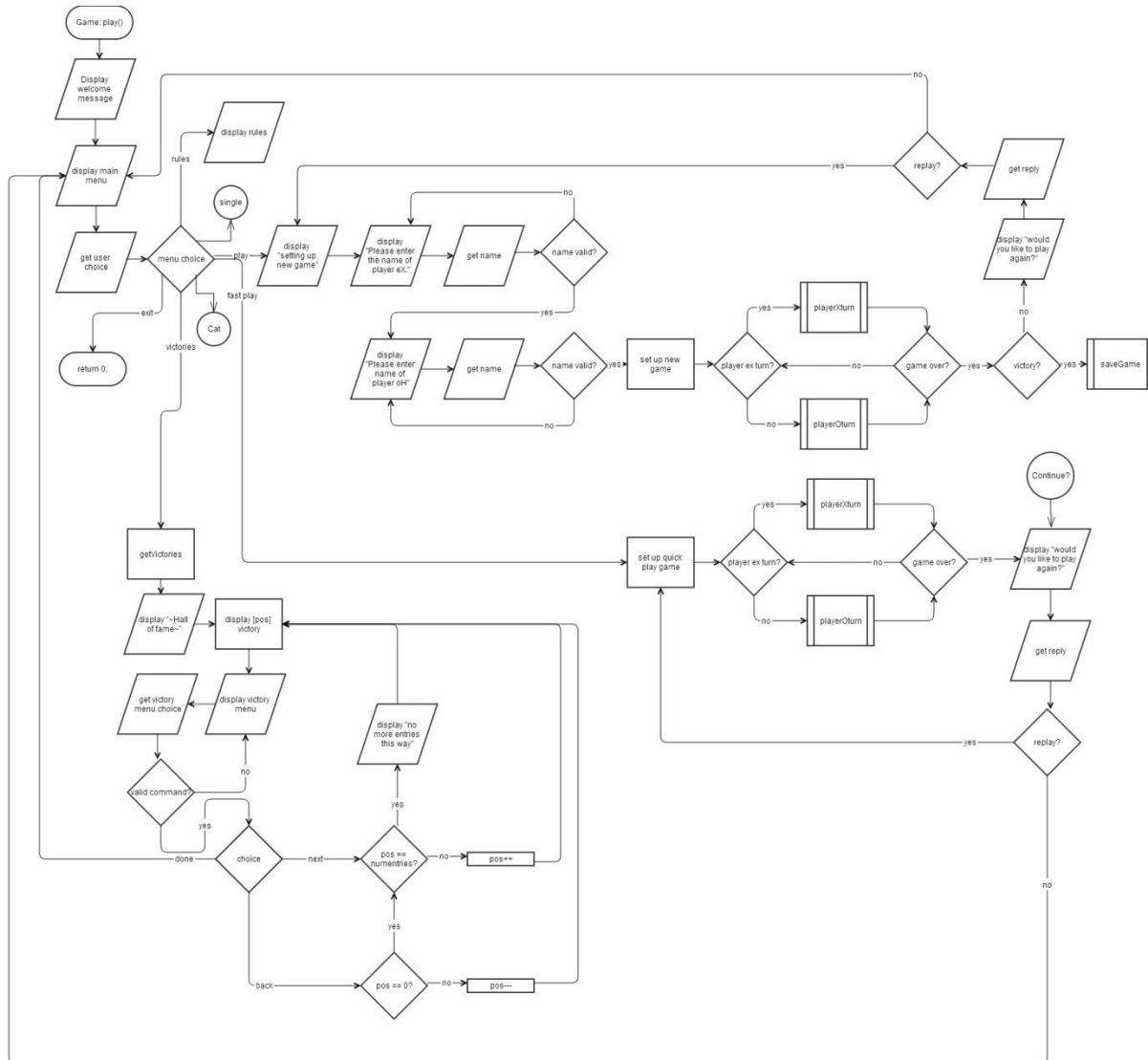
3.2 Visual documentation of code.

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 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

Many of the major variables utilized in my solution are instances of classes. The entirety of my project is an aggregate of classes interacting with each other.

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 array of searchable Row objects which is a template class that works with its dynamically allocated array of Space objects, using the indirection or dot operator to access the Space and Row 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 SearchableRow objects which itself has 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. The largest part of the Game class are the two system turn methods and my two AI.

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, I used Gliffy to create my UML class diagrams and flow chart, I learned everything I know about naughts and crosses from Wikipedia at <http://en.wikipedia.org/wiki/Tic-tac-toe>, I got the inspiration for having my AI use a die object from <http://pine.fm/LearnToProgram/?Chapter=09> many years ago.

5 Program

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

```
#include "Space.h"  
#include "SearchableRow.h"
```

```

#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:
    SearchableRow<Space> *spaces; //Spaces now stored in a dynamic array of
    SearchableRow<Space>
                                //Template objects.
    //Space **spaces;          //pointer to two dimensional dynamic array of
    space objects.
public:
    Board();
    //Default constructor.
    Board(int);
    //Constructor for different sized boards.
    ~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 */

//This struct will store information about the player's moves so that Cat may
move accordingly.
//File: CatStrat.h
//Author: John Palacios

//#include <string>
using namespace std;

#ifndef CATSTRAT_H

```

```
#define CATSTRAT_H

struct CatStrat
{
    char opener;
    bool numOpYeX;
    bool numYeqEx;
    bool numConst;
    bool numUndef;
    bool blkOpYeX;
    bool blkYeEx;
    bool blkConst;
    bool blkUndef;
    int constBlkd;
    int undefBlkd;
    int posConst;
    int posUndef;
    int humanFstX;
    int humanFstY;
    int fstPlayX;
    int fstPlayY;
    int nextMoveX;
    int nextMoveY;
};

#endif /* CATSTRAT_H */

//class derived from the abstract base class Die. This die has 10 sides.
//File: Decahedron.h
//Author: John Palacios

#ifndef DECAHEDRON_H
#define DECAHEDRON_H

#include "Die.h"

class Decahedron : public Die
{
private:
    int numSides;
public:
    Decahedron() : Die()
    { numSides = 10; }
    //default constructor.
    int roll();
    //f(x) from abstract base class which must be defined.
    void cheat(int);
    //Standard sheater f(x).
};

#endif /* DECAHEDRON_H */
```

```
//This abstract base class represents the concept of a die; a real world
random number generator.
//File: die.h
//Author: John Palacios
```

```
#include <ctime>
#include <cstdlib>
using namespace std;

#ifndef DIE_H
#define DIE_H

class Die
{
private:
    int showing;
public:
    Die()
    { showing = 1;}
    //Default constructor.
    int checkDie()
    { return showing; }
    //Check to see which number is showing.
    void setDie(int num)
    {showing = num; }
    //set side showing to the number given.
    virtual int roll() = 0;
    //Generates a random number and sets showing.
};

#endif /* DIE_H */
```

```
//The Game class; this class orchestrates the constituent 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 "Tetrahedron.h"
#include "Decahedron.h"
#include "CatStrat.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
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.
    CatStrat compStrat;      //Structure storing human player's moves
and stratigic information.
    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 calculations.
    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.
    void systemRandTurn();
    //Random AI player's turn.
    void systemCatTurn();
    //AI actively seeks a cats's game.

```

```

    void catTurnOne();
    //AI first turn.
    void catTurnTwo();
    //AI second turn.
    void catTurnThree();
    //AI Third turn.
    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 menu 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 */

// ObjectRow class template
#ifndef OBJECTROW_H
#define OBJECTROW_H

//Since this object borrows memory from the free store; I should make sure it
does so properly.
//This header contains the bad_alloc error class thrown by the new operator.
#include <new>
using namespace std;

template <class T>
class ObjectRow
{
private:
    T *dynTArr;           //points to allocated array of T type
    int arrSize;          //Number of used elements in the array
    int capacity;         //Total Maximum number of elements available.

public:
    //Default constructor simply creates an instance of this class,
    //no objects of type T are created.
    ObjectRow()

```

```

{ dynTArr = 0; arrSize = 0, capacity = 0;}
//Creates instance of this class along with s type T objects.
ObjectRow(int);
//Your standard copy constructor.
ObjectRow(const ObjectRow &);
//Your standard destructor.
~ObjectRow();
//Memory allocation exception class.
class AllocError
{ };
//Array bounds checking exception class.
class OutOfBounds
{
private:
    bool upper;          //If out of bounds above array size, true.
public:
    OutOfBounds(bool dir)
    { upper = dir; }
    bool getDir()
    { return upper; }
};
int size() const
{ return arrSize; }
//Returns value of element at position.
T getElementAt(int position);
// Overloaded [] operator.
T &operator[](const int &);
//Returns the actual number of available elements.
int getMax() const
{ return capacity; }
//adds object to end of array, allocating more memory
//if necessary.
void push(T);
//Decrements the size of the array, returning
//The last element.
T pop();
};

template <class T>
ObjectRow<T>::ObjectRow(int s)
{
    arrSize = s;
    capacity = s;
    //Attempt to allocate mem.
    try
    {
        dynTArr = new T [s];
    }
    catch (bad_alloc)
    {
        //If fail, tell the program.
        throw AllocError();
    }
}

template <class T>
ObjectRow<T>::~~ObjectRow()

```

```
{
    //If the object is initialized without argument, it has size zero; and
    no T objects pointed to by dynTArr.
    if (arrSize > 0){
        delete [] dynTArr;
        dynTArr = 0;
    }
}

template <class T>
ObjectRow<T>::ObjectRow(const ObjectRow &obj)
{
    arrSize = obj.arrSize;

    dynTArr = new T [arrSize];
    if (dynTArr == 0)
        throw AllocError();

    //Copy the elements of obj's array.
    for(int count = 0; count < arrSize; count++)
        *(dynTArr + count) = *(obj.dynTArr + count);
}

template <class T>
T ObjectRow<T>::getElementAt(int sub)
{
    //Prevent accessing memory outside of the array
    //Before the system complains and shuts down the
    //Program.
    if (sub < 0)
        throw OutOfBounds(false);
    if(sub >= arrSize){
        //This gives the program better understanding
        //Of what is happening and allows for finer
        //handling of such problems.
        throw OutOfBounds(true);
    }
    return dynTArr[sub];
}

template <class T>
T &ObjectRow<T>::operator[] (const int &sub)
{
    //Prevent accessing memory outside of the array
    //Before the system complains and shuts down the
    //Program.
    if (sub < 0)
        throw OutOfBounds(false);
    if(sub >= arrSize){
        //This gives the program better understanding
        //Of what is happening and allows for finer
        //handling of such problems.
        throw OutOfBounds(true);
    }
    return dynTArr[sub];
}
```

```

template <class T>
void ObjectRow<T>::push(T val)
{
    //The object may be initialized with zero
    //elements, and have elements added at runtime.
    //Since there are 0 elements at this time, no
    //Need to copy dynTArr objects.
    if(arrSize == 0){
        dynTArr = new T;
        dynTArr[0] = val;
        arrSize = 1;
        capacity = 1;
    }
    else if(arrSize++ == capacity){
        capacity *= 2;
        T *temp = new T [capacity];
        for(int i = 0; i < arrSize - 1; i++){
            temp[i] = dynTArr[i];
        }
        temp[arrSize - 1] = val;
        if(capacity == 2){
            delete dynTArr;
        }
        else{
            delete [] dynTArr;
        }
        dynTArr = temp;
    }
    else{
        dynTArr[arrSize - 1] = val;
    }
}

template <class T>
T ObjectRow<T>::pop()
{
    //When objects are added to dynTArr, they will overwrite these
    //unindexed objects.
    arrSize--;
    return dynTArr[arrSize - 1];
}

#endif /* OBJECTROW_H */

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

#include <string>
using namespace std;

#ifdef PLAYER_H
#define PLAYER_H

struct Player
{

```

```

        int nameLength;                //The number of characters in
playerName.                            //The Name of player ex,
        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 */

//The searchable functionality may be useful for making AI moves...

#ifndef SEARCHABLEROW_H
#define SEARCHABLEROW_H
#include "ObjectRow.h"

template <class T>
class SearchableRow : public ObjectRow<T>
{
private:
    //creates a copy of ObjectRow::dynTArr, returning a pointer to that
    //array.
    T *copyArray();
    //Sorts array to facilitate binary search.
    T *sortArray();
public:
    //Just using base class constructor.
    SearchableRow() : ObjectRow<T>()
    { }
    //Just passing the size to the base class constructor.
    SearchableRow(int size) : ObjectRow<T>(size)
    { }
    //Copy constructor
    SearchableRow(const SearchableRow &);
    //find item using binary search, returns -1 when unsuccessful.
    //Does not rearrange elements, not useful for finding actual index of
object.
    int binSearch(const T) const;
    //Find item using linear search, returns -1 when unsuccessful.
    //Does not rearrange elements.
    int linSearch(const T);
};

template <class T>
SearchableRow<T>::SearchableRow(const SearchableRow &obj) :
ObjectRow<T>(obj.size())
{
    for(int count = 0; count < this->size(); count++)
        this->operator[](count) = obj[count];
}

template <class T>
int SearchableRow<T>::binSearch(const T item) const
{
    T *srtdArr = sortArray();

```

```

    int first = 0,                                //First element
        last = this->size() - 1,                  //Last element
        middle,                                  //Mid point of search
        position = -1;                           //Position of search value
    bool found = false;                          //Flag

    while (!found && first <= last)
    {
        middle = (first + last) / 2;
        if (srtdArr[middle] == item)
        {
            found = true;
            position = middle;
        }
        //Item value > than middle value?
        else if (srtdArr[middle] > item)
            last = middle - 1;
        else
            first = middle + 1;
    }
    return position;
}

template <class T>
T *SearchableRow<T>::copyArray()
{
    T *copy;
    copy = new T [this->size()];

    for(int i = 0; i < this->size(); i++){
        *(copy + i) = this->operator [] (i);
    }

    return copy;
}

template <class T>
T *SearchableRow<T>::sortArray()
{
    T *srtdArr = this->copyArray();
    int size = this->size();
    int startScan;                                //This is the current index to place
smallest value
    int minIndex;                                //thie is the index of the current
smallest value
    int minValue;                                //this is the current smallest value

    for (startScan = 0; startScan < (size - 1); startScan++)
    {
        minIndex = startScan;
        minValue = srtdArr[startScan];

        for (int i = startScan + 1; i < size; i++)
        {
            if (srtdArr[i] < minValue)
            {
                minValue = srtdArr[i];
            }
        }
    }
}

```

```

        minIndex = i;
    }
    }
    srtddArr[minIndex] = srtddArr[startScan];
    srtddArr[startScan] = minValue;
}
return srtddArr;
}

template <class T>
int SearchableRow<T>::linSearch(const T item)
{
    for (int i = 0; i < this->size(); i++){
        if (getElementAt(i) == item){
            return i;
        }
    }
    return -1;
}
#endif /* SEARCHABLEROW_H */

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

#include <iostream>
using namespace std;

#ifdef 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 stream extraction operator used to display owner's
    character to screen.
    bool Space::operator == (const Space &);
    /*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 */

//class derived from the abstract base class Die. This die has 4 sides.
//File: Tetrahedron.h
//Author: John Palacios

#ifndef TETRAHEDRON_H
#define TETRAHEDRON_H

#include "Die.h"

class Tetrahedron : public Die
{
private:
    int numSides;
public:
    Tetrahedron() : Die()
    { numSides = 4; }
    //default constructor.
    int roll();
    //f(x) from abstract base class which must be defined.
    void cheat(int);
    //Standard sheater f(x).
};

#endif /* TETRAHEDRON_H */

//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 */

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

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

Board::Board()
{
    bool retry = false;
    int ch = 55; //Integer base to assigned count character.
    root = 3;
    numSpaces = root * root;
    //I could use a two dimensional array of space objects;
    //But the SearchableRow takes care of some tasks
    //Related to dynamic allocation.
    spaces = new SearchableRow<Space> [root];
    do{
        //This process may be repeated should the system cause
        //my vector object to throw a memory allocation error
        //Class for some temporary reason.
        try{
            for(int i = 0; i < root; i++){
                for(int j = 0; j < root; j++){
                    //Now to set each space to designate it's
                    //Key to be pressed to capture it.
                    ch -= i * root;
                    ch += j;
                    spaces[i].push(Space(ch));
                }
            }
        }
        catch(ObjectRow<Space>::AllocError){
            string reply;
            cout << "It would seem that there is not enough\n"
                 << "available memory on your system to play\n"
                 << "this game, or something has gone terribly\n"
                 << "wrong. Not to pannic; I can try again, type\n"
                 << "[yes] to try again.\n";
            getline(cin, reply);
            if(tolower(reply[0]) == 'y'){
                retry = true;
            }
            else{
                cout << "Okay, exiting program- please inspect
your\n"

```

```

        << "System. Good bye.\n";
        exit(1);
    }
}
}while(retry == true);
//Default board is 3*3

//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(int base)
{
    bool retry = false;
    int ch = 55; //Interger base to assigned count character.
    root = base;
    numSpaces = root * root;
    //I could use a two dimensional array of space objects;
    //But the SearchableRow takes care of some tasks
    //Related to dynamic allocation.
    spaces = new SearchableRow<Space> [root];
    do{
        //This process may be repeated should the system cause
        //my vector object to throw a memory allocation error
        //Class for some temporary reason.
        try{
            for(int i = 0; i < root; i++){
                for(int j = 0; j < root; j++){
                    //Now to set each space to designate it's
                    //Key to be pressed to capture it.
                    ch -= i * root;
                    ch += j;
                    spaces[i].push(Space(ch));
                }
                ch = 55;
            }
        }
    } while(retry == true);
}

```

```

    }
    }
    catch(ObjectRow<Space>::AllocError){
        string reply;
        cout << "It would seem that there is not enough\n"
              << "available memory on your system to play\n"
              << "this game, or something has gone terribly\n"
              << "wrong. Not to pannic; I can try again, type\n"
              << "[yes] to try again.\n";
        getline(cin, reply);
        if(tolower(reply[0]) == 'y'){
            retry = true;
        }
        else{
            cout << "Okay, exiting program- please inspect
                  your\n"
                  << "System. Good bye.\n";
            exit(1);
        }
    }
}while(retry == true);

//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()
{
    delete [] spaces;
    spaces = 0; //<--Problem: system is calling this f(x) in
               //addition to the one I am calling.
               //Grounding the pointer fixes this
               //problem.
}

void Board::flipSpace(int pos, char token)

```

```

{
    //This algorithme will only work on a 3 by three board.
    int i;
    int j;
    if(pos >= numSpaces - (root - 1) && pos <= numSpaces)
    {
        //Top row.
        i = 0;
        j = pos - 7;        //This statement = 0, 1, or 2.
    }
    if(pos >= numSpaces - (2 * root - 1) && pos <= numSpaces - root)
    {
        //Middle row.
        i = 1;
        j = pos - 4;        //The index of the dynTArr pointer.
    }
    if(pos >= numSpaces - (3 * root - 1) && pos <= numSpaces - (root*2))
    {
        //Bottom row.
        i = 2;
        j = pos - 1;
    }
    spaces[i][j].flipSpace(token);
    //spaces[i][j].flipSpace(token); <-- this is identical to the old code.
}

void Board::showBoard()
{
    rightPos = 0;
    downPos = 0;
    //This was fun to work out.
    for(int row = 0; row < root + 2; row++){
        cout << "\t";
        //Based on evenness of outer for loop counter,
        //Call either the space containing row print f(x)
        //Or the seperation printing f(x);
        for(int col = 0; col < root + 2; col++){
            if(row % 2 == 0)
                printSpaces(col, row);
            else{
                printSeps(col);
            }
        }
        //Each time f(x) printSpaces is called,
        //The first index of the SearchableRow
        //Array must be incremented so as to work with
        //The next object.
        if(row % 2 == 1){
            downPos++;
        }
        //Each row starts at index 0.
        rightPos = 0;
        cout << endl;
    }
    cout << endl;
}

```

```
void Board::printSeps(int col)
{
    //These look like +      & = except more visually appealing.
    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)
{
    //This is the || symbol; connects with the + symbol of
    //The printSeps f(x) for a coherent looking board.
    char symb = 186;
    if(col % 2 == 0){
        cout << " " << spaces[downPos][rightPos] << " ";
        rightPos++;
    }
    else
        cout << symb;
}

//Implementation file for the Decahedron class.
//File: Decahedron.cpp
//Author: John Palacios

#include "Decahedron.h"

int Decahedron::roll()
{
    int result;
    result = rand() % numSides + 1;
    setDie(result);
    return checkDie();
}

void Decahedron::cheat(int num)
{
    if(num > 0 && num < numSides){
        setDie(num);
    }
    else{
        setDie(0);
    }
}

//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;
    srand(static_cast<unsigned>(time(0)));
    compStrat.blkOpYeX = false;
    compStrat.blkYeEx = false;
    compStrat.blkUndef = false;
    compStrat.blkConst = false;
    compStrat.constBlkd = -1;
    compStrat.undefBlkd = -1;
    compStrat.posConst = -1;
    compStrat.posUndef = -1;
    compStrat.nextMoveX = -1;
    compStrat.nextMoveY = -1;
}

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;
    srand(static_cast<unsigned>(time(0)));
    compStrat.blkOpYeX = false;
    compStrat.blkYeEx = false;
    compStrat.blkUndef = false;
    compStrat.blkConst = false;
    compStrat.constBlkd = -1;
    compStrat.undefBlkd = -1;
    compStrat.posConst = -1;
    compStrat.posUndef = -1;
    compStrat.nextMoveX = -1;

```

```

        compStrat.nextMoveY = -1;
    }

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.
    compStrat.numOpYeX = false; //set/reset conditional human play
flags.
    compStrat.numYeqEx = false;
    compStrat.numConst = false;
    compStrat.numUndef = false;

    //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 - 1 && act.playerToken == 'X'){
                compStrat.numOpYeX = true;
            }
            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 - 1 && act.playerToken == 'X'){
            compStrat.numYeqEx = true;
        }
        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 - 1 && act.playerToken == 'X'){
                compStrat.numConst = true;
                compStrat.posConst = i;
            }
            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 - 1 && act.playerToken == 'X'){
                compStrat.numUndef = true;
                compStrat.posUndef = i;
            }
            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 [help] to learn how to navigate game menus.\n"
         << "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 [single] to play a fast game against an easy AI.\n"
         << "type [cat] to play a fast game against a challenging AI.\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 opening 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 menu.\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;
    char option;
    getline(cin, choice);
    option = tolower(choice[0]);

    switch (option){
        case 'h':
            num = 1;
            break;
        case 'r':
            num = 2;
            break;
        case 'p':
            num = 3;
            break;
        case 'f':
            num = 4;
            break;
        case 's':
            num = 5;
            break;
        case 'c':
            num = 6;
            break;
        case 'v':
            num = 7;
            break;
        case 'e':
            num = 8;
            break;
        default:
            cout << "[" << choice << "]" was not an option, please try
again.\n";
            break;
    }
    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;
}

void Game::systemRandTurn()
{
    cout << "Turn number " << turnNum << endl;
    cout << "system's turn.\n";
    availMoves = getAvailableMoves();
    displayField();
    cout << "Available moves: ";

```

```

    for(int i = 0; i < numMoves; i++){
        cout << availMoves[i] << " ";
    }
    cout << endl;
    int move = rand() % numMoves;
    cout << "Your system chooses: " << availMoves[move] << "\n";
    cap(availMoves[move] - 48, oh);
    turnNum++;
    exTurn = true;
}

void Game::systemCatTurn()
{
    int move;
    cout << "Turn number " << turnNum << endl;
    cout << "Cat's turn.\n";
    availMoves = getAvailableMoves();
    displayField();
    cout << "Available moves: ";
    for(int i = 0; i < numMoves; i++){
        cout << availMoves[i] << " ";
    }
    cout << endl;
    if(turnNum == 2){
        for(int i = 0; i < root; i++){
            if(field->spaces[i].linSearch(Space('X')) != -1){
                compStrat.humanFstX = field-
>spaces[i].linSearch(Space('X'));
                compStrat.humanFstY = i;
            }
        }
        catTurnOne();
    }
    else if(turnNum == 4){
        catTurnTwo();
    }
    else if(turnNum == 6){
        catTurnThree();
    }
    else{
        move = rand() % numMoves;
        cout << "Cat chooses: " << availMoves[move] << "\n";
        cap(availMoves[move] - 48, oh);
    }
    turnNum++;
    exTurn = true;
}

void Game::catTurnOne()
{
    bool corner = false;
    bool center = false;
    Tetrahedron oneD4;
    Decahedron oneD10;
    int roll;
    //look at the corners of the board.
    for(int i = 0; i < root; i += 2){

```

```

        for(int j = 0; j < root; j += 2){
            if(field->spaces[i][j].owner == 'X'){
                //compStrat.humanFstY = i;
                //compStrat.humanFstX = j;
                compStrat.opener = 'A';
                corner = true;
            }
        }
    }
    //look at the center of the board.
    if(field->spaces[1][1].owner == 'X'){
        compStrat.opener = 'B';
        center = true;
    }
    //Smart first move by player ex, time to limit thier next move.
    if(corner == true){
        compStrat.fstPlayY = 1;
        compStrat.fstPlayX = 1;
        cap(5, oh);
    }
    else if(center == true){
        oneD4.roll();
        roll = oneD4.checkDie();
        switch(roll){
            case 1:
                compStrat.fstPlayY = 0;
                compStrat.fstPlayX = 0;
                cap(7, oh);
                break;
            case 2:
                compStrat.fstPlayY = 0;
                compStrat.fstPlayX = 2;
                cap(9, oh);
                break;
            case 3:
                compStrat.fstPlayY = 2;
                compStrat.fstPlayX = 0;
                cap(1, oh);
                break;
            case 4:
                compStrat.fstPlayY = 2;
                compStrat.fstPlayX = 2;
                cap(3, oh);
                break;
            default:
                cout << "Cat says, \"I think my d 4 is broken
:C.\\\"\\n\"
                << "Default case in catTurnOne.\\n\";
                break;
        }
    }
    else{
        compStrat.opener = 'C';
        oneD10.roll();
        roll = oneD10.checkDie() / 2;
        if(roll == 0){
            roll = 1;

```



```

    }
    switch(roll){
        case 1:
            compStrat.fstPlayY = 0;
            compStrat.fstPlayX = 0;
            cap(7, oh);
            break;
        case 2:
            compStrat.fstPlayY = 0;
            compStrat.fstPlayX = 2;
            cap(9, oh);
            break;
        case 3:
            compStrat.fstPlayY = 2;
            compStrat.fstPlayX = 0;
            cap(1, oh);
            break;
        case 4:
            compStrat.fstPlayY = 2;
            compStrat.fstPlayX = 2;
            cap(3, oh);
            break;
        case 5:
            compStrat.fstPlayY = 1;
            compStrat.fstPlayX = 1;
            cap(5, oh);
            break;
        default:
            cout << "Cat says, \"I think my d 10 is broken
:C.\\\"\\n\"
                << "Default case in catTurnOne.\\n\";
                break;
    }
}
if(compStrat.opener == 'A'){
    cout << "Cat says, \"I see that you've played this game
before.\\\"\\n\";
}
if(compStrat.opener == 'B'){
    cout << "Cat says, \"A decent first move for an amature.\\\"\\n\";
}
if(compStrat.opener == 'C'){
    cout << "Cat says, \"Perhaps you should try taking the center,\\n\"
        << "or a corner as your first move next time.\\\"\\n\";
}
}

void Game::catTurnTwo()
{
    Tetrahedron oneD4;
    //Player opened with the center capture...
    if(compStrat.opener == 'B'){
        if(compStrat.numConst){
            for(int i = 0; i < root; i++){
                for(int j = 0; j < root; j += 2){
                    if(field->spaces[1][i].owned == false){
                        field->spaces[1][i].flipSpace('O');

```

```

        compStrat.blkConst = true;
        compStrat.constBlkd = 1;
        cout << "\n";
        if(field->spaces[compStrat.fstPlayY][i].owned == false){
            compStrat.nextMoveX = i;
            compStrat.nextMoveY =

compStrat.fstPlayY;

        }
        else{
            compStrat.nextMoveX = 2 - i;
            compStrat.nextMoveY = 2 -

compStrat.fstPlayY;

        }
        break;
    }
    break;
}
}
}
if(compStrat.numUndef){
    for(int i = 0; i < root; i++){
        if(field->spaces[i][2].owned == false){
            cap(8 - root * i, oh);
            compStrat.blkUndef = true;
            compStrat.undefBlkd = 2;
            if(field->spaces[0][0].owned == false){
                compStrat.nextMoveX = 0;
                compStrat.nextMoveY = 0;
            }
            else{
                compStrat.nextMoveX = 2;
                compStrat.nextMoveY = 0;
            }
            cout << "\n";
        }
    }
}
//player is moving in line y=x.
else if(compStrat.numYeqEx){
    compStrat.blkYeqEx = true;
    //check/cap available space in line y = x.
    if(compStrat.fstPlayY == 0 && compStrat.fstPlayX == 2){
        cap(7, oh);
        compStrat.nextMoveX = 0;
        compStrat.nextMoveY = 1;
    }
    else if(compStrat.fstPlayY == 2 && compStrat.fstPlayX ==
0){
        cap(3, oh);
        compStrat.nextMoveX = 1;
        compStrat.nextMoveY = 2;
    }
    else{
        if(field->spaces[2][0].owned == false){
            cap(1, oh);

```

```

        compStrat.nextMoveX = 2;
        compStrat.nextMoveY = 2;
    }
    else
    {
        cap(9, oh);
        compStrat.nextMoveX = 0;
        compStrat.nextMoveY = 0;
    }
}
//The player is moving in line -y=x
else if(compStrat.numOpYeX){
    compStrat.blkOpYeX = true;
    //Check/cap available space in line y = -x.
    if(compStrat.fstPlayY == 2 && compStrat.fstPlayX == 2){
        cap(6, oh);
    }
    else if(compStrat.fstPlayY == 0 && compStrat.fstPlayX ==
0){
        cap(4, oh);
    }
    else{
        if(field->spaces[2][2].owned == false){
            cap(3, oh);
        }
        else
            cap(7, oh);
    }
}

}

if(compStrat.opener == 'A'){
    //If the player started in a corner and is going diagonal left to
right
    //The player is attempting to fork the game, take an edge to
block one
    //fork now.
    if(compStrat.numOpYeX == true){
        //This line is now blocked.
        compStrat.blkOpYeX = true;
        oneD4.roll();
        cap(oneD4.checkDie() * 2, oh);
        cout << "\n";
        switch(oneD4.checkDie()){
            case 1:
                compStrat.nextMoveX = 1;
                compStrat.nextMoveY = 0;
                break;
            case 2:
                compStrat.nextMoveX = 2;
                compStrat.nextMoveY = 1;
                break;
            case 3:
                compStrat.nextMoveX = 0;
                compStrat.nextMoveY = 1;

```

```

        break;
    case 4:
        compStrat.nextMoveX = 1;
        compStrat.nextMoveY = 2;
        break;
    }
}
//Now to check y = x.
if(compStrat.numYeqEx == true){
    //This line is now blocked.
    compStrat.blkYeEx = true;
    oneD4.roll();
    cap(oneD4.checkDie() * 2, oh);
    cout << "\n";
    switch(oneD4.checkDie()){
        case 1:
            compStrat.nextMoveX = 1;
            compStrat.nextMoveY = 0;
            break;
        case 2:
            compStrat.nextMoveX = 2;
            compStrat.nextMoveY = 1;
            break;
        case 3:
            compStrat.nextMoveX = 0;
            compStrat.nextMoveY = 1;
            break;
        case 4:
            compStrat.nextMoveX = 1;
            compStrat.nextMoveY = 2;
            break;
    }
}
//Now to check x = some constant.
if(compStrat.numUndef == true){
    for(int j = 0; j < root; j++){
        if(field->spaces[j][compStrat.humanFstX].owned ==
false){
            compStrat.blkUndef = true;
            compStrat.undefBlkd = compStrat.humanFstX;
            cap((7 - root * j) + compStrat.humanFstX, oh);
            compStrat.nextMoveX = 2 - compStrat.humanFstX;
            compStrat.nextMoveY = j;
        }
    }
}
//Player must be in y = constant.
if(compStrat.numConst){
    for(int j = 0; j < root; j++){
        if(field->spaces[compStrat.humanFstY][j].owned ==
false){
            compStrat.blkConst = true;
            compStrat.constBlkd = compStrat.humanFstY;
            cap((7 - compStrat.humanFstY * root) + j, oh);
            compStrat.nextMoveX = 2 - j;
            compStrat.nextMoveY = 2 - compStrat.humanFstY;
        }
    }
}

```

```

    }
}
}
//Need to check for players who started on the edges.
if(compStrat.opener == 'C'){
    //player is attempting to capture in line with undefined slope.
    if(compStrat.numUndef){
        for(int i = 0; i < root; i++){
            if(field->spaces[i].linSearch(Space('X')) == -1){
                //We know that the undefined line first marked
                by x is now blocked.

                compStrat.blkUndef = true;
                compStrat.undefBlkd = compStrat.humanFstX;
                if(field->spaces[i][compStrat.humanFstX].owned
                == false){

                    cap((7 - i * root) + compStrat.humanFstX, oh);
                    compStrat.nextMoveY = i;
                    compStrat.nextMoveX = (compStrat.humanFstX + 1)

                    % 3;

                }
                else{
                    oneD4.roll();
                    switch(oneD4.checkDie()){
                        case 1:
                            cap(7, oh);
                            compStrat.nextMoveX = 2;
                            compStrat.nextMoveY = 2;
                            break;
                        case 2:
                            cap(9, oh);
                            compStrat.nextMoveX = 0;
                            compStrat.nextMoveY = 2;
                            break;
                        case 3:
                            cap(1, oh);
                            compStrat.nextMoveX = 2;
                            compStrat.nextMoveY = 0;
                            break;
                        case 4:
                            cap(3, oh);
                            compStrat.nextMoveX = 0;
                            compStrat.nextMoveY = 0;
                            break;
                        default:
                            cout << "Cat says, \"I think
                                << "Error catTurnTwo,
                                break;
                    }
                }
            }
        }
    }
}
//The player is attempting to complete three accross.
if(compStrat.numConst){
    for(int i = 0; i < root; i++){

```

```

        //Cat only needs to check those searchable rows
containing spaces
        //Captured by x. To trip numConst flag, both x's will
be in a row.
        if(field->spaces[i].linSearch(Space('X')) != -1){
            //Now line y = i is blocked.
            compStrat.blkConst = true;
            compStrat.constBlkd = i;
            for(int j = 0; j < root; j++){
                if(field->spaces[i][j].owned == false){
                    cap((7 - root * i) + j, oh);
                    cout << "\n";
                }
            }
        }
    }
}

void Game::catTurnThree()
{
    bool goodIndex = false;
    bool capd = false;
    int move;

    //Ignore blocked lines with two player marks.
    if(compStrat.blkYeEx)
        compStrat.numYeqEx = false;
    if(compStrat.blkOpYeX)
        compStrat.numOpYeX = false;
    do{
        try{
            //Move in to victory or take next move.
            if(field-
>spaces[compStrat.nextMoveY][compStrat.nextMoveX].owned == false){
                cap((7 - compStrat.nextMoveY * root) +
compStrat.nextMoveX, oh);
                capd = true;
            }
            goodIndex = true;
        }
        catch(ObjectRow<Space>::OutOfBounds err){
            if(err.getDir()){
                compStrat.nextMoveY--;
                compStrat.nextMoveX--;
            }
            else{
                compStrat.nextMoveY++;
                compStrat.nextMoveX++;
            }
        }
    }while(goodIndex == false);
    if(capd == false){
        //Check for threats from the player.
        if(compStrat.numYeqEx && compStrat.blkYeEx == false){

```

```

        //This is not a false alarm. player threatens y = x.
        for(int row = 2, col = 0; col < root; row--, col++){
            if(field->spaces[row][col].owned == false){
                cap((7 - row * root) + col, oh);
                capd = true;
            }
        }
    }
    if(compStrat.numOpYeX && compStrat.blkOpYeX == false){
        //Player threatens y = -x.
        for(int i = 0; i < root; i++){
            if(field->spaces[i][i].owned == false){
                cap((7 - i * root) + i, oh);
                capd = true;
            }
        }
    }
    if(compStrat.numConst && compStrat.constBlkd !=
compStrat.posConst){
        for(int i = 0; i < root; i++){
            if(field->spaces[compStrat.posConst][i].owned ==
false){
                cap((7 - compStrat.posConst * root) + i, oh);
                capd = true;
            }
            else{
                cap(availMoves[rand() % numMoves], oh);
                capd = true;
            }
        }
    }
    if(compStrat.numUndef && compStrat.undefBlkd !=
compStrat.posUndef){
        for(int i = 0; i < root; i++){
            if(field->spaces[i][compStrat.posUndef].owned ==
false){
                cap((7 - i * root) + compStrat.posUndef, oh);
                capd = true;
            }
        }
    }
    else if(capd == false){
        move = rand() % numMoves;
        cap(availMoves[move], oh);
    }
}

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 classic 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 << "This menu system allows you to make selections by
typing the first or more\n"
                    << "letters of the word appearing in brackets in the
menu. Example:\n"
                    << "The text: \"type [next] to see the next entry.\"
means that you should type\n"
                    << "the letter \"n\" and then hit enter to see the
next entry. But you may also\n"

```



```

        << "type the letters \"n\" \"e\" \"x\" \"t\" and then
hit enter to accomplish the\n"
        << "same task- infact, you could type any combination
of letters so long as the\n"
        << "first one is \"n\" to see the next entry.\n"
        << "  ->Option Details<-\n"
        << "In the main menu, you have the option for this
message, rules, play, fast play,\n"
        << "single, cat, victories, and exit. Here's a quick
description of each.\n"
        << "rules:  This option gives you a simple run down
of the rules of tic-tac-toe and\n"
        << "  how to select a space for capture during game
play in either play or\n"
        << "  fastplay modes.\n"
        << "play:  This option allows you to enter your name
and your opponents name. This\n"
        << "  game is identical to the fast play game option
of tic-tac-toe, but the\n"
        << "  winner's name and the board configuration will
be saved, for later\n"
        << "  viewing using the [victories] option in the
main menu.\n"
        << "fast play:\n"
        << "  This option allows you and an opponent to play
a fast game of \n"
        << "  tic-tac-toe. Names do not need to be entered
and the board will not be\n"
        << "  saved when a player wins.\n"
        << "single:\n"
        << "  This option is a low dificulty one player game;
the system is player Oh\n"
        << "  whom will chose it's moves randomly... so you
can't be sure if you will\n"
        << "  win or get a cat's game.\n"
        << "cat:\n"
        << "  This option is a more diffcult one player
game; the system is\n"
        << "  player oh who will actively try to block
you.\n"
        << "victories:\n"
        << "  This option allows you to view saved
victoriesin the following format:\n"
        << "  Winner: Beck      <-- This is the name of the
winning player.\n"
        << "  Who's Board looked like:\n"
        << "      7 " << symb[0] << " 8 "<< symb[0] << "
X\n"
        << "      " << symb[1] << symb[1] << symb[1]<<
symb[2] << symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] <<
symb[1] << endl
        << "      4 " << symb[0] << " X "<< symb[0] << " 0
<- This is what Beck's board looked like.\n"
        << "      " << symb[1] << symb[1] << symb[1]<<
symb[2] << symb[1] << symb[1]<< symb[1] << symb[2] << symb[1]<< symb[1] <<
symb[1] << endl

```

```

O\n"
    << "          x " << symb[0] << " 2 "<< symb[0] << "
    << "          This is entry 1 of 5    <- *This means
that there are five\n"
    << "          entries availablefor viewing and you are
looking at the\n"
    << "          first one.*\n"
    << "          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"
    << "          *To view another entry; simply choose one
of the above options*\n"
    << "          *If you try to tpye back here, you will
hear a beep and see\n"
    << "          a messagetelling you that there are no
more options in that\n"
    << "          direction: simply type next instad.*\n";
    break;

case 2:
    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] << " 0
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] << " 0
\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"
    << "Configuration 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 3:
    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 4:
    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 5:
    do
    {
        Game *newGame;
        newGame = new Game();
        cout << static_cast<char>(7);
        do
        {
            if(newGame->getExTurn()){

```

```

        newGame->playerXturn();
    }
    else
        newGame->systemRandTurn();

} 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 6:
do
{
    Game *newGame;
    newGame = new Game();
    cout << static_cast<char>(7);
    do
    {
        if(newGame->getExTurn()){
            newGame->playerXturn();
        }
        else
            newGame->systemCatTurn();

    } 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 7:
    dispVictories();
    break;

case 8:
    cout << "Good bye.\n";

```

```
                break;
            }
        }while(choice != 8);
    }

//This file simply calls Game's static play member.
//File: Palacios_Main.cpp
//Author: John Palacios

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

int main()
{
    Game::play();
    return 0;
}

//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);
    }
}
```

```
bool Space::operator ==(const Space &right)
{
    bool flag;
    if (owner == right.owner){
        flag = true;
    }
    else
        flag = false;

    return flag;
}

//Implementation file for the Tetrahedron class.
//File: Tetrahedron.cpp
//Author: John Palacios

#include "Tetrahedron.h"

int Tetrahedron::roll()
{
    int result;
    result = rand() % numSides + 1;
    setDie(result);
    return checkDie();
}

void Tetrahedron::cheat(int num)
{
    if(num > 0 && num < numSides){
        setDie(num);
    }
    else{
        setDie(0);
    }
}
```