

Written by Billy D. Spelchan for www.BlazingGames.com

Copyright © 2003-2005 Blazing Games Inc. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the file called fdl.txt

Chapter 19

Tic Tac AI

Contents

As we did with Pent Up Anger, the Tic Tac Toe game we created in the last chapter was a two player game. This chapter we add a very challenging computer opponent.

- The AI Problem - A limitation of Flash that game developers should be aware of
- Planning the AI - How we are going to handle the ai for this game
- AI - Coding the AI
- Animated Title Sequence - Adding the title sequence
- About - Writing the about section
- Instructions - a brief look at how to put together in-game instructions

The AI Problem

One problem of handling AI in Flash is the Time limit Flash puts on scripts. For those of you who have yet to run into this, when Flash executes a script and the script takes too long to execute, Flash prompts you with a dialog box telling you the problem. To demonstrate, let us create a simple delay movie. On frame 1 put the message "Movie Starting". On frame 10 change the message text to "Delay code starting". Finally on frame 11 place the message "Done". Have an additional 10 frames in the movie.

The following code should be placed in frame 10

```
for (var cntr = 0; cntr < 1000000000; ++cntr) ;
```

Unless you run this movie on a machine significantly faster than the machines I have access to, when you run this you will be prompted that a script is taking too long to execute and that this could cause problems with your computer. If you select to keep running, the message will re-appear after a brief delay and keep re-appearing. If you select to stop the script, the movie continues. When the movie loops back to frame 10 you will notice that it does NOT try to run the script again.

Now, imagine that this was your AI thinking about what move to make. What would happen to your game if the AI was cancelled? And the AI script never ran again? Not good! The only good solution is to come up with an AI that executes really quick. This however, is not always possible, so we will briefly look at three ways to work around this problem.

The first way is to break the AI up into a bunch of smaller short executing steps and run one of these steps each frame of the thinking delay. As many AI techniques can be broken down into multiple steps, this can work fairly nice.

Similar to the first method, the second method would be to create a class or movie for the AI, and cheat by having the thinking method use global variables for all of it's internal working and have the internal working of the main function only execute a subset of the work. You would then have a controlling function that is called every frame and would call the controlling function getting it to continue where it left off.

Finally, another way would be to have the logic done in an outside program that Flash calls. Flash would then simply run the please wait code until the outside program called Flash to tell it that the work has been done. Obviously this adds a huge amount of potential problems. I haven't tried this myself so I can't comment on how well it would work.

Planning the AI

When thinking about how to handle the computer's intelligence for playing the Three Dimensional Tic Tac Toe game, the first thought that came to my mind was to use a recursive algorithm. This technique simply has the computer take a look at every possible move, with each move being examined by taking a look at the possible moves the opponent could make, with each of these moves then being examined for all the possible moves the computer could make which are then examined for ... well, you get the idea.

While this technique would work, each level of recursion takes exponentially longer than the previous level (well, not quite as the number of possible moves drops each level, but close enough). As you seen in the last section, this could be a big problem if the recursion takes long enough to cause the delay dialog box to appear.

This leads to the question, is there other ways of making the computer move intelligently? As programming almost always has a large number of solutions to any given problem, the answer is undoubtably yes. I approached this dilemma by working out how ultimately the recursion would be looking at this problem. Is there some way of condensing the calculations that the recursion is doing into something quick?

When you get right down to it, the AI is just looking at how many potential wins selecting a particular tile will get it as well as how many potential wins it is blocking the opponent from getting. We already have the win tables, so all that would have to be done is look at those tables and see how many potential wins each location would result in and how many potential wins would be blocked!

The advantages to using this technique? Speed and very easy implementation! The disadvantages? The computer plays an exceedingly good game! I hate losing to a machine! I suppose that the scoring tables could be adjusted to make the game a bit easier, but we will cover that next section.

AI

Thankfully, we don't have to worry about the AI taking too much time, as the method we are using for this takes advantage of the already existing win arrays to calculate the best move. The only thing that we need is a way of weighing the value of the number of X's or O's in a potential winning line. To do that we will add the following lines to the game initialization. If you want to make the game a bit easier, you can adjust the numbers in these tables so the computer pays a bit less attention to a potential win or loss.

```
ai_Scoring_O = new Array(1, 2, 6, 24);  
ai_Scoring_X = new Array(0, 1, 4, 16);
```

Now that we have all the tables to build the AI, we need a section of the movie to actually handle the AI. This section will be labelled "AI" and will slightly alter the "From Number" so it has a "Please Wait...thinking" message under the player number. The AI is so quick that this is probably not even needed.

We could create the AI in a function on the first frame, but we want the player to be able to play a single player game before the AI has been loaded. By having the code on the AI frame, the single player game will load slightly faster. In truth, there isn't much code here so it probably doesn't make a difference, still that is the way I originally had designed the game so I will leave it that way. Here is the code we place in the AI frame.

```
trace ("Computer is deciding it's move");  
var highest = 0;  
var highValue = -1;  
for (var cntr = 1; cntr < 65; ++cntr)  
{  
  
    var tileValue = 0;  
  
    if (getTileState(cntr) == 0)  
    {  
        var wins = winLists[cntr-1];  
        for (var cntrWin = 0; cntrWin < wins.length; ++cntrWin)  
        {  
            var curLine = wins[cntrWin];  
            var numberOfX = 0;  
            var numberOfO = 0;  
            // we don't know which array element the tile is, so just  
check them all  
            for (var cntrLine = 0; cntrLine < 4; ++cntrLine)  
            {  
                var tempTileState = getTileState(curLine[cntrLine]);  
                if (tempTileState == 1)  
                    ++numberOfX;  
                else if (tempTileState == 2)
```

```
                ++numberOfO;
            }
            // score the move on chance of computer winning and chance
player winning if not taken
            if (numberOfX == 0)
            {
                tileValue += ai_Scoring_O[numberOfO];
            }
            else if (numberOfO == 0)
            {
                tileValue += ai_Scoring_X[numberOfX];
            }
        }
    }
    else
    {
        tileValue = -2;
    }

    trace("Tile " + cntr + " scored " + tileValue);

    if (tileValue > highValue)
    {
        highest = cntr;
        highValue = tileValue;
        trace ("Currently selected tile changed to " + highest);
    }
    else if (tileValue == highValue)
    {
        if (random(2) == 1)
        {
            highest = cntr;
            highValue = tileValue;
            trace ("Currently selected tile changed to " + highest);
        }
    }
}

setTileState(highest, 2);
g_playedTile = highest;
g_player = 0;
gotoAndPlay("P2Move");
```

We still need to get the computer to use the AI. As this only happens when we are playing a single player game, we will need a variable that lets you know if the game is single or two player. As we haven't created the starting menu, where this variable would properly get set, we will have to manually force this variable to the single player mode. This is done by adding the following line to the top of frame 1. To help us remember that we will be removing this line later, a comment is also added to the line. It is always a good idea to clearly mark any test code that you add to your programs.

```
g_numberOfPlayers = 1; // testing - remove when menu implemented
```

Having a way of knowing if we are using the AI is a good start, but that is not enough. As the computer is always player 2, we can add the code to actually use the ai if the game is a single player game to the "P1Turn" code by replacing the existing code with the following.

```
if (g_numberOfPlayers == 1)
    gotoAndPlay("AI");
else
{
    g_player = 2;
    stop();
}
```

You will notice that the setting of the g_player variable and the stop() are in a block as part of the else block. If we didn't do this, the game would see the stop statement and stop the movie, preventing the AI from running.

Animated Title Sequence

The first thing that we have to do to get the title sequence going is to create the title backdrop. This is simply a black area where the title appears, a marble like rectangle for the buttons that has holes cut out to hold the four buttons, and an orange bar that is used for messages. The backdrop uses a tweened fade in.

The 3D tic tac toe message is tweened scaling and motion animation. The 3D is one object and each letter of tic tac toe is a separate object. Each object is on it's own layer.

As we want fancy marbled buttons, we import a marble texture that will be used for the buttons. Each of the four buttons uses this texture, but to make the buttons all distinct, the texture for each of the buttons is scaled, translated, and oriented different. The button appearing is simply a sliding motion tween.



Figure 1: Title Screen

Part of what I wanted when I originally created this game is for the game buttons to appear once enough of the flash code to allow that part of the game has been loaded. Doing this requires the use of a depreciated function. Each of the buttons is a simple motion tween with a movie loop that loops until enough of the movie has been loaded to allow the next button to be shown. The code for doing this is as follows.

```
ifFrameLoaded ("scene name here", "frame name here")
{
    gotoAndPlay("Name of next button sequence here");
}
```

A better way of handling loading is to use the movie clip methods `_framesloaded` and `_totalframes` methods. Flash MX 2004 even adds more flexibility by providing a `MovieClipLoader` class.

About

We start our about scene out with the games credits. The credits simply scroll from the bottom of the screen to the middle. Pause for a second and then scroll off the top of the screen. We have the following lines

Programming
Billy D. Spelchan

Graphics
Billy D. Spelchan

Animation
Billy D. Spelchan

Sound Effects
Billy D. Spelchan

Special Thanks
Crack Dot Com
Galgotha Public Domain Release

We then get to the pages of text. As we need a way to go through these pages, I again used my marble image in order to create a set of VCR like buttons that can be used for navigation.

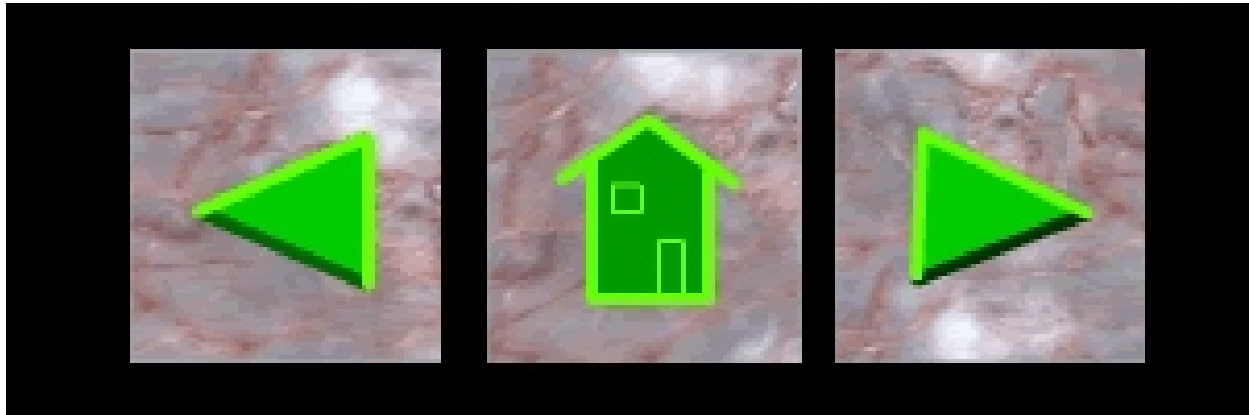


Figure 2 VCR Buttons

The first page, labelled “Why”, uses the following text.

Why did I do this game?

I had played with earlier releases of Flash but found that the programmability of those versions were far too restrictive. When Flash 5 came out I heard claims that the scripting language had been greatly enhanced. I had to see for myself. While the documentation clearly showed a much better programming environment than previous versions of Flash, the only way to be really sure of how well something works is to dive in and do something with the technology. So, I needed a project.

The home button had the following code attached to it

```
on (release)
{
    gotoAndStop("Title", "MenuWait");
}
```

The code attached to the forward button is

```
on (release)
{
    gotoAndStop("Goals");
}
```

The second page is labelled “Goals” and has all three vcr buttons. The text is as follows.

The goals of the project

The project had to be:

- + Something that can be put together quickly
- + Something that requires complex logic
- + Something that people can have fun playing

3D Tic Tac Toe meets these requirements.

The back button code:

```
on (release)
{
    gotoAndStop("Why");
}
```

The home button code is unchanged. The forward button’s code:

```
on (release)
{
    gotoAndStop("Spelchan");
}
```

Page three is labelled “Spelchan” and contains the following text.

About Blazing Games Inc.

I founded Spelchan Software in October of 1997. Originally I was releasing games every month but with increasing demands for my services found that this was just becoming too difficult to maintain such a tight schedule. After my workload eased up, I got back into developing games by starting my BlazingGames website. While this site releases new content every week. In October of 2003 I teamed up with two other people to form Blazing Games Inc. While the Blazing Games site will continue to run, the ultimate goal of the new company is the development of higher quality "commercial" games.

The back button has the following code attached to it:

```
on (release)
{
    gotoAndStop("Goals");
}
```

The home again is unchanged. The forward button has the following code:

```
on (release)
{
    gotoAndStop("Contact");
}
```

Finally, the last page is labelled “Contact” and has the following text.

Contacting Blazing Games Inc.

Blazing Games Inc. develops custom software in a variety of programming languages and licenses it's existing library of tiles to anyone who is interested. If you are interested in licensing software or having Blazing Games Inc. develop some custom software for you, direct inquiries to me at business@blazinggames.com.

The home button's code is unchanged while the back button has the following code:

```
on (release)
{
    gotoAndStop("Spelchan");
}
```

Instructions

As we already have VCR buttons that we created for the about sequence, using them in the instructions is a no-brainer. Now we simply assemble a series of pages describing the basic game flow.

The pages are assembled pretty much the way the instructions were so I won't go into details.

The biggest thing in the instructions was the demonstration of the seven distinct ways of winning the game. The best way of doing this is to actually have animated sequences showing the wins. These animated sequences are very easy to make. Essentially you have the four board layers, and you simply add the X images to the appropriate location every ten frames.