

Written by Billy D. Spelchan for [www.BlazingGames.com](http://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 20

## Board Games Summary

### Contents

This is a simple summary of what was learned in this part of the book and some suggestions on how you can applied what was learned here towards your own projects.

- Chapter 15 Board Game Overview - summarized
- Chapter 16 Pent Up Anger - summarized
- Chapter 17 Pent Up AI - summarized
- Chapter 18 Three dimensional Tic Tac Toe - summarized
- Chapter 19 Tic Tac AI - summarized
- Projects - some ideas for projects

## Chapter 15: Board Game Overview

### **Board Games**

Board games can be broken into three styles of games: luck, skill, and both. Games that are entirely based on luck require no decision making. An example of such a game are those track based games where you roll a die and move your piece along a linear path, occasionally landing on a square that gives you movement instructions. Skill based games give the player lots of options but have no random element. Chess is the best example here. In between games have choices and a bit of randomness. This type of game is still largely won by the more skilled player but a lesser skilled player can occasionally win. A good example of this type of game is backgammon.

### **Why Computer Versions?**

One question that a lot of people have is why even bother to create a computer version of a board game when you can just play the game using the board? There are five main reasons that I would do a computerized version of a board game: convenience, no lost pieces, fair referee, solo play, and computer enhancements.

### **Pent Up Anger**

The first game is a traditional track based board game, but with a bit of a twist. First, the players each have five pieces, with the ability to move any piece in a given turn. Second, the game uses an eight sided die. Third, the game is played on a five sided (Pentagram) board. The goal is to get all of your pieces from their starting location to their ending location. However, if you land on an opponent's piece, you send them to their starting location.

### **Three Dimensional Tic Tac Toe**

The second game we develop in this part of the book is a three dimensional version of the classic game Tic Tac Toe. The way you win this game is by forming a line consisting of four of your pieces while preventing your opponent from doing the same. The board is made up of 4 layers. Each of these layers is made up of 4 rows and four columns. You may place your piece on any of the 64 locations available, but can not place a piece on an occupied location.

## Chapter 16 Pent Up Anger

### **A Detailed look at Pent Up Anger**

Pent up Anger is an original board game that I designed which is loosely based on other board games that I have seen. The game's board is pentagon shaped. Each player is assigned a different color and has five playing pieces numbered one through five. Turns revolve clockwise around the board. Players start their turn by rolling an eight sided dice. Before the player can move, they need to get one of their pieces out of the starting gate. Players can only move or start once piece a turn. If the player lands on an opponent's piece, that piece get's taken back to the starting gate. Once the player's piece has landed in the loading zone, the piece is able to leave the board. Players win the game once all their pieces have been removed from the board.

### **Building the Board**

I draw a 24x24 box and duplicate it to get a row of 12 boxes. Rotating 5 times at 72 degrees each rotation allows you to create the pentagon shape. Join and clean up the corners and you have the board. The starting and ending zones are blocks of 5 squares and can easily be moved and rotated to the appropriate positions. Add a bit of color and you have the board.

### **Building the Player**

Every piece is going to have three states associated with it. There is also a number on each piece. Even worse, there are 5 colors. 3x5x5 is 75, which is a lot of pieces to build so we are going to have to cheat. Creating a grayscale version of the piece with the three states supported is easy enough. A bit of code to control all of this and we have the player's piece.

### **Finishing the Piece**

To get the appropriate color, we use Flash's tinting ability. To get numbers we use Flash's dynamic text ability. This is done in a new movie which makes use of the piece object created last section. We have some functions that are used for manipulating this movie and for getting and setting some of the movies information.

### **Building the Die**

This game uses an eight sided die. Building the die movie is fairly simple. First we start with an image of the die in a finished position. On a separate layer we have text for the eight different values (labelled r1 to r8). We also have mid-roll die images. As with our other movies, we treat the die movie as a class. To handle the rolling of the die we will need some functions for initialising the die, starting the roll, animating the roll, and finishing the roll.

### **Board Layout**

Pieces need locations. We need to know the coordinates of these locations so we can place the pieces on the proper screen locations. We could create a table by hand that contains all the coordinates of the locations. This is time consuming and very prone to errors. For board games that have complex layouts, this may be required. As our board is laid out in a fairly linear and consistent fashion, we can algorithmically create the table.

### **Preparing the Pieces**

At the end of initialization, the pieces used by the player exist in a two dimensional array named pieces. None of the pieces have been assigned their color, number or properly placed on the board. As we have all the starting information in the BOARD\_LAYOUT array that we created, all we have to do is grab the value from that table using the appropriate indexes and adjusting the value of the piece to lay out's coordinates, color, and number.

### **Turn Handling**

At this point we are ready to get the game under way. The first thing we need is a way for the player to roll the die. This can be combined with our solution for determining which players' turn it is by simply having a roll button for each player. The first task then is to create a roll button. As we did with the pieces, instead of creating five buttons we only create one grayscale version and tint it. We place these buttons on the board to correspond with the player's turn and add a bit of code to handle the roll.

### **Selecting the Piece**

With the roll made, the next task is to allow the player to select the piece to move. This is done in the board movie and is simply a matter of determining which pieces can be moved and setting their mode to highlight and enabling mouse click on those pieces.

### **Moving the Piece**

While it is quite possible to just move the piece to the destination, it is more fun if the piece actually moves to the desired location. This means that we are going to have to write our own motion routines to handle the animation. These routines should be familiar to those of you who read the Bomb Nim chapters of this book.

### **Winning the Game**

To see if the current player has won, we simply need to check to see if all the pieces are in their end positions. A simple function in the board movie can handle this. Next we simply need to create five win pages with a generic (so the color can be tinted) continue button that sends the movie to the title screen. With the winning pages created, we need a way to reach these pages. And now we have a playable version of the game.

## Chapter 17 Pent Up AI

### **AI in Board Games**

AI stands for Artificial Intelligence. It is commonly used in computer games to describe a computer controlled opponent. There are two basic techniques that are used when it comes to AI for board games. They are Recursive Algorithms, and Decision Trees. Some more complicated board games (such as war games) may require other techniques, but for most classic board games these two techniques should work.

### **Planning Pent Up AI**

For Pent Up Anger we are going to use a decision tree to handle the computer moves. This is largely because of the random nature that the game has. As we don't know what the player's will roll, all decisions on the move will have to be based on the current state of the board. First, we assign a value to every piece and move the piece with the highest value. The weight value reflects the priority, so the higher the step is on the list of questions, the higher the weight.

### **Tri-state buttons**

At this point in time we have a complete and playable game. I would like it possible to play against a computer player and also to leave some of the players out of the game. This means that we are going to need some type of menu system that lets the player choose who controls a particular color. As every color is going to have one of three choices, and only one of the three choices is valid at a time, it makes sense to create a component that lets the player choose one of three states. As there is no such component built into flash, this tri-state button then needs to be created.

### **Title Menu**

Now we can assemble the title screen. First, a simple logo. Next we put together the menu of options and the "Start Game" button. The menu is made up of five of the tri-state buttons that we created in the last section. Each of the tri-state buttons is tinted the appropriate color. To get the title screen to actually do something, we are going to need to add some code. To prevent an infinite loop from occurring, it also makes sure that there is at least one human or computer player.

### **Adding skipping**

One of the options in the tri-state menu's is "No player". This is to allow a small number of players play against each other without having any computer opponents. After playing through the game I noticed that it would be nice to be able to skip a turn. Especially in cases where your only move is to move your piece onto a player's launching area when that launching area is nearly full! Having no opponent is the same as skipping a turn so it would only be a marginal amount of effort to add a skip feature.

### **Making a Computer Opponent**

As computers can't think, we need to give the computer a set of rules for playing. To determine the weight, we are going to make changes to the prepareMove function that is located in the board symbol. Quite simply, we are assigning a weight to each move. The weight is assigned using the rules we outlined earlier in the chapter. The actual routine that uses the weight is also placed in the board symbol. It simply goes through the pieces and moves the piece with the highest weight.

### **Fine Tuning the code**

The first thing I am going to add to the game to give it more of a finished feel is a sound when the dice is rolled. More sound for when a piece moves should now be added. This is a bit trickier as the sound for handling this is going to have to be handled entirely with Action Script. The final sound that we are going to add to the game is my over-used win sound which will be played when one of the players wins. The game is now fairly good. One thing still bothers me, however. The skip button is visible when the result of the dice hasn't even been revealed as well as when the player has started to move so we are going to have to hide it.

## Chapter 18 Three Dimensional Tic Tac Toe

### **Tic Tac Toe Overview**

I originally wrote this game just after Flash 5 came out in order to see if the new Action Script scripting language would meet my game design needs. The way you win this game is by forming a line consisting of four of your pieces while preventing your opponent from doing the same. The board is made up of 4 layers. Each of these layers is made up of 4 rows and four columns. You may place your piece on any of the 64 locations available, but can not place a piece on an occupied location. As with tic tac toe, 3D Tic Tac Toe is a turn based game. This means that the first player makes a move, and only after he or she has made a move can the second player make a move. By Tic Tac Toe tradition, the first player is assigned X and the second player is assigned O. In single player mode, the computer always plays player 2.

### **Creating the Board**

The first thing we want to do is work out an image for the board. Because this is a three dimensional game, we want to give the board the appearance of depth. I also wanted to give it a bit of hand-drawn feel. This image is placed into a movie clip called BoardLayer as the layers are going to have extra functionality added to them. We then create four layers in the game scene of the movie for the four plains of the board. If a fancier backdrop for the game is desired, you could also have a fifth layer for the backdrop.

### **Tiles**

Every location on a playfield is going to need it's own x or o to be placed there. More important, the x or o must be able to be dimly visible as the player moves the mouse around and must be highlighted when the game has been won! All of these conditions can be combined into a single movie by having the following states: Empty, X\_Highlight, O\_Highlight, MakeX, X, WinningX, MakeO, O, and WinningO. Once all the sequences have been built, we need a bit of code to control the tile. We will need a function to turn on or off the highlight state. We also need to know what symbol is in any given tile and we need to highlight the winning hand, so an ability to set the tile to a win state is needed.

### **Adding Tiles to the Board**

Now that we have tiles, we are going to need to add them to the board. We simply return to the BoardLayer movie. For the 16 squares on the layer we add an instance of the tile. We need code to actually handle the tiles. These consist of array initialization and passthrough functions for getting and setting the various aspects of a tile.

### **Taking Turns**

The first thing we are going to need is a way to let the players know who's turn it is. This can be handled by having the player information written on the side of the screen. Two additional layers are needed to hold number transition animations. Once the player has made a move, the from number will be faded out and the to number, the number of the player to play next, will be faded in.

### **Handling the Mouse**

Mouse handling is handled by assigning our own functions to onMouseMove and onMouseUp. We only do any mouse work when it is a player's turn. We also create some support functions. The clearTargets function quite simply resets the entire board. Instead of having to worry about which board the player is interacting with, we are using a number between 1 and 64 to find the particular tile that is being handled. To do this we have pass through functions that take the desired number and convert it into the appropriate number calling the appropriate layer. Passthrough functions for the setHighlight, showWins, findTile, setTileState, getTileState, and setWinState are needed.

### **Winning Array**

Determining a win is a fair bit of work. To simplify this task we will create a 64 element array. Each element of the array, which corresponds to a particular tile, will contain an array of varying size. This array contains a reference to an array that contains the four tiles that form the winning move.

### **Checking for a Win**

Now we have the master array, we can now quickly see if the game has been won. We write a simple function to handle this. This simply finds the winArray entry for the tile and goes through each of the winning combinations. If one of the lines contains all four tiles set to the player's tile then the game has been won!

### **Winning the Game**

Now we want to create fancy winning sequences. First we will create the sequence for X winning. The animation has the O in the middle as the winning X's dance around it. The O winning sequence is done the same way except that the x's and O's are switched.



## Chapter 19 Tic Tac AI

### **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. Worse, if the user does stop the script, that script never runs again. What would happen to your game if the AI was cancelled? And the AI script never ran again? Not good! Alternatives? Span the AI over multiple frames. Create an ai that executes itself in small chunks possibly embedded into a Please Wait movie. Or possibly have an outside program handle the AI.

### **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. While this technique would work, each level of recursion takes exponentially longer than the previous level. 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!

### **AI**

With already existing win tables, the only additional 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. Some simple arrays holding weight values will do this trick. Next, we write the AI handling code. Due to the use of the tables, this code runs really fast! Finally, we add code to make the computer use the AI routines!

### **Animated Title Sequence**

We start the title sequence by creating a simple backdrop. The 3D tic tac toe message is tweened scaling and motion animation. As we want fancy marbled buttons, we import a marble texture that will be used for the buttons. 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. 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. 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.

### **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.

## Projects

With what you have learned in this section of the book, you should have no trouble creating your own Flash based board games. What would I recommend for starting projects? I would take the knowledge that I had just learned and apply it to an existing board game. I would start out by trying to create a simple track based board game and then perhaps try my hand at a more complex board game such as backgammon or checkers. Chess would also be a possible game to do, though the AI for that game could prove to be a huge obstacle.

When creating new board games it is not a bad idea to create the game on paper first using playing pieces from other board games. This way, you can get the game partially balanced before doing any work. Likewise, if the game proves to be poor, you haven't lost as much time as you would have if you created the game only on the computer.

I would highly advise readers who are creating their own games to take a look at my Three Dimensional Tic Tac Toe instruction section. The built in instructions for that game were very well done and really help novice players understand how to play the game.