

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 18

## Three Dimensional Tic Tac Toe

### Contents

The second game is the every popular Tic Tac Toe, except this time the game is three dimensional. The addition of a third dimension really adds to the complexity of the game making a very simple game into a much more challenging task.

- Tic Tac Toe Overview - How the game is played.
- Creating the Board - Building the board
- Tiles - Building the Xs and Os
- Adding Tiles to the Board - Populating the Board
- Taking Turns - The game flow
- Handling the Mouse - Actually being able to play the game
- Winning Array - Working out the possible ways of winning
- Checking for a Win - Seeing if player won
- Winning the Game - Winning animation.

## Tic Tac Toe Overview

When Flash 5 came out I heard claims that the scripting language had been greatly enhanced from what they had in previous versions. I had worked with earlier releases of Flash but found that the programmability of those versions had been far too restrictive. I had to see for myself if the scripting was good enough for game development. So, I needed a 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

I ended up deciding on Three-Dimensional Tic Tac Toe. This is a 3 dimensional version of the classic tic tac toe game. This game was originally created with Flash 5 in January of 2001, though I never actually released the game until I got a copy of Flash MX.

The game has a title that has four buttons that appear as the game is being loaded in the background. When the 4 menu buttons appear, click on the one you want to start.

- The single player game pits you against a very tough opponent. In fact, I was very impressed with how well the games' AI worked.
- The two player game lets you play against another person (on the same computer).
- The instructions provide a more graphical version of the instructions below.
- About is a must read section. Why? Well, you will just have to go there to find out!

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. We will be looking at the various types of winning lines later on in the instructions. 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.

Lines can be formed horizontally, vertically, or elevated. Combinations of the 3 are also allowed. There are a lot of ways to win (or lose) the game

### **Horizontal Win**

A horizontal win is a line that is straight across a level. There are 16 different ways to win horizontally.

### **Vertical Win**

I am defining vertical as what would be vertical if a level was two dimensional. There are 16 different ways to win vertically.

### **Horizontal and Vertical Win**

In the classic game, this type of win was commonly referred to as diagonal. Each level has two of these wins making for a total of 8 wins.

### **Elevated Win**

The elevated win is defined as a win where the same location in each of the 4 separate levels contains the same piece. As each level has 16 pieces, there are 16 different elevated wins.

### **Elevated Horizontal Win**

The diagonal but with a three dimensional twist to it. There are 8 of this type of diagonal.

### **Elevated Vertical Win**

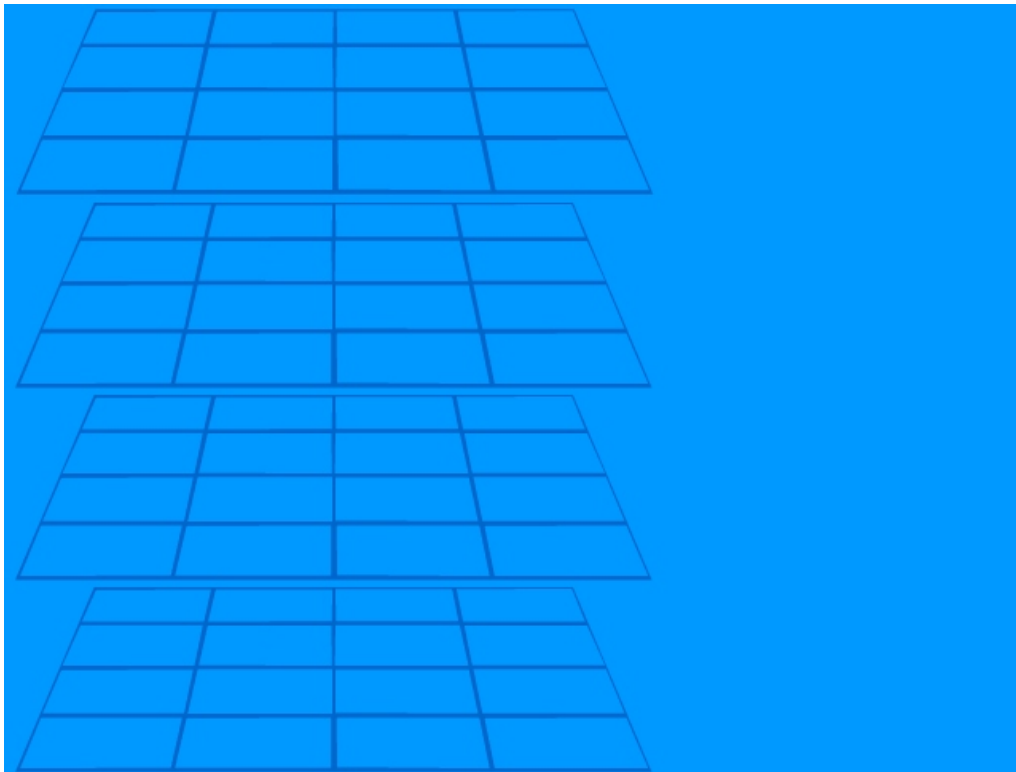
The diagonal but with a three dimensional twist to it. There are 8 of this type of diagonal.

### **Elevated Horizontal and Vertical Win**

The ultimate diagonal! There are four of these! I personally consider winning with one of these to be the ultimate win (or most humiliating loss).

## 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. To accomplish this, I worked out a layer of the board using lines and then drew the board by hand around this guideline. While I could have done a much rougher version than I finished, with, I decided not too.



**Figure 1:** Layout of game

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.

Next we animate the four levels of the board appearing on the screen. This is simply a motion tween of the board object moving from off the screen to it's proper position.

## Tiles

Now we know what the game layout is like so we can spend a bit of time and create the image that we are going to use for x and for 0. The only real requirement is that the X and O image be able to fit inside the squares, though a three dimensional look to them is also nice. In other words, we want the top of the image to be smaller than the bottom of the image so it looks like the object is lying on the board.



**Figure 2:** X and O images

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. We create the following sequences.

**Empty.** This sequence is essentially a single frame that contains dots at four corners. While we could have this completely blank, it is easier to place an object if there is some visible reference.

**X\_Highlight.** To create this sequence we take the x piece we created last chapter and set it's alpha level to 20%. The purpose of this frame is to give players an idea of where their piece is going to be placed.

**O\_Highlight.** This sequence is done the same way the X\_Highlight sequence was created except that we use the O piece.

**MakeX.** The purpose of this sequence is to visually show the X being placed on the board. The animation that we use to do this is your basic size tween. In other words, we start with the x scaled down. We then have a twenty frame gap and create a keyframe with the X at it's finishing size. Between the two keyframes we click and select create motion tween.

**X.** This is simply the X in the final position.

**WinningX.** This is the X object but to make it stand out we set the brightness to 50%.

**MakeO, O, WinningO.** These are all the O equivalents of the three previous X sequences.

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. By highlight we mean showing a dim version of the x or the o on tiles that are not occupied. 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.

```
function setHighlight(state)
{
    if (this._currentFrame >= 30)
        return;
    else if (state == 0)
        gotoAndStop(1);
    else if (state == 1)
        gotoAndStop(10);
    else if (state == 2)
        gotoAndStop(20);
}

function getTileState()
{
    if (this._currentFrame > 59)
        return 2;
    else if (this._currentFrame > 29)
        return 1;
    return 0;
}

function setWinState()
{
    if (this._currentFrame > 59)
    {
        gotoAndPlay("WinningO");
    }
    else if (this._currentFrame > 29)
    {
        gotoAndPlay("WinningX");
    }
}
```

## 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. The tiles are labelled using the R#C# convention.

We need code to actually handle the tiles, so we create a code layer in the BoardLayer movie where we create a keyframe on frame 2 to hold the code. The first action this frame takes is to stop the movie and call the initialization function which we call buildArray. The buildArray function we will create is a support function that simply makes sure all the tiles have been assembled into an array. As it is possible that this has already been done, we check to see if the length of the array is 16. While we could have used the standby routine of creating a boolean variable and checking to see if that variable has been defined, the array will be needed anyway so this works just as well.

```
buildArray();
stop();

function buildArray()
{
    // see if we already created the array, and if so just return
    if (boardTiles.length == 16)
    {
        return;
    }
    boardTiles = new Array(16);

    // add first row to array
    boardTiles[0] = r1c1;
    boardTiles[1] = r1c2;
    boardTiles[2] = r1c3;
    boardTiles[3] = r1c4;

    // add second row to array
    boardTiles[4] = r2c1;
    boardTiles[5] = r2c2;
    boardTiles[6] = r2c3;
    boardTiles[7] = r2c4;

    // add third row to array
    boardTiles[8] = r3c1;
    boardTiles[9] = r3c2;
    boardTiles[10] = r3c3;
    boardTiles[11] = r3c4;

    // add fourth row to array
    boardTiles[12] = r4c1;
    boardTiles[13] = r4c2;
    boardTiles[14] = r4c3;
    boardTiles[15] = r4c4;
}
```

We are going to have to write a variety of support functions as well, so let's start with a mouse support function. The `getTarget` function simply returns an id value of the tile that the mouse is over. The ID is in the range of 1 to 16 (first row returning 1 through 4, second row returning 5 through 8 and so on), with 0 being returned in the case when the mouse is currently not over any target. Quite simply, we loop through the array we built above and perform a `hitTest` to see if the point intersects the tile movie clip.

```
function getTarget()
{
    var rvalue = 0;

    for (cntr = 0; cntr < 16; ++cntr)
    {
        if (boardTiles[cntr].hitTest(_root._xmouse, _root._ymouse, false))
        {
            var rvalue = cntr + 1;
        }
    }
    return rvalue;
}
```

The next function will reset the board. The technique I used when originally creating the game is a bit esoteric, but I was just learning action script when I originally created this game (remember, this game was created with Flash 5, which is when Action Script was first introduced to Flash). What I did to reset the tile was simply move the tile's play head to the start of the tile which would be the equivalent to setting the tile to a blank frame.

```
function clearTargets()
{
    for (var cntr = 0; cntr < 16; ++cntr)
    {
        boardTiles[cntr].gotoAndStop(1);
    }
}
```

One important aspect of the tiles is the highlighting. If you remember, when the mouse is over a blank tile, the tile should show a dim version of the symbol that will be placed there if the player were to click there. The tile class already supports this concept, so we are just calling the tile's function. The thing is, the ID values that the game works with are one off, so the actual array element referenced has to be one less than the array element indicated.

```
function setHighlight(n, state)
{
    boardTiles[n-1].setHighlight(state);
}
```



*Blazing Games Guide to Flash Game Development Chapter 18: Three Dimensional Tic Tac Toe*

The next function simply sets the indicated tile to the desired state. State 0 is empty, State 1 is an X and State 2 is an O. As with the above function, the id passed to this function is one greater than the actual array location.

```
function setTileState(n, state)
{
    if (state == 0)
    {
        boardTiles[n-1].gotoAndStop(1);
    }
    else if (state == 1)
    {
        boardTiles[n-1].gotoAndPlay("MakeX");
    }
    else if (state == 2)
    {
        boardTiles[n-1].gotoAndPlay("MakeO");
    }
}
```

The opposite of setting a value is retrieving a value, so the next function returns the current state of the tile as follows: 0 is empty, 1 is an X and 2 is an O.

```
function getTileState(n)
{
    return boardTiles[n-1].getTileState();
}
```

Finally, we need to be able to call the tile's setWinState function, so we write a simple pass through function for handling this task.

```
function setWinState(n)
{
    return boardTiles[n-1].setWinState();
}
```

## Taking Turns

Three Dimensional Tic Tac Toe is a turn based game, so we are going to need to support turns. The actual support for making moves will be created next section. This section we just want the game to transition between the two players. 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.

To add this side information panel, three layers will be created. The first layer is the "Side Panel" layer and will be used to hold the word "Player." The word is written in a large bold font and is then broken apart twice and then converted into a symbol. I named the symbol "Player\_text." To make the side panel appear on the screen I have a very simple motion tween with the symbol moving from off the top of the screen to it's final resting location. This animation occurs between frames 40 and 50, which is when the last layer of the game board is appearing on the screen.

Two additional layers are needed to hold number transition animations. These two layers are labelled "From Number" and "To Number." The From Number will hold the current player. 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. The numbers will be two symbols, which hold either the number 1 or the number 2. They are created the same way the player symbol was created.

To handle this transition, we are going to need to create four different blocks of animation. We will start out on frame 50 with the label "P1Turn." This will be used for handling the first player's turn so the From Number will be a 1. Here is where we will have a bit of code that will be used to tell Flash which player is currently in control.

```
g_player = 1;  
stop();
```

Next we will have a section labelled "P1Move." This is where the transition animation between player 1 and player 2 takes place. The third movie block is labelled "P2Turn" and is where the second player's turn will take place. As with the first turn block, a bit of code is needed.

```
g_player = 2;  
stop();
```

Finally, we have a section labelled "P2Move" which has the transition animation between player 2 and player 1. The last frame of this animation loops back to the first players turn by using the following code.

```
gotoAndPlay("P1Turn");
```

## Handling the Mouse

In order to have mouse support, we also need to implement a bunch of support functions. In addition, while we are at it anyway, we are going to create additional support functions that we will need later. All of this code is placed in the code layer of frame 1.

We will start by setting up game control variables. As frame 1 should only be entered whenever a new game is started, placing them on this frame will cause the reset to happen every time a new game is started.

```
g_lastHighlighted = 0;  
g_player = 0;  
g_playedTile = 0;  
g_movesMade = 0;
```

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. If you recall last section, whenever it is a player's turn, the g\_player variable is assigned the value of the player. When we are between players (animating the player's move, for instance) the g\_player variable is set to 0.

The onMouseMove quite simply determines which tile the player is over by using the soon to be created findTile function. As we are working with the whole board now, tiles are given values between 1 and 64. The last tile that the mouse is over has the highlight removed while the new tile the mouse is over is highlighted.

```
onMouseMove = function()  
{  
    if (g_player == 0)  
        return;  
    var temp = findTile();  
    setHighlight(g_lastHighlighted, 0);  
    setHighlight(temp, g_player);  
    g_lastHighlighted = temp;  
}
```

The onMouseUp function is also used for returning to the menu when the game is over (which we will be handling later this chapter). To support this functionality, we simply assign the g\_player variable a number greater than 2. As with the mouse move, this function simply returns without doing anything if it is not one of the player's turn. The function is also returned if the tile selected is not an empty tile. If the tile is valid, the piece is placed, the g\_player variable is set to 0, and the appropriate move function is called so that the player turn transition takes place.

```
onMouseUp = function()
{
    if (g_player > 2)
        gotoAndStop("Title", "MenuWait");
    if (g_player == 0)
        return;
    var temp = findTile();
    if (getTileState(temp) != 0)
    {
        trace ("Tile " + temp + " is " + getTileState(temp));
        return;
    }
    setTileState(temp, g_player);
    g_playedTile = temp;
    var player = g_player;
    g_player = 0;
    if (player == 1)
        gotoAndPlay("PMove");
    else if (player == 2)
        gotoAndPlay("P2Move");
    else
        trace("Unknown player number!?");
}
```

The clearTargets function quite simply resets the entire board (all four plains of the board). It works by calling all of the board's clearTargets functions.

```
function clearTargets()
{
    plain1.clearTargets();
    plain2.clearTargets();
    plain3.clearTargets();
    plain4.clearTargets();
}
```

*Blazing Games Guide to Flash Game Development Chapter 18: Three Dimensional Tic Tac Toe*

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, so lets quickly create them. As you can see, the work is very straightforward.

```
function setHighlight(n, state)
{
    if (n < 17)
    {
        plain1.setHighlight(n, state);
    }
    else if (n < 33)
    {
        plain2.setHighlight(n-16, state);
    }
    else if (n < 49)
    {
        plain3.setHighlight(n-32, state);
    }
    else if (n < 65)
    {
        plain4.setHighlight(n-48, state);
    }
}

function showWins(n)
{
    if ((n < 1) || (n > 64))
    {
        return;
    }

    var wArray = winLists[n-1];

    for (var cntrl = 0; cntrl < wArray.length; ++cntrl)
    {
        var tArray = wArray[cntrl];
        for (var cntr2 = 0; cntr2 < tArray.length; ++cntr2)
        {
            setHighlight(tArray[cntr2], 2);
        }
    }
}
```

```
function findTile()
{
    // figure out what tile (if any) the mouse is over
    var temp = plain1.getTarget();
    if (temp == 0)
    {
        temp = plain2.getTarget();
        if(temp > 0)
            temp += 16;
    }
    if (temp == 0)
    {
        temp = plain3.getTarget();
        if(temp > 0)
            temp += 32;
    }
    if (temp == 0)
    {
        temp = plain4.getTarget();
        if(temp > 0)
            temp += 48;
    }

    return temp;
}

function setTileState(n, state)
{
    trace("inside setTileState(" + n + ", " + state + ")");
    if (n < 17)
    {
        plain1.setTileState(n, state);
    }
    else if (n < 33)
    {
        plain2.setTileState(n-16, state);
    }
    else if (n < 49)
    {
        plain3.setTileState(n-32, state);
    }
    else if (n < 65)
    {
        plain4.setTileState(n-48, state);
    }
}
```

```
function getTileState(n)
{
    if (n < 17)
    {
        return plain1.getTileState(n);
    }
    else if (n < 33)
    {
        return plain2.getTileState(n-16);
    }
    else if (n < 49)
    {
        return plain3.getTileState(n-32);
    }
    else if (n < 65)
    {
        return plain4.getTileState(n-48);
    }
}

function setWinState(n)
{
    if (n < 17)
    {
        return plain1.setWinState(n);
    }
    else if (n < 33)
    {
        return plain2.setWinState(n-16);
    }
    else if (n < 49)
    {
        return plain3.setWinState(n-32);
    }
    else if (n < 65)
    {
        return plain4.setWinState(n-48);
    }
}
```

## Winning Array

We now have a playable two player game. The only problem is that it is up to the players to tell the other player that they have won! That simply will not do. The thing is, 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.

In other words, we have an array of 64 elements, each element corresponding to a particular tile. Every tile on the board is part of anywhere from four to seven winning combinations. As all the winning combinations will be used on four tiles, it only makes sense that we re-use a winning combination once it has been used. For easy reference, I will have the array declaration that would have been made in comments. While you could have all the arrays combined in one huge array statement, I prefer creating a bunch of smaller arrays and then assembling them together at the end. This may take a bit longer for the computer to create, but as it is initialization code that only runs once this is hardly a concern.

First, of course, we call the initialization at the top of the frame's code.

```
initGame();
```

At this point I am half tempted to tell you to look at the source file, as the code is very long. Still, when I started writing this book I decided to have all the code in the book for the reader's convenience. So here it is...

```
function initGame()
{
    if (t1w1.length == 4)
        return;

    // Tile 1 (plain 1, Row 1, Column 1)
    t1w1 = new Array(1, 2, 3, 4);
    t1w2 = new Array(1, 5, 9, 13);
    t1w3 = new Array(1, 6, 11, 16);
    t1w4 = new Array(1, 17, 33, 49);
    t1w5 = new Array(1, 18, 35, 52);
    t1w6 = new Array(1, 21, 41, 61);
    t1w7 = new Array(1, 22, 43, 64);
    t1w = new Array(t1w1, t1w2, t1w3, t1w4, t1w5, t1w6, t1w7);

    // Tile 2 (plain 1, Row 1, Column 2)
    t2w1 = t1w1; // = new Array(1, 2, 3, 4);
    t2w2 = new Array(2, 6, 10, 14);
    t2w3 = new Array(2, 18, 34, 50);
    t2w4 = new Array(2, 22, 42, 62);
    t2w = new Array(t2w1, t2w2, t2w3, t2w4);

    // Tile 3 (plain 1, Row 1, Column 3)
```



```
t3w1 = t1w1; // = new Array(1, 2, 3, 4);
t3w2 = new Array(3, 7, 11, 15);
t3w3 = new Array(3, 19, 35, 51);
t3w4 = new Array(3, 23, 43, 63);
t3w = new Array(t3w1, t3w2, t3w3, t3w4);

// Tile 4 (plain 1, Row 1, Column 4)
t4w1 = t1w1; // = new Array(1, 2, 3, 4);
t4w2 = new Array(4, 8, 12, 16);
t4w3 = new Array(4, 7, 10, 13);
t4w4 = new Array(4, 20, 36, 52);
t4w5 = new Array(4, 19, 34, 49);
t4w6 = new Array(4, 24, 44, 64);
t4w7 = new Array(4, 23, 42, 61);
t4w = new Array(t4w1, t4w2, t4w3, t4w4, t4w5, t4w6, t4w7);

// Tile 5 (plain 1, Row 2, Column 1)
t5w1 = t1w2; // = new Array(1, 5, 9, 13);
t5w2 = new Array(5, 6, 7, 8);
t5w3 = new Array(5, 21, 37, 53);
t5w4 = new Array(5, 22, 39, 56);
t5w = new Array(t5w1, t5w2, t5w3, t5w4);

// Tile 6 (plain 1, Row 2, Column 2)
t6w1 = t1w3; // = new Array(1, 6, 11, 16);
t6w2 = t2w2; // = new Array(2, 6, 10, 14);
t6w3 = t5w2; // = new Array(5, 6, 7, 8);
t6w4 = new Array(6, 22, 38, 54);
t6w = new Array(t6w1, t6w2, t6w3, t6w4);

// Tile 7 (plain 1, Row 2, Column 3)
t7w1 = t3w2; // = new Array(3, 7, 11, 15);
t7w2 = t4w3; // = new Array(4, 7, 10, 13);
t7w3 = t5w2; // = new Array(5, 6, 7, 8);
t7w4 = new Array(7, 23, 39, 55);
t7w = new Array(t7w1, t7w2, t7w3, t7w4);

// Tile 8 (plain 1, Row 2, Column 4)
t8w1 = t4w2; // = new Array(4, 8, 12, 16);
t8w2 = t5w2; // = new Array(5, 6, 7, 8);
t8w3 = new Array(8, 24, 40, 56);
t8w4 = new Array(8, 23, 38, 53);
t8w = new Array(t8w1, t8w2, t8w3, t8w4);

// Tile 9 (plain 1, Row 3, Column 1)
t9w1 = t1w2; // = new Array(1, 5, 9, 13);
t9w2 = new Array(9, 10, 11, 12);
t9w3 = new Array(9, 25, 41, 57);
t9w4 = new Array(9, 26, 43, 60);
t9w = new Array(t9w1, t9w2, t9w3, t9w4);

// Tile 10 (plain 1, Row 3, Column 2)
t10w1 = t2w2; // = new Array(2, 6, 10, 14);
t10w2 = t4w3; // = new Array(4, 7, 10, 13);
t10w3 = t9w2; // = new Array(9, 10, 11, 12);
t10w4 = new Array(10, 26, 42, 58);
```

```
t10w = new Array(t10w1, t10w2, t10w3, t10w4);

// Tile 11 (plain 1, Row 3, Column 3)
t11w1 = t1w3; // = new Array(1, 6, 11, 16);
t11w2 = t3w2; // = new Array(3, 7, 11, 15);
t11w3 = t9w2; // = new Array(9, 10, 11, 12);
t11w4 = new Array(11, 27, 43, 59);
t11w = new Array(t11w1, t11w2, t11w3, t11w4);

// Tile 12 (plain 1, Row 3, Column 4)
t12w1 = t4w2; // = new Array(4, 8, 12, 16);
t12w2 = t9w2; // = new Array(9, 10, 11, 12);
t12w3 = new Array(12, 28, 44, 60);
t12w4 = new Array(12, 27, 42, 57);
t12w = new Array(t12w1, t12w2, t12w3, t12w4);

// Tile 13 (plain 1, Row 4, Column 1)
t13w1 = t1w2; // = new Array(1, 5, 9, 13);
t13w2 = t4w3; // = new Array(4, 7, 10, 13);
t13w3 = new Array(13,14,15,16);
t13w4 = new Array(13, 29, 45, 61);
t13w5 = new Array(13, 25, 37, 49);
t13w6 = new Array(13, 30, 47, 64);
t13w7 = new Array(13, 26, 39, 52);
t13w = new Array(t13w1, t13w2, t13w3, t13w4, t13w5, t13w6, t13w7);

// Tile 14 (plain 1, Row 4, Column 2)
t14w1 = t2w2; // = new Array(2, 6, 10, 14);
t14w2 = t13w3; // = new Array(13,14,15,16);
t14w3 = new Array(14, 30, 46, 62);
t14w4 = new Array(14, 26, 38, 50);
t14w = new Array(t14w1, t14w2, t14w3, t14w4);

// Tile 15 (plain 1, Row 4, Column 3)
t15w1 = t3w2; // = new Array(3, 7, 11, 15);
t15w2 = t13w3; // = new Array(13,14,15,16);
t15w3 = new Array(15, 31, 47, 63);
t15w4 = new Array(15, 27, 39, 51);
t15w = new Array(t15w1, t15w2, t15w3, t15w4);

// Tile 16 (plain 1, Row 4, Column 4)
t16w1 = t1w3; // = new Array(1, 6, 11, 16);
t16w2 = t4w2; // = new Array(4, 8, 12, 16);
t16w3 = t13w3; // = new Array(13,14,15,16);
t16w4 = new Array(16, 32, 48, 64);
t16w5 = new Array(16, 31, 46, 61);
t16w6 = new Array(16, 28, 40, 52);
t16w7 = new Array(16, 27, 38, 49);
t16w = new Array(t16w1, t16w2, t16w3, t16w4, t16w5, t16w6, t16w7);

// * Second plain

// Tile 17 (plain 2, Row 1, Column 1)
t17w1 = t1w4; // = new Array(1, 17, 33, 49);
t17w2 = new Array(17, 18, 19, 20);
t17w3 = new Array(17, 21, 25, 29);
```

```
t17w4 = new Array(17, 22, 27, 32);
t17w = new Array(t17w1, t17w2, t17w3, t17w4);

// Tile 18 (plain 2, Row 1, Column 2)
t18w1 = t1w5; // = new Array(1, 18, 35, 52);
t18w2 = t2w3; // = new Array(2, 18, 34, 50);
t18w3 = t17w2; // = new Array(17, 18, 19, 20);
t18w4 = new Array(18, 22, 26, 30);
t18w = new Array(t18w1, t18w2, t18w3, t18w4);

// Tile 19 (plain 2, Row 1, Column 3)
t19w1 = t3w3; // = new Array(3, 19, 35, 51);
t19w2 = t4w5; // = new Array(4, 19, 34, 49);
t19w3 = t17w2; // = new Array(17, 18, 19, 20);
t19w4 = new Array(19, 23, 27, 31);
t19w = new Array(t19w1, t19w2, t19w3, t19w4);

// Tile 20 (plain 2, Row 1, Column 4)
t20w1 = t4w4; // = new Array(4, 20, 36, 52);
t20w2 = t17w2; // = new Array(17, 18, 19, 20);
t20w3 = new Array(20, 24, 28, 32);
t20w4 = new Array(20, 23, 26, 29);
t20w = new Array(t20w1, t20w2, t20w3, t20w4);

// Tile 21 (plain 2, Row 2, Column 1)
t21w1 = t1w6; // = new Array(1, 21, 41, 61);
t21w2 = t5w3; // = new Array(5, 21, 37, 53);
t21w3 = t17w3; // = new Array(17, 21, 25, 29);
t21w4 = new Array(21, 22, 23, 24);
t21w = new Array(t21w1, t21w2, t21w3, t21w4);

// Tile 22 (plain 2, Row 2, Column 2)
t22w1 = t1w7; // = new Array(1, 22, 43, 64);
t22w2 = t2w4; // = new Array(2, 22, 42, 62);
t22w3 = t5w4; // = new Array(5, 22, 39, 56);
t22w4 = t6w4; // = new Array(6, 22, 38, 54);
t22w5 = t17w4; // = new Array(17, 22, 27, 32);
t22w6 = t18w4; // = new Array(18, 22, 26, 30);
t22w7 = t21w4; // = new Array(21, 22, 23, 24);
t22w = new Array(t22w1, t22w2, t22w3, t22w4, t22w5, t22w6, t22w7);

// Tile 23 (plain 2, Row 2, Column 3)
t23w1 = t3w4; // = new Array(3, 23, 43, 63);
t23w2 = t4w7; // = new Array(4, 23, 42, 61);
t23w3 = t7w4; // = new Array(7, 23, 39, 55);
t23w4 = t8w4; // = new Array(8, 23, 38, 53);
t23w5 = t19w4; // = new Array(19, 23, 27, 31);
t23w6 = t20w4; // = new Array(20, 23, 26, 29);
t23w7 = t21w4; // = new Array(21, 22, 23, 24);
t23w = new Array(t23w1, t23w2, t23w3, t23w4, t23w5, t23w6, t23w7);

// Tile 24 (plain 2, Row 2, Column 4)
t24w1 = t4w6; // = new Array(4, 24, 44, 64);
t24w2 = t8w3; // = new Array(8, 24, 40, 56);
t24w3 = t20w3; // = new Array(20, 24, 28, 32);
t24w4 = t21w4; // = new Array(21, 22, 23, 24);
```

```
t24w = new Array(t24w1, t24w2, t24w3, t24w4);

// Tile 25 (plain 2, Row 3, Column 1)
t25w1 = t9w3; // = new Array(9, 25, 41, 57);
t25w2 = t13w5; // = new Array(13, 25, 37, 49);
t25w3 = t17w3; // = new Array(17, 21, 25, 29);
t25w4 = new Array(25, 26, 27, 28);
t25w = new Array(t25w1, t25w2, t25w3, t25w4);

// Tile 26 (plain 2, Row 3, Column 2)
t26w1 = t9w4; // = new Array(9, 26, 43, 60);
t26w2 = t10w4; // = new Array(10, 26, 42, 58);
t26w3 = t13w7; // = new Array(13, 26, 39, 52);
t26w4 = t14w4; // = new Array(14, 26, 38, 50);
t26w5 = t18w4; // = new Array(18, 22, 26, 30);
t26w6 = t20w4; // = new Array(20, 23, 26, 29);
t26w7 = t25w4; // = new Array(25, 26, 27, 28);
t26w = new Array(t26w1, t26w2, t26w3, t26w4, t26w5, t26w6, t26w7);

// Tile 27 (plain 2, Row 3, Column 3)
t27w1 = t11w4; // = new Array(11, 27, 43, 59);
t27w2 = t12w4; // = new Array(12, 27, 42, 57);
t27w3 = t15w4; // = new Array(15, 27, 39, 51);
t27w4 = t16w7; // = new Array(16, 27, 38, 49);
t27w5 = t17w4; // = new Array(17, 22, 27, 32);
t27w6 = t19w4; // = new Array(19, 23, 27, 31);
t27w7 = t25w4; // = new Array(25, 26, 27, 28);
t27w = new Array(t27w1, t27w2, t27w3, t27w4, t27w5, t27w6, t27w7);

// Tile 28 (plain 2, Row 3, Column 4)
t28w1 = t12w3; // = new Array(12, 28, 44, 60);
t28w2 = t16w6; // = new Array(16, 28, 40, 52);
t28w3 = t20w3; // = new Array(20, 24, 28, 32);
t28w4 = t25w4; // = new Array(25, 26, 27, 28);
t28w = new Array(t28w1, t28w2, t28w3, t28w4);

// Tile 29 (plain 2, Row 4, Column 1)
t29w1 = t13w4; // = new Array(13, 29, 45, 61);
t29w2 = t17w3; // = new Array(17, 21, 25, 29);
t29w3 = t20w4; // = new Array(20, 23, 26, 29);
t29w4 = new Array(29, 30, 31, 32);
t29w = new Array(t29w1, t29w2, t29w3, t29w4);

// Tile 30 (plain 2, Row 4, Column 2)
t30w1 = t13w6; // = new Array(13, 30, 47, 64);
t30w2 = t14w3; // = new Array(14, 30, 46, 62);
t30w3 = t18w4; // = new Array(18, 22, 26, 30);
t30w4 = t29w4; // = new Array(29, 30, 31, 32);
t30w = new Array(t30w1, t30w2, t30w3, t30w4);

// Tile 31 (plain 2, Row 4, Column 3)
t31w1 = t15w3; // = new Array(15, 31, 47, 63);
t31w2 = t16w5; // = new Array(16, 31, 46, 61);
t31w3 = t19w4; // = new Array(19, 23, 27, 31);
t31w4 = t29w4; // = new Array(29, 30, 31, 32);
t31w = new Array(t31w1, t31w2, t31w3, t31w4);
```

```
// Tile 32 (plain 2, Row 4, Column 4)
t32w1 = t16w4; // = new Array(16, 32, 48, 64);
t32w2 = t17w4; // = new Array(17, 22, 27, 32);
t32w3 = t20w3; // = new Array(20, 24, 28, 32);
t32w4 = t29w4; // = new Array(29, 30, 31, 32);
t32w = new Array(t32w1, t32w2, t32w3, t32w4);

// * Third plain

// Tile 33 (plain 3, Row 1, Column 1)
t33w1 = t1w4; // = new Array(1, 17, 33, 49);
t33w2 = new Array(33, 34, 35, 36);
t33w3 = new Array(33, 37, 41, 45);
t33w4 = new Array(33, 38, 43, 48);
t33w = new Array(t33w1, t33w2, t33w3, t33w4);

// Tile 34 (plain 3, Row 1, Column 2)
t34w1 = t2w3; // = new Array(2, 18, 34, 50);
t34w2 = t4w5; // = new Array(4, 19, 34, 49);
t34w3 = t33w2; // = new Array(33, 34, 35, 36);
t34w4 = new Array(34, 38, 42, 46);
t34w = new Array(t34w1, t34w2, t34w3, t34w4);

// Tile 35 (plain 3, Row 1, Column 3)
t35w1 = t1w5; // = new Array(1, 18, 35, 52);
t35w2 = t3w3; // = new Array(3, 19, 35, 51);
t35w3 = t33w2; // = new Array(33, 34, 35, 36);
t35w4 = new Array(35, 39, 43, 47);
t35w = new Array(t35w1, t35w2, t35w3, t35w4);

// Tile 36 (plain 3, Row 1, Column 4)
t36w1 = t4w4; // = new Array(4, 20, 36, 52);
t36w2 = t33w2; // = new Array(33, 34, 35, 36);
t36w3 = new Array(36, 40, 44, 48);
t36w4 = new Array(36, 39, 42, 45);
t36w = new Array(t36w1, t36w2, t36w3, t36w4);

// Tile 37 (plain 3, Row 2, Column 1)
t37w1 = t5w3; // = new Array(5, 21, 37, 53);
t37w2 = t13w5; // = new Array(13, 25, 37, 49);
t37w3 = t33w3; // = new Array(33, 37, 41, 45);
t37w4 = new Array(37, 38, 39, 40);
t37w = new Array(t37w1, t37w2, t37w3, t37w4);

// Tile 38 (plain 3, Row 2, Column 2)
t38w1 = t6w4; // = new Array(6, 22, 38, 54);
t38w2 = t8w4; // = new Array(8, 23, 38, 53);
t38w3 = t14w4; // = new Array(14, 26, 38, 50);
t38w4 = t16w7; // = new Array(16, 27, 38, 49);
t38w5 = t33w4; // = new Array(33, 38, 43, 48);
t38w6 = t34w4; // = new Array(34, 38, 42, 46);
t38w7 = t37w4; // = new Array(37, 38, 39, 40);
t38w = new Array(t38w1, t38w2, t38w3, t38w4, t38w5, t38w6, t38w7);

// Tile 39 (plain 3, Row 2, Column 3)
```

```
t39w1 = t5w4; // = new Array(5, 22, 39, 56);
t39w2 = t7w4; // = new Array(7, 23, 39, 55);
t39w3 = t13w7; // = new Array(13, 26, 39, 52);
t39w4 = t15w4; // = new Array(15, 27, 39, 51);
t39w5 = t35w4; // = new Array(35, 39, 43, 47);
t39w6 = t36w4; // = new Array(36, 39, 42, 45);
t39w7 = t37w4; // = new Array(37, 38, 39, 40);
t39w = new Array(t39w1, t39w2, t39w3, t39w4, t39w5, t39w6, t39w7);

// Tile 40 (plain 3, Row 2, Column 4)
t40w1 = t8w3; // = new Array(8, 24, 40, 56);
t40w2 = t16w6; // = new Array(16, 28, 40, 52);
t40w3 = t36w3; // = new Array(36, 40, 44, 48);
t40w4 = t37w4; // = new Array(37, 38, 39, 40);
t40w = new Array(t40w1, t40w2, t40w3, t40w4);

// Tile 41 (plain 3, Row 3, Column 1)
t41w1 = t1w6; // = new Array(1, 21, 41, 61);
t41w2 = t9w3; // = new Array(9, 25, 41, 57);
t41w3 = t33w3; // = new Array(33, 37, 41, 45);
t41w4 = new Array(41, 42, 43, 44);
t41w = new Array(t41w1, t41w2, t41w3, t41w4);

// Tile 42 (plain 3, Row 3, Column 2)
t42w1 = t2w4; // = new Array(2, 22, 42, 62);
t42w2 = t4w7; // = new Array(4, 23, 42, 61);
t42w3 = t10w4; // = new Array(10, 26, 42, 58);
t42w4 = t12w4; // = new Array(12, 27, 42, 57);
t42w5 = t34w4; // = new Array(34, 38, 42, 46);
t42w6 = t36w4; // = new Array(36, 39, 42, 45);
t42w7 = t41w4; // = new Array(41, 42, 43, 44);
t42w = new Array(t42w1, t42w2, t42w3, t42w4, t42w5, t42w6, t42w7);

// Tile 43 (plain 3, Row 3, Column 3)
t43w1 = t1w7; // = new Array(1, 22, 43, 64);
t43w2 = t3w4; // = new Array(3, 23, 43, 63);
t43w3 = t9w4; // = new Array(9, 26, 43, 60);
t43w4 = t11w4; // = new Array(11, 27, 43, 59);
t43w5 = t33w4; // = new Array(33, 38, 43, 48);
t43w6 = t35w4; // = new Array(35, 39, 43, 47);
t43w7 = t41w4; // = new Array(41, 42, 43, 44);
t43w = new Array(t43w1, t43w2, t43w3, t43w4, t43w5, t43w6, t43w7);

// Tile 44 (plain 3, Row 3, Column 4)
t44w1 = t4w6; // = new Array(4, 24, 44, 64);
t44w2 = t12w3; // = new Array(12, 28, 44, 60);
t44w3 = t36w3; // = new Array(36, 40, 44, 48);
t44w4 = t41w4; // = new Array(41, 42, 43, 44);
t44w = new Array(t44w1, t44w2, t44w3, t44w4);

// Tile 45 (plain 3, Row 4, Column 1)
t45w1 = t13w4; // = new Array(13, 29, 45, 61);
t45w2 = t33w3; // = new Array(33, 37, 41, 45);
t45w3 = t36w4; // = new Array(36, 39, 42, 45);
t45w4 = new Array(45, 46, 47, 48);
t45w = new Array(t45w1, t45w2, t45w3, t45w4);
```

```
// Tile 46 (plain 3, Row 4, Column 2)
t46w1 = t14w3; // = new Array(14, 30, 46, 62);
t46w2 = t16w5; // = new Array(16, 31, 46, 61);
t46w3 = t34w4; // = new Array(34, 38, 42, 46);
t46w4 = t45w4; // = new Array(45, 46, 47, 48);
t46w = new Array(t46w1, t46w2, t46w3, t46w4);

// Tile 47 (plain 3, Row 4, Column 3)
t47w1 = t13w6; // = new Array(13, 30, 47, 64);
t47w2 = t15w3; // = new Array(15, 31, 47, 63);
t47w3 = t35w4; // = new Array(35, 39, 43, 47);
t47w4 = t45w4; // = new Array(45, 46, 47, 48);
t47w = new Array(t47w1, t47w2, t47w3, t47w4);

// Tile 48 (plain 3, Row 4, Column 4)
t48w1 = t16w4; // = new Array(16, 32, 48, 64);
t48w2 = t33w4; // = new Array(33, 38, 43, 48);
t48w3 = t36w3; // = new Array(36, 40, 44, 48);
t48w4 = t45w4; // = new Array(45, 46, 47, 48);
t48w = new Array(t48w1, t48w2, t48w3, t48w4);

// * Fourth plain

// Tile 49 (plain 4, Row 1, Column 1)
t49w1 = t1w4; // = new Array(1, 17, 33, 49);
t49w2 = t4w5; // = new Array(4, 19, 34, 49);
t49w3 = t13w5; // = new Array(13, 25, 37, 49);
t49w4 = t16w7; // = new Array(16, 27, 38, 49);
t49w5 = new Array(49, 50, 51, 52);
t49w6 = new Array(49, 53, 57, 61);
t49w7 = new Array(49, 54, 59, 64);
t49w = new Array(t49w1, t49w2, t49w3, t49w4, t49w5, t49w6, t49w7);

// Tile 50 (plain 4, Row 1, Column 2)
t50w1 = t2w3; // = new Array(2, 18, 34, 50);
t50w2 = t14w4; // = new Array(14, 26, 38, 50);
t50w3 = t49w5; // = new Array(49, 50, 51, 52);
t50w4 = new Array(50, 54, 58, 62);
t50w = new Array(t50w1, t50w2, t50w3, t50w4);

// Tile 51 (plain 4, Row 1, Column 3)
t51w1 = t3w3; // = new Array(3, 19, 35, 51);
t51w2 = t15w4; // = new Array(15, 27, 39, 51);
t51w3 = t49w5; // = new Array(49, 50, 51, 52);
t51w4 = new Array(51, 55, 59, 63);
t51w = new Array(t51w1, t51w2, t51w3, t51w4);

// Tile 52 (plain 4, Row 1, Column 4)
t52w1 = t1w5; // = new Array(1, 18, 35, 52);
t52w2 = t4w4; // = new Array(4, 20, 36, 52);
t52w3 = t13w7; // = new Array(13, 26, 39, 52);
t52w4 = t16w6; // = new Array(16, 28, 40, 52);
t52w5 = t49w5; // = new Array(49, 50, 51, 52);
t52w6 = new Array(52, 56, 60, 64);
t52w7 = new Array(52, 55, 58, 61);
```

```
t52w = new Array(t52w1, t52w2, t52w3, t52w4, t52w5, t52w6, t52w7);

// Tile 53 (plain 4, Row 2, Column 1)
t53w1 = t5w3; // = new Array(5, 21, 37, 53);
t53w2 = t8w4; // = new Array(8, 23, 38, 53);
t53w3 = t49w6; // = new Array(49, 53, 57, 61);
t53w4 = new Array(53, 54, 55, 56);
t53w = new Array(t53w1, t53w2, t53w3, t53w4);

// Tile 54 (plain 4, Row 2, Column 2)
t54w1 = t6w4; // = new Array(6, 22, 38, 54);
t54w2 = t49w7; // = new Array(49, 54, 59, 64);
t54w3 = t50w4; // = new Array(50, 54, 58, 62);
t54w4 = t53w4; // = new Array(53, 54, 55, 56);
t54w = new Array(t54w1, t54w2, t54w3, t54w4);

// Tile 55 (plain 4, Row 2, Column 3)
t55w1 = t7w4; // = new Array(7, 23, 39, 55);
t55w2 = t51w4; // = new Array(51, 55, 59, 63);
t55w3 = t52w7; // = new Array(52, 55, 58, 61);
t55w4 = t53w4; // = new Array(53, 54, 55, 56);
t55w = new Array(t55w1, t55w2, t55w3, t55w4);

// Tile 56 (plain 4, Row 2, Column 4)
t56w1 = t5w4; // = new Array(5, 22, 39, 56);
t56w2 = t8w3; // = new Array(8, 24, 40, 56);
t56w3 = t52w6; // = new Array(52, 56, 60, 64);
t56w4 = t53w4; // = new Array(53, 54, 55, 56);
t56w = new Array(t56w1, t56w2, t56w3, t56w4);

// Tile 57 (plain 4, Row 3, Column 1)
t57w1 = t9w3; // = new Array(9, 25, 41, 57);
t57w2 = t12w4; // = new Array(12, 27, 42, 57);
t57w3 = t49w6; // = new Array(49, 53, 57, 61);
t57w4 = new Array(57, 58, 59, 60);
t57w = new Array(t57w1, t57w2, t57w3, t57w4);

// Tile 58 (plain 4, Row 3, Column 2)
t58w1 = t10w4; // = new Array(10, 26, 42, 58);
t58w2 = t50w4; // = new Array(50, 54, 58, 62);
t58w3 = t52w7; // = new Array(52, 55, 58, 61);
t58w4 = t57w4; // = new Array(57, 58, 59, 60);
t58w = new Array(t58w1, t58w2, t58w3, t58w4);

// Tile 59 (plain 4, Row 3, Column 3)
t59w1 = t11w4; // = new Array(11, 27, 43, 59);
t59w2 = t49w7; // = new Array(49, 54, 59, 64);
t59w3 = t51w4; // = new Array(51, 55, 59, 63);
t59w4 = t57w4; // = new Array(57, 58, 59, 60);
t59w = new Array(t59w1, t59w2, t59w3, t59w4);

// Tile 60 (plain 4, Row 3, Column 4)
t60w1 = t9w4; // = new Array(9, 26, 43, 60);
t60w2 = t12w3; // = new Array(12, 28, 44, 60);
t60w3 = t52w6; // = new Array(52, 56, 60, 64);
t60w4 = t57w4; // = new Array(57, 58, 59, 60);
```



```
t60w = new Array(t60w1, t60w2, t60w3, t60w4);

// Tile 61 (plain 4, Row 4, Column 1)
t61w1 = t1w6; // = new Array(1, 21, 41, 61);
t61w2 = t4w7; // = new Array(4, 23, 42, 61);
t61w3 = t13w4; // = new Array(13, 29, 45, 61);
t61w4 = t16w5; // = new Array(16, 31, 46, 61);
t61w5 = t49w6; // = new Array(49, 53, 57, 61);
t61w6 = t52w7; // = new Array(52, 55, 58, 61);
t61w7 = new Array(61, 62, 63, 64);
t61w = new Array(t61w1, t61w2, t61w3, t61w4, t61w5, t61w6, t61w7);

// Tile 62 (plain 4, Row 4, Column 2)
t62w1 = t2w4; // = new Array(2, 22, 42, 62);
t62w2 = t14w3; // = new Array(14, 30, 46, 62);
t62w3 = t50w4; // = new Array(50, 54, 58, 62);
t62w4 = t61w7; // = new Array(61, 62, 63, 64);
t62w = new Array(t62w1, t62w2, t62w3, t62w4);

// Tile 63 (plain 4, Row 4, Column 3)
t63w1 = t3w4; // = new Array(3, 23, 43, 63);
t63w2 = t15w3; // = new Array(15, 31, 47, 63);
t63w3 = t51w4; // = new Array(51, 55, 59, 63);
t63w4 = t61w7; // = new Array(61, 62, 63, 64);
t63w = new Array(t63w1, t63w2, t63w3, t63w4);

// Tile 64 (plain 4, Row 4, Column 4)
t64w1 = t1w7; // = new Array(1, 22, 43, 64);
t64w2 = t4w6; // = new Array(4, 24, 44, 64);
t64w3 = t13w6; // = new Array(13, 30, 47, 64);
t64w4 = t16w4; // = new Array(16, 32, 48, 64);
t64w5 = t49w7; // = new Array(49, 54, 59, 64);
t64w6 = t52w6; // = new Array(52, 56, 60, 64);
t64w7 = t61w7; // = new Array(61, 62, 63, 64);
t64w = new Array(t64w1, t64w2, t64w3, t64w4, t64w5, t64w6, t64w7);

winLists = new Array(
    t1w, t2w, t3w, t4w, t5w, t6w, t7w, t8w,
    t9w, t10w, t11w, t12w, t13w, t14w, t15w, t16w,
    t17w, t18w, t19w, t20w, t21w, t22w, t23w, t24w,
    t25w, t26w, t27w, t28w, t29w, t30w, t31w, t32w,
    t33w, t34w, t35w, t36w, t37w, t38w, t39w, t40w,
    t41w, t42w, t43w, t44w, t45w, t46w, t47w, t48w,
    t49w, t50w, t51w, t52w, t53w, t54w, t55w, t56w,
    t57w, t58w, t59w, t60w, t61w, t62w, t63w, t64w );
}
```

## 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!

```
function checkIfWin(n)
{
    trace ("CheckIfWin(" + n + ")");
    var wins = winLists[n-1];
    var hasWon = false;
    var tileValue = getTileState(n);
    trace("tileValue = " + tileValue);
    for (var cntrWin = 0; cntrWin < wins.length; ++cntrWin)
    {
        trace("Checking potential win " + cntrWin);
        var curLine = wins[cntrWin];
        var winTest = true;
        // we don't know which array element the tile is, so just check
        them all
        for (var cntrLine = 0; cntrLine < 4; ++cntrLine)
        {
            trace(cntrLine + ". " + tileValue + " = " +
getTileState(curLine[cntrLine]));
            if (getTileState(curLine[cntrLine]) != tileValue)
                winTest = false;
        }
        if (winTest == true)
        {
            hasWon = true;
            for (var cntrLine = 0; cntrLine < 4; ++cntrLine)
            {
                setWinState(curLine[cntrLine]);
            }
        }
        else
            trace("result is false!");
    }

    return hasWon;
}
```

### *Blazing Games Guide to Flash Game Development Chapter 18: Three Dimensional Tic Tac Toe*

To actually use this win checking function, we are going to need to add a bit of code at the end of both of the PxMove sequences. In the last frame of P1Move we add the following code

```
if (checkIfWin(g_playedTile) == true)
    gotoAndPlay("P1Wins");
else
    g_player = 2;

++g_movesMade;
if (g_movesMade > 63)
    gotoAndPlay("Tie");
```

And in P2Move's last frame

```
if (checkIfWin(g_playedTile) == true)
    gotoAndPlay("P2Wins");
else
{
    ++g_movesMade;
    if (g_movesMade > 63)
        gotoAndPlay("Tie");
    else
    {
        g_player = 1;
        gotoAndPlay("PTurn");
    }
}
```

## Winning the Game

Now we want to create fancy winning sequences. First we will create the sequence for X winning. The O winning sequence is done the same way except that the x's and O's are switched.

The animation consists of 10 layers. The four guide layers, the four layers for the X's, the layer for the losing O, and the code layer. The X layers are named Cyclops, Wolverine, Rogue, and Storm as an inside joke. If you don't understand the reference, simply go to a comic book shop and ask the clerk.

Two of the X layers (and their guides) are placed in front of the O and the other two are placed behind the O. The X's are appropriately scaled. The motion guides are designed to move the X to the next step. The first frame starts with the X's on the starting part of the guide, sized for their starting position. The ending frame (frame 20) has the x's on their ending frame, sized as if they were in that location. Between the two keyframes is a motion tween.

One last bit of code,

```
gotoAndPlay(2);
```

makes the animation play smoothly.

With the animations ready, the sequences have to be created. The animations are placed on a new layer called "Winning" on sequences labelled "P1Wins", "P2Wins", and "Tie."

On the first frame of the win sequence for P1Wins, we set the player to 3 (4 for player 2 and 5 for a tie). This will cause the game to return to the title when clicked. Of course, we don't have any title screen yet.... Now we simply have a tweened sequence of the winning screen coming into view.



**Figure 3: Winning Sequence**