

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 12

Video Poker

Contents

With a deck class in hand, we are now able to create one of the most well known game out there. Video poker is one of the most popular forms of poker where you play a five card draw hand with better hands paying out more.

- Game Layout - The layout of the game.
- Betting - Letting the player place bets.
- Revealing the Hand - Showing what cards the player has.
- Selecting the Cards - Letting the player select cards to draw.
- Drawing the Cards - Dealing new cards to replace cards set as Draw.
- Calculating the Win - Determining the results of the hand.
- Handling the Results - Winning message and adding sound effects.

Game Layout

As the game is a simulation of a video game found at many casinos it only makes sense to have a background that reflects this fact. This is actually done in two layers. The “Screenback” layer holds the screen and the “Console” layer holds the console (outer part of the machine). Figure 1 shows the “Screenback” layer, Figure 2 shows the “Console” layer and figure 3 shows all of the layers combined.



Figure 2 - Screenback Layer

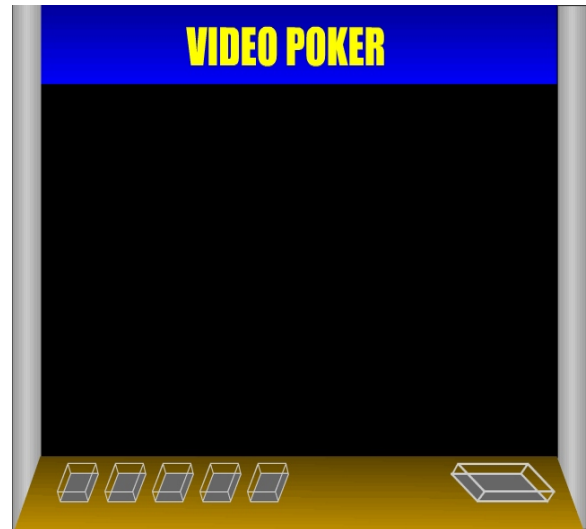


Figure 1- Console Layer

In order to actually play the game, we are going to need a few more layers. The layers used in the movie are “Code,” “Info,” “Bet,” “Console,” “FRCard,” “RCard,” “CCard,” “LCard,” “FLCard,” and “Screenback.”

The Code layer is where we place our code and labels. The “Console” and “Screenback” layers were described above. The “FRCard” layer holds the card on the far right. The “RCard” layer holds the card on the right. And so on. We simply place a card object on each of the layers. The card instances are named, “FLCard_movie,” “LCard_movie,” “CCard_movie,” “Rcard_movie,” and “FRCard_movie.” To make sure the cards are aligned vertically and spaced evenly, you can use the align submenu under the modify menu. To use this, select the objects you want to align by clicking on them while the shift key is down and then choose the alignment option of your choice.

Now, the info layer is going to contains a line of text that gives players instructions and results. Quite simply, we create this layer by creating a dynamic text object using a red 50 point “_sans” font and naming the instance “results_txt.” We also create dynamic text objects for the bet, last win and current balance. These will be created using a green 20 point “_sans” font. The instances will be named “bet_text,” “win_txt,” and “balance_txt.” Later on we will be adding Draw movies to this layer.

The bet layer is mostly blank. It only appears when the bet buttons are active. The rest of the time the player will only see the non-functioning buttons that are part of the console image. This work is stored on the disk as “/source/chapter12/videopoker_layout fla”.

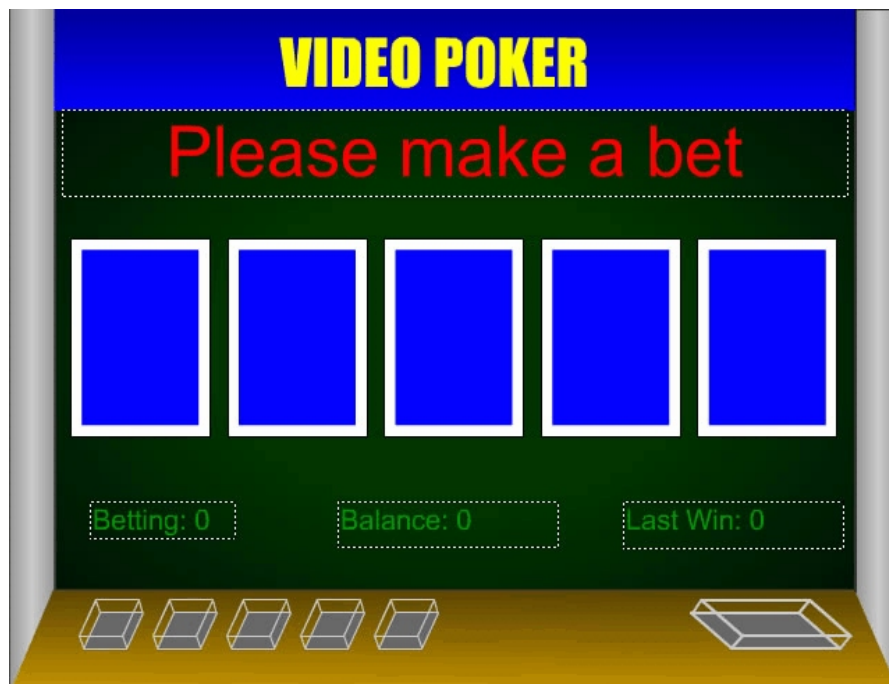


Figure 3 - Layout

Betting

Video poker, and poker in general for that matter, is a casino game. This means that betting is an important aspect of the game. In a casino, the video poker games I have seen are based around a unit of currency. That unit, be it nickles, quarters, or dollars is the amount of a bet. Most machines allow for more than one unit be bet at a time. I have decided that my virtual machine will allow for 1 to 5 units to be used.

As you seen last chapter, the background console already has buttons drawn. These are not usable. What we are going to do is create a button class that looks like the console button but lit up (a common casino convention to let the player know what options are currently selectable). This lit up button will be placed over top the console image so to the user it will appear as if the console has lit up the button. Figure 4 shows the four frames that make up one of the buttons.

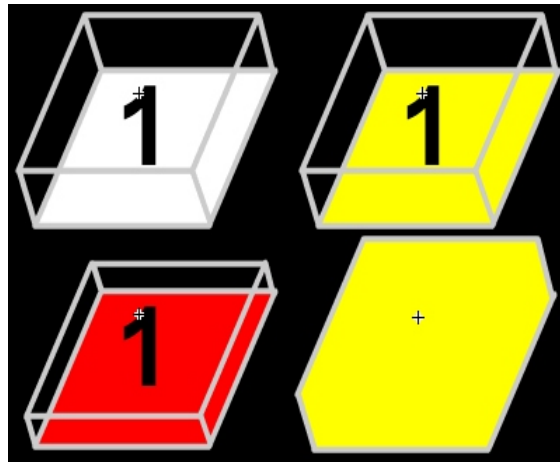


Figure 4 - Buttons

Placing the buttons over their corresponding console button is simply a matter of zooming in the image and using the cursor keys to finely position the buttons. The button instance names are labeled “bet1_btn,” “bet2_btn,” “bet3_btn,” “bet4_btn” and “bet5_btn.” Now we are ready to start adding code.

In the first frame, we want to make sure that we import our deck class. For that reason the following line of code is used. Note that you should make sure that the file is in that directory. If not, you can provide a more detailed path to the location of the file.

```
#include "Deck.as"
```

The second frame of the “Code” layer is where we place our initialization. The initialization is designed to run only once. We also have a function that sets the four dynamic text variables to the values contained within game variables.

```
initGame();

function initGame()
{
    if (bjInitialized != undefined)
        return;

    bjInitialized = true;
    selecting = false;
    bet = 0;
    cash = 0;
    lastWin = 0;
    message = "?";
    updateText();
    hand = new Array(FLCard_movie, LCard_movie, CCard_movie, RCard_movie,
FRCard_movie);
}

function updateText()
{
    results_text.text = message;
    bet_text.text = "Betting: " + bet;
    balance_text.text = "Balance: " + cash;
    win_text.text = "Last Win: " + lastWin;
}
```

The third frame of the code layer we label “BetWait.” This will be the location the movie moves to whenever the player has to make another bet. To handle the betting we simply need to make sure that our message text is changed to the appropriate line. We then activate the buttons and get the button’s onRelease function to set the bet variable with the amount the button is worth and then make the movie go to a label called “ClearBets”, which we will be creating shortly.

```
message = "Please make a bet...";
updateText();
stop();

bet1_btn.onRelease = function() {
    bet = 1;
    gotoAndPlay("ClearCards");
}

bet2_btn.onRelease = function() {
    bet = 2;
    gotoAndPlay("ClearCards");
}

bet3_btn.onRelease = function() {
    bet = 3;
    gotoAndPlay("ClearCards");
}

bet4_btn.onRelease = function() {
    bet = 4;
    gotoAndPlay("ClearCards");
}

bet5_btn.onRelease = function() {
    bet = 5;
    gotoAndPlay("ClearCards");
}
```

Now, on the fifth frame (though technically, the specific frame doesn’t matter), we create a keyframe in the code layer and label it “ClearCards.” For the purpose of debugging the game, we place a stop() in this frame. The “Bet” layer should have a clear keyframe placed here. For testing we will write the words “BET MADE!” in this frame so we know the bet buttons work. All the other layers should be extended to this frame. I do this by right clicking on the frame I want to extend to and selecting “create frame” from the pulldown menu. You could also click on the frame and then select “frame” from the “insert” menu. You could also have used the keyboard shortcut.

Revealing the Hand

Now that we have verified that the bet buttons are functioning, we can clear the bet layer's keyframe on the "ClearCards" frame. We can also add the following code to the "Code" layer of the "ClearCards" frame. This code will clear the message and remove the bet from the players current cash value.

```
message = " ";  
cash -= bet;  
updateText();
```

The revealing of the initial drawing of cards is done in three phases. The first phase the cards are removed from the game play field. This is simply a matter of moving the cards, each card separated by a few frames, from it's current position to a position along the top of the screen while changing the size of the card to 20%. This is done in the movie section labeled "ClearCards." I placed the moves in frames 5, 8, 11, 14, and 17.

The second phase is dealing the cards. To do this, we need an instance of the Deck class. For this reason, we add the following lines to the end of the initGame() function on frame 2. The first line creates an instance of the deck class. The second line creates an array that holds information about which cards are going to be drawn. While we don't need this right now, we are going to need it eventually so we might as well deal with it now.

```
myDeck = new Deck();  
drawList = new Array(false, false, false, false, false);
```

Now, on Frame 20 of the code layer we create a keyframe labeled “Deal.” We also add the following code to the frame. This code goes through the five cards in the hand (in the initialization section of the movie, all of the card movies were put into an array named hand). We set the value of the card, make sure the card face is showing, and make sure the selected value is false. The selected value isn’t used yet, but will be needed for drawing cards.

```
var cntr;  
  
myDeck.shuffle();  
for (cntr = 0; cntr < 5; ++cntr)  
{  
    hand[cntr].setCard(myDeck.draw());  
    hand[cntr].showFace();  
    drawList[cntr] = false;  
}
```

Finally we animate the cards going from their top position to their proper final locations. What I do is stagger the starting frame of the reveal animation by five frames, yet have the reveal animation take 10 frames. In other words, we have starting frames of 20, 25, 30, 35 and 40. Each card moves from the hidden spot at the top to it’s proper position and at the same time returns to it’s normal size. This is a simple motion tween for all five of the cards.

All of the layers should be extended to frame fifty. To test the changes we have made, we create a keyframe in the code layer of frame fifty and label the frame “Select”. It is being called select because the next thing that happens in the game is that the player selects which cards they want to draw. Place a stop(); in the code. Now on the bet layer on the “Select” frame create a blank keyframe. Within this frame place the message “Deal Done”.

Selecting the Cards

Now the player is in a position to select cards to use. We need a way of letting the player know that the card is selected so we will create a draw movie. This movie is a single frame movie. Why a movie? Quite simple, we need to be able to hide the movie, and for that we need to use a movie clip. This movie simply contains the word “DRAW” in bold red letters.

Now, we place the draw movies above each of the cards. I am placing the movies in the “Info” layer on the first frame. The movie clips will be labeled, “draw1_movie,” “draw2_movie,” “draw3_movie,” “draw4_movie,” and “draw5_movie.” We also need to make them initially invisible and we want to place them in an array so we can easily access them. To do this, we add the following lines of code to the initGame function on frame 2 of our “Code” layer.

```
drawMovie = new Array(draw1_movie, draw2_movie, draw3_movie, draw4_movie,
draw5_movie);
for (cntr = 0; cntr < 5; ++cntr)
{
    drawMovie[cntr]._visible = false;
}
```

Now, we need a way of handling the mouse clicking. This is done in two parts. First, we have the mouse support code, which we will place at the top of frame 2 in the “Code” layer. This simply sets the root movie’s “onMouseDown” function to our own mouse handling function. This function checks to see if we are currently selecting cards. If we are, it loops through all five of the card movies and checks to see if the mouse is over the card. If the mouse is over the card, it toggles the value in that card’s drawList array entry. The drawList is used to determine which cards are to be drawn. It also sets the movie visibility of the appropriate draw movie to the appropriate state.

```
onMouseDown = function()
{
    if (selecting != true)
        return;
    var cntr;
    for (cntr = 0; cntr < 5; ++cntr)
    {
        if (hand[cntr].hitTest(_xmouse, _ymouse, false))
        {
            drawList[cntr] = !drawList[cntr];
            drawMovie[cntr]._visible = drawList[cntr];
        }
    }
}
```

Next we are going to need a way for the player to indicate that they are ready to draw cards. To do this we will need to create the draw button. This button is created the same way we created the five bet buttons. An instance of this button is placed in the bet layer (replacing the test message we had in that frame for testing our last code section. We name this button instance “draw_btn.”

Now, to activate the draw button and make sure the mouse button handling will work we add the following code to the “select” frame of the “Code” layer. This sets the mouse handling select flag. It then sets the draw button to go to a frame labeled “Draw_1.”

```
selecting = true;
draw_btn.onRelease = function()
{
    gotoAndPlay("Draw_1");
}
stop();
```

As we have done in all the other sections, we will test the modification we have made. While you probably don’t need to test as often as shown, it doesn’t hurt. This is simply a matter of adding a keyframe on the “Code” layer on frame 55 and labeling this frame “Draw_1.” We will place a stop here. All the layers should be extended to frame 55 and on the bet frame we will add a blank keyframe. To this keyframe we will write the text “Ready to Draw!”

Drawing the Cards

This has to be done on a per card basis so that we can allow Flash to do the animation. The code to handle the drawing of the cards is fairly similar for all the cards. For this reason we can create a simple function to handle the drawing. We will place this function in frame 2 of the “Code” layer of the movie, where we have all our other functions. As you can see, this function simply removes the draw state, hides the draw movie, and then sets the card’s value to the next card in the deck.

```
function drawCard(slot)
{
    drawList[slot] = false;
    drawMovie[slot]._visible = false;
    hand[slot].setCard(myDeck.draw());
}
```

The draw animation consists of the card being removed followed by the new card being moved into place. This is simply a motion tween going from the full sized card to the small card hidden at the bottom of the screen. This is followed by a small card at the top of the screen motion tweened to a full size card. For the “FLCard” layer the removal tween is on frames 56 to 65 and the drawing tween is on frames 66 to 74. The “Lcard” layer has the removal tween on frames 76 to 85 and the drawing tween is on frames 86 to 94. For the “CCard” layer the removal tween is on frames 96 to 105 and the drawing tween is on frames 106 to 114. The “Rcard” layer has the removal tween on frames 116 to 125 and the drawing tween is on frames 126 to 134. Finally, the “FRCard” layer has the removal tween on frames 136 to 144 and the drawing tween on frames 146 to 154.

Right now all the cards are being drawn, and the value of the drawn card is the same value that the card already was. To fix these problems we are going to need to add some action script as well as a few more labels. Let us start by replacing the code on the “Code” layer of the frame labelled “Draw_1.” As you can see, we first make sure the mouse up handler knows that we are no longer allowing cards to be selected. We then see if the first card (array element 0) has been selected to be drawn. If it has not been selected to be drawn we skip over the drawing animation for the first card.

```
selecting = false;
if (drawList[0] == false)
    gotoAndPlay("Draw_2");
```

Once the card has been removed from the screen, we can then draw a card. This is done by calling the function we created earlier in this chapter. Notice that we use the array index of the card in the function, which is one less than the card that we are on.

```
drawCard(0);
```

To handle the second card, we create a label on frame 75 named “Draw_2.” We then add the following code, which is similar to that found on the “Draw_1” frame. The reason the selecting variable isn’t cleared here like it was earlier is simply because there is no need to. The variable has been cleared at the “Draw_1” frame.

```
if (drawList[1] == false)
    gotoAndPlay("Draw_3");
```

Again, we need code to actually do the drawing, which is again placed after the card has been removed.

```
drawCard(1);
```

The third card is handled on frame 95 and is labelled “Draw_3.” The code is fairly obvious.

```
if (drawList[2] == false)
    gotoAndPlay("Draw_4");
```

And the code placed on a keyframe created after the card has been removed.

```
drawCard(2);
```

The third card is handled on frame 95 and is labelled “Draw_3.” The code is fairly obvious.

```
if (drawList[3] == false)
    gotoAndPlay("Draw_5");
```

And the code placed on a keyframe created after the card has been removed.

```
drawCard(3);
```

The final card is handled on frame 135 and is labelled “Draw_3.” The code is fairly obvious.

```
if (drawList[4] == false)
    gotoAndPlay("Results");
```

And the code placed on a keyframe created after the card has been removed.

```
drawCard(4);
```

Finally we extend all layers to frame 155, label this frame “Results” and place a stop(); in this frame. On the bet layer we write a simple message so we know we have reached the frame.

Calculating the Win

The results of the hand can now be calculated. While this work is all done in a single function, the work is kind of complicated. For that reason, I will be breaking the function into a series of sections so I can better explain the logic behind the function.

The first thing that is done is the cards are sorted by their face values. Face values are calculated by using the modulus (remainder) of dividing the deck by 13. To make this work, the card ID is reduced by 1.

```
function findResults()
{
    var sortedCards = new Array(5);
    var cntrl, cntr2, temp, low, testA, testB;

    // sort, by face value
    for (cntrl = 0; cntrl < 4; ++cntrl)
    {
        low = cntrl;
        for (cntr2 = cntrl + 1; cntr2 < 5; ++cntr2)
        {
            testA = (sortedCards[low] - 1) % 13;
            testB = (sortedCards[cntr2] - 1) % 13;
            if (testB < testA)
                low = cntr2;
        }
        temp = sortedCards[low];
        sortedCards[low] = sortedCards[cntrl];
        sortedCards[cntrl] = temp;
    }
}
```

Next, we prepare all the variables that we are going to need to find the results.

```
var pairFace = 0;
var isStreight = true;
var isFlush = true;
var longestRun = 0;
var numRuns = 0;
var targetSuit = Math.floor((sortedCards[0] - 1) / 13);
var curRun = 1;
```

With the variables set, we can now determine if the results of the hand is a straight, a flush, or both as well as checking how many groups of cards there are. To be a flush, all suits must be the same, so we simply check to see that the suit of the current card in the loop is the same as that of the first card. To see if we have a pair or better, we track the run length of the current card by counting how many cards in a row contain the same face value. As it is possible for more than one pair, we also count how many different face cards are matching. Likewise, to handle a full house, we need to know the length of the longest run of matching cards. Straights are simply a test to make sure that the face value of each card in the sorted hand increases by one (taking into account the fact that an ace can be in two possible positions).

```
for (cntrl = 1; cntrl < 5; ++cntrl)
{
    if ( Math.floor((sortedCards[cntrl] - 1) / 13) != targetSuit)
        isFlush = false;
    testA = (sortedCards[cntrl - 1] - 1) % 13;
    testB = (sortedCards[cntrl] - 1) % 13;
    if (testA == testB)
    {
        isStreight = false;
        ++curRun;
    }
    else
    {
        if (curRun > 1)
        {
            if (curRun > longestRun)
                longestRun = curRun;
            curRun = 1;
            ++numRuns;
            pairFace = testA; // if only a pair, need to know face
value
        }
        if (((testB-testA) != 1) && ((testA != 1) || (testB != 10)))
            isStreight = false;
    }
}
if (curRun > 1)
{
    if (curRun > longestRun)
        longestRun = curRun;
    ++numRuns;
    pairFace = testA; // if only a pair, need to know face value
}
```

Next we analyse the results to determine what the hand was.

```
if ((isStreight & isFlush) == true)
{
    if (((sortedCards[4] - 1) % 13) == 12)
    {
        message = "ROYAL FLUSH!!!";
        lastWin = 250 * bet;
    } else {
        message = "Streight Flush";
        lastWin = 50 * bet;
    }
}
else if (longestRun == 4)
{
    message = "Four of a Kind";
    lastWin = 25 * bet;
}
else if (longestRun == 3)
{
    if (numRuns == 2)
    {
        message = "Full House";
        lastWin = 9 * bet;
    } else {
        message = "Three of a Kind";
        lastWin = 3 * bet;
    }
}
else if (isFlush)
{
    message = "Flush";
    lastWin = 5 * bet;
}
else if (isStreight)
{
    message = "Streight";
    lastWin = 4 * bet;
}
else if (numRuns == 2)
{
    message = "Two Pairs";
    lastWin = 2 * bet;
}
else if ( (numRuns == 1) && ((pairFace == 0) || (pairFace > 9)) )
{
    message = "Jacks or Better";
    lastWin = bet;
}
else
{
    message = "Nothing";
    lastWin = 0;
}
cash += lastWin;
return (lastWin > 0);
```

```
}
```

Handling the Results

In the results frame we have the following code. This code goes to the bet loop immediately if the result of the hand wasn't a winning hand, otherwise the movie continues playing to give us time to show the results.

```
var didWin = findResults();  
updateText();  
if (didWin == false)  
    gotoAndPlay("BetWait");
```

Finally we skip over 30 frames (so the result message will be shown for a few seconds) and we then place our final "Code" layer keyframe, which simply contains one line of code which goes back to the "betWait" frame.

```
gotoAndPlay("BetWait");
```

Now we can add some sound. I imported three sounds: "bell.wav," "draw.wav," and "shuffle.wav". A new "sound" layer is then created. I can then add sounds to the game.

The shuffle sound is only needed when the cards are being dealt so I place it on frame 5. The draw sound is used every time a card is drawn. As a result it is used on ten different frames. The frames I placed it on are 20, 25, 30, 35, 40, 65, 85, 105, 125, and 145. Finally, the bell sound is used to indicate that the player has won money so it is placed on frame 156 (which only is reached if the results of a hand is a win).