Programming Cookbook III

"The DarkBasic Professional / Dark GDK Years"

By

Nicholas Kingsley

Introduction

Welcome to the second part of my “Programming Cookbook” series.

This version of the book contains my Dark GDK code, almost all of which wont be in the first part of the series!

Hopefully you will find the code useful, and if not, then hope it will give you some fresh ideas!

With each routine, I have included the C/C++ code, headers and if needed DarkBasic Professional constants, for a lot of this code was used to generate modules for that programming language

Shapies

This was a game written during one of my Conventions and never actually finished.

```
#include "DarkSDK.h"
#include "FindFree.h"
#include "Deletes.h"
#include "Timer.h"
#include "CFade.h"
#include "stdio.h"

CFade::CFade()
{
	image=0;
	sprite=0;
}

CFade::~CFade()
{
	DELETE_IMAGE(image);
	DELETE_SPRITE(sprite);
}

void CFade::initialise(DWORD width,DWORD height)
{
register DWORD bitmap;

	bitmap=findFreeBitmap();
	image=findFreeImage();
	sprite=findFreeSprite();

	dbCreateBitmap(bitmap,width+1,height+1);
	dbSetCurrentBitmap(bitmap);

	dbCLS(dbRgb(1,1,1));
	dbGetImage(image,0,0,width,height,1);
	dbSetCurrentBitmap(0);
	DELETE_BITMAP(bitmap);

	dbSprite(sprite,0,0,image);
	dbSetSpritePriority(sprite,2);
	dbSetSprite(sprite,0,1);
	dbHideSprite(sprite);
}

void CFade::hide(void)
{
	dbHideSprite(sprite);
}

void CFade::fade(bool up,struct __GAMESPEED *gameSpeed,void (*routine)(double speed))
{
double alpha,speed;

	alpha=(up ? 0.0 : 255.0);
	dbSetSpriteAlpha((int) sprite,(int) alpha);
	dbShowSprite(sprite);
	speed=updateTimer(gameSpeed);
	while ((up ? alpha<=255.0 : alpha>=0.0))
	{
		dbText(0,200,dbStr((float) alpha));
		dbSetSpriteAlpha((int) sprite,(int) alpha);

		if (routine)
		{
			(routine)(speed);
		}
		alpha+=(up ? 1.0 :-1.0)*speed*10.0;
		speed=updateTimer(gameSpeed);
		dbSync();
	}
}

#include "windows.h"
#include "Shapies.h"
#include "DarkSDK.h"
#include "GetDisplay.h"
#include "stdio.h"
#include "stdlib.h"
#include "io.h"

void setupControlMethods(struct __CONTROLMETHOD *controlMethod,DWORD player)
{
	// First, read in P1 keyboard controls
```

```
controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVELEFT]=GetPrivateProfileInt((player==P
LAYER1 ? CONTROL_SETTING1KEYBOARD : CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVELEFT,
(player==PLAYER1 ? 30 : 203),SETTING_FILE);

controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVERIGHT]=GetPrivateProfileInt((player==
PLAYER1 ? CONTROL_SETTING1KEYBOARD : CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVERIGHT,
(player==PLAYER1 ? 31 : 205),SETTING_FILE);

controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVEUP]=GetPrivateProfileInt((player==PLA
YER1 ? CONTROL_SETTING1KEYBOARD : CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEUP,
(player==PLAYER1 ? 18 : 200),SETTING_FILE);

controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVEDOWN]=GetPrivateProfileInt((player==P
LAYER1 ? CONTROL_SETTING1KEYBOARD : CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEDOWN,
(player==PLAYER1 ? 32 : 208),SETTING_FILE);

controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_ROTATE]=GetPrivateProfileInt((player==PLA
YER1 ? CONTROL_SETTING1KEYBOARD : CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEROTATE,
(player==PLAYER1 ? 44 : 54),SETTING_FILE);
}

void writeControlMethods(struct __CONTROLMETHOD *controlMethod,DWORD player)
{
char temp[6];

	SecureZeroMemory(&temp,sizeof(temp));
	sprintf(temp,"%03ld",controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVELEFT]);
	WritePrivateProfileString((player==PLAYER1 ? CONTROL_SETTING1KEYBOARD :
CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVELEFT,(char *) &temp,SETTING_FILE);
	sprintf(temp,"%03ld",controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVERIGHT]);
	WritePrivateProfileString((player==PLAYER1 ? CONTROL_SETTING1KEYBOARD :
CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVERIGHT,(char *) &temp,SETTING_FILE);
	sprintf(temp,"%03ld",controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVEUP]);
	WritePrivateProfileString((player==PLAYER1 ? CONTROL_SETTING1KEYBOARD :
CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEUP,(char *) &temp,SETTING_FILE);
	sprintf(temp,"%03ld",controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_MOVEDOWN]);
	WritePrivateProfileString((player==PLAYER1 ? CONTROL_SETTING1KEYBOARD :
CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEDOWN,(char *) &temp,SETTING_FILE);
	sprintf(temp,"%03ld",controlMethod[CONTROL_KEYBOARDINDEX].control[CONTROL_INDEX_ROTATE]);
	WritePrivateProfileString((player==PLAYER1 ? CONTROL_SETTING1KEYBOARD :
CONTROL_SETTING2KEYBOARD),CONTROL_INDEX_TEXT_MOVEROTATE,(char *) &temp,SETTING_FILE);


}

#include "DarkSDK.h"
#include "CPointer.h"
#include "Deletes.h"
#include "FindFree.h"
#include "Shapies.h"
#include "stdio.h"
#include "stdlib.h"

CPointer::CPointer()
{
	sprite=0;
}

CPointer::~CPointer()
{
	DELETE_SPRITE(sprite);
}

void CPointer::initialise(void)
{
	sprite=findFreeSprite();
	scale=100.0;
	dir=1.0;
	time=100.0;
	dbSprite(sprite,dbMouseX(),dbMouseY(),IMAGE_ARROW);
	dbOffsetSprite(sprite,0,0);
	dbSetSprite(sprite,0,1);
	dbSetSpritePriority(sprite,4);
	hide();
}

void CPointer::moveMousePointer(double speed)
{
	dbScrollObjectTexture(OBJECT_BACKDROP,(float) 0.01*(float) speed,(float) 0.01*(float)
speed);

	if (dbMouseMoveX()==0 && dbMouseMoveY()==0)
	{
		if (time>0.0)
		{
			time-=(double) 10.0*speed;
			if (time<=0.0)
			{
				dir=1.0;
				time=0.0;
			}
		}
		else
		{
			scale+=dir*speed*10.0;
			if (scale<=100.0 || scale>=500.0)
			{
				dir=0.0-dir;
			}
		}
	}
	else
	{
		// Put sprite back to original size
		time=1000.0;
		scale=100.0;
	}

	px=dbMouseX();
	py=dbMouseY();
	dbSprite(sprite,px,py,IMAGE_ARROW);
	dbScaleSprite(sprite,(float) scale);
}

void CPointer::hide(void)
{
	dbHideSprite(sprite);
}

void CPointer::show(void)
{
	dbShowSprite(sprite);
}

void CPointer::getXYWidthHeight(DWORD sprite,int *x,int *y,int *dx,int *dy)
{
	*(x)=dbSpriteX(sprite);
	*(y)=dbSpriteY(sprite);
	*(dx)=*(x)+dbSpriteWidth(sprite);
	*(dy)=*(y)+dbSpriteHeight(sprite);
}

bool CPointer::getContinueCollision(void)
{
int cx,cy,cw,ch;

	getXYWidthHeight(SPRITE_CONTINUE,&cx,&cy,&cw,&ch);
	return ((dbMouseClick()==1 && (px>=cx && px<=cw) && (py>=cy && py<=ch)) ? true : false);
}

int CPointer::getTopMenuCollision(struct __MENUITEMS *menuItem)
{
register DWORD loop,hit,loop2;
int mx,my,mh,mw;

	loop=0;
	while (menuItem[loop].menuItem.sprite)
	{
		if (dbSpriteVisible(menuItem[loop].menuItem.sprite))
		{
			getXYWidthHeight(menuItem[loop].menuItem.sprite,&mx,&my,&mw,&mh);

			if ((px>=mx && px<=mw) &&
				(py>=my && py<=mh))
			{
				return (menuItem[loop].menuItem.sprite);
			}
		}

		loop2=0;
		while (menuItem[loop].subMenuItems[loop2].sprite)
		{
			if (dbSpriteVisible(menuItem[loop].subMenuItems[loop2].sprite))
			{
	getXYWidthHeight(menuItem[loop].subMenuItems[loop2].sprite,&mx,&my,&mw,&mh);
				if ((px>=mx && px<=mw) &&
					(py>=my && py<=mh))
				{
					return (menuItem[loop].subMenuItems[loop2].sprite);
				}
			}

			loop2++;
		}

		loop++;
	}

	return NO_COLLISION;
}
```

```
#include "DarkSDK.h"
#include "Shapies.h"
#include "FindFree.h"
#include "stdio.h"
#include "Timer.h"
#include "Deletes.h"
#include "CPointer.h"
#include "CFade.h"

void displayHideExtraMenu(DWORD index,bool show);
void showHideMenu(bool show);
void showHideTitle(bool show);

extern struct __MENUITEMS     menuItems[MAX_MENUITEMS];
extern struct __BALL          balls[MAX_BALLS];
extern CPointer                               mousePointer;
extern CFade                          fade;

DWORD menu(struct __GAMESPEED *gameSpeed)
{
register DWORD loop;
int                           result;
DWORD                 objState;
bool                  show[MAX_MENUITEMS];
bool                  canClick;
double                dir;
double                time,speed,scale,cameraXPos,cameraYPos,cameraZPos,objectXAngle,objectZAngle;

        dbFlushVideoMemory();
        dbCLS(0);

        showHideMenu(true);

        SecureZeroMemory(&show,sizeof(show));

        canClick=true;
        result=NO_COLLISION;
        time=1000.0;
        dir=0.0;
        scale=100.0;

        cameraXPos=0.0;
        cameraYPos=-0.5;
        cameraZPos=-100.0;

        objectXAngle=0.0;
        objectZAngle=0.0;

        dbRotateObject(OBJECT_TITLE,(float) objectXAngle,(float) 0.0,(float) objectZAngle);
        dbPositionCamera((float) cameraXPos,(float) cameraYPos,(float) cameraZPos);

        showHideTitle(true);

        mousePointer.show();

        speed=updateTimer(gameSpeed);
        speed=updateTimer(gameSpeed);

        while (LoopSDK() && result==NO_COLLISION) {
                if (dbEscapeKey())
                {
                        result=menuItems[MENU_INDEX_QUITGAME].menuItem.sprite;
                        break;
                }

                if (cameraZPos<-5.0)
                {
                        cameraZPos+=10.0*speed;
                        cameraXPos-=0.*speed;
                        if (cameraZPos>=-5.0)
                        {
                                objState=OBJSTATE_ROTATEX;
                        }
                }
                else
                {
                        switch (objState) {
                                case    OBJSTATE_ROTATEX        :       // Rotate to so that the text can
be seen

        objectXAngle+=3.0*speed;
                                                                                                    if
(objectXAngle>=90.0)
                                                                                                    {

        objectXAngle=90.0;

        objState=OBJSTATE_ROTATEZ;
                                                                                                    }

                                                                                                    break;
```

```
                              case    OBJSTATE_ROTATEZ       :       // Rotate the text around

       objectZAngle=(double) dbWrapValue((float) objectZAngle+((float) 3.0*(float) speed));
                                                                                 break;
                              };

                       dbRotateObject(OBJECT_TITLE,(float) objectXAngle,(float) objectZAngle,(float)
0.0);
                       //dbPointCamera(0.0,0.0,0.0);

                }

                dbPositionCamera((float) cameraXPos,(float) cameraYPos,(float) cameraZPos);

                for (loop=0; loop<MAX_BALLS; loop++)
                {
                       balls[loop].y-=(balls[loop].speed*speed);
                       dbPositionObject((int) balls[loop].obj,(float) balls[loop].x,(float)
balls[loop].y,(float) balls[loop].z);
                       if (balls[loop].y<-100.0)
                       {
                              randomBall((struct __BALL *) &balls[loop]);
                       }
                }

                mousePointer.moveMousePointer(speed);

                switch (dbMouseClick()) {
                       case    0      :      canClick=true;
                                                    break;
                       case    1      :      // LMB
                                                    if (canClick==true)
                                                    {

       result=mousePointer.getTopMenuCollision(&menuItems[0]);
                                                           if (result!=NO_COLLISION)
                                                           {
                                                                  if
(result==menuItems[MENU_INDEX_SINGLEPLAYER].menuItem.sprite)
                                                                  {

                                                                         if
(show[MENU_INDEX_MULTIPLAYER])
                                                                         {

       show[MENU_INDEX_MULTIPLAYER]=false;

       displayHideExtraMenu(MENU_INDEX_MULTIPLAYER,show[MENU_INDEX_MULTIPLAYER]);
                                                                         }


       show[MENU_INDEX_SINGLEPLAYER]=!show[MENU_INDEX_SINGLEPLAYER];

       displayHideExtraMenu(MENU_INDEX_SINGLEPLAYER,show[MENU_INDEX_SINGLEPLAYER]);
                                                                         result=0;
                                                                  }
                                                                  else
                                                                  if
(result==menuItems[MENU_INDEX_MULTIPLAYER].menuItem.sprite)
                                                                  {
                                                                         if
(show[MENU_INDEX_SINGLEPLAYER])
                                                                         {

       show[MENU_INDEX_SINGLEPLAYER]=false;

       displayHideExtraMenu(MENU_INDEX_SINGLEPLAYER,show[MENU_INDEX_SINGLEPLAYER]);
                                                                         }


       show[MENU_INDEX_MULTIPLAYER]=!show[MENU_INDEX_MULTIPLAYER];

       displayHideExtraMenu(MENU_INDEX_MULTIPLAYER,show[MENU_INDEX_MULTIPLAYER]);
                                                                         result=NO_COLLISION;
                                                                  }
                                                                  else
                                                                  if
(result==menuItems[MENU_INDEX_HISCORETABLE].menuItem.sprite)
                                                                  {

       fade.fade(true,gameSpeed,NULL);
                                                                  }
                                                           }

                                                           canClick=false;
                                                    }
                                                    break;

                       };

                dbSync();
                speed=updateTimer(gameSpeed);
        };

        showHideMenu(false);
        showHideTitle(false);
        mousePointer.hide();

        return result;
}

void showHideTitle(bool show)
{
register DWORD loop;

        for (loop=0; loop<MAX_BALLS; loop++)
        {
                if (show)
                {
                        dbShowObject(balls[loop].obj);
                }
                else
                {
                        dbHideObject(balls[loop].obj);
                }
        }

        if (show)
        {
                dbShowObject(OBJECT_TITLE);
        }
        else
        {
                dbHideObject(OBJECT_TITLE);
        }
}

void showHideMenu(bool show)
{
register DWORD loop;

        loop=0;
        while (menuItems[loop].menuItem.sprite)
        {
                if (show)
                {
                        dbShowSprite(menuItems[loop].menuItem.sprite);
                }
                else
                {
                        dbHideSprite(menuItems[loop].menuItem.sprite);
                        displayHideExtraMenu(loop,false);
                }

                loop++;
        }
}

void displayHideExtraMenu(DWORD index,bool show)
{
register DWORD loop;

        loop=0;
        while (menuItems[index].subMenuItems[loop].sprite)
        {
                if (show)
                {
                        dbShowSprite(menuItems[index].subMenuItems[loop].sprite);
                }
                else
                {
                        dbHideSprite(menuItems[index].subMenuItems[loop].sprite);
                }

                loop++;
        }
}

#include "Player.hpp"
#include "DarkSDK.h"
#include "Shapies.h"
#include "FindFree.h"
#include "Timer.h"
#include "stdio.h"
#include "Deletes.h"
#include "LinkList.h"
#include "math.h"
#include "CBFont.hpp"
#include "..\Resource\Resource.h"

extern struct __BORDERINFO borderInfo;
extern DWORD m_gameShapeWidth;
extern DWORD m_halfSelectorWidth,m_halfSelectorHeight;
extern struct __CONTROLMETHOD controlMethod[MAX_PLAYERS][3];

CPlayer::CPlayer()
{
register DWORD x;

	SecureZeroMemory(&control,sizeof(control));
	SecureZeroMemory(&offset,sizeof(offset));
	SecureZeroMemory(&rotate,sizeof(rotate));
	SecureZeroMemory(&halfColumn,sizeof(halfColumn));
	SecureZeroMemory(&movedUp,sizeof(movedUp));
	SecureZeroMemory(&playerInfo,sizeof(playerInfo));
	SecureZeroMemory(&multiplier,sizeof(multiplier));
	SecureZeroMemory(&endOfGame,sizeof(endOfGame));

	for (x=0; x<GRID_WIDTH; x++)
	{
		halfColumn[x].state=STATE_NONE;
	}

	column.state=STATE_NONE;
	m_numMatching=0;
	gameState=STATE_NONE;
	header=NULL;
	finalSprite=0;
	borderSprite=0;
}

CPlayer::~CPlayer()
{
register DWORD x,y;

	for (x=0; x<GRID_WIDTH; x++)
	{
		for (y=0; y<GRID_HEIGHT; y++)
		{
			if (grid[x][y].image!=NO_OBJECT)
			{
				DELETE_SPRITE(grid[x][y].sprite);
			}
		}
	}

	DELETE_SPRITE(finalSprite);
	DELETE_SPRITE(borderSprite);
	DELETE_SPRITE(multiplier.sprite);
	DELETE_SPRITE(control.sprite);
	DELETE_SPRITE(endOfGame.sprite);
}

void CPlayer::clearGrid(DWORD level)
{
register DWORD x,y;

	SecureZeroMemory(&grid,sizeof(grid));
	for (y=0; y<GRID_HEIGHT; y++)
	{
		for (x=0; x<GRID_WIDTH; x++)
		{
			grid[x][y].sprite=NO_OBJECT;
			grid[x][y].image=NO_OBJECT;
			grid[x][y].state=STATE_NONE;
		}
	}

	playerInfo.score=0;
	playerInfo.level=level;
	playerInfo.timeToNextExtraAdd=newExtraTime();
	playerInfo.numItemsToBeAdded=0;
	playerInfo.itemsToNextLevel=10;
	playerInfo.playerState=STATE_NONE;
	playerInfo.countdownTimer=0.0;

	gameState=STATE_NONE;


}

void CPlayer::UpdateMultiplier(void)
{
char temp[MULTIPLIER_WIDTH+1];

	SecureZeroMemory(&temp,sizeof(temp));
	sprintf((char *) &temp,"%02ld",multiplier.total);
	playerMultiplier.setText((char *) &temp);
}

void CPlayer::UpdateTimer(void)
{
char temp[TIMER_WIDTH+1];
```

```
        SecureZeroMemory(&temp,sizeof(temp));
        if (playerInfo.playerState==STATE_NONE)
        {
               memset((char *) &temp,(char) '-',sizeof(temp));
        }
        else
        {
               sprintf(temp,"%02ld",(long) playerInfo.countdownTimer);
        }

        playerCountDown.setText((char *) &temp);
}

void CPlayer::UpdateLevel(void)
{
char temp[LEVEL_WIDTH+1];

        SecureZeroMemory(&temp,sizeof(temp));
        sprintf((char *) &temp,"%03ld",playerInfo.level);
        playerLevel.setText((char *) &temp);
}

void CPlayer::UpdateScore(void)
{
char temp[SCORE_WIDTH+1];

        SecureZeroMemory(&temp,sizeof(temp));
        sprintf(temp,"%06ld",playerInfo.score);
        playerScore.setText((char *) &temp);
}

void CPlayer::ShowBorder(void)
{
        //dbSprite(borderSprite,offset.x-3,offset.y-3,IMAGE_BORDER);
        dbShowSprite(borderSprite);
}

// Set display offset
void CPlayer::setOffset(DWORD x,DWORD y,DWORD fontSet,DWORD controlMethod,DWORD player,bool
placeOnRight)
{
float   scorePos,yPos,xPos;
DWORD textXSize,textYSize;

        borderSprite=findFreeSprite();
        dbSprite(borderSprite,x,y,IMAGE_BORDER);
        dbSetSprite(borderSprite,0,1);
        dbHideSprite(borderSprite);

        offset.x=x+3;
        offset.y=y+3;
        offset.fontSet=fontSet;

        playerControls=controlMethod;
        m_player=player;

        header=Create_Header();
        if (header==NULL)
        {
               MessageBox(NULL,"Header Errro!","*",MB_OK);
        }

        scorePos=(float) (borderInfo.width-borderInfo.scoreWidth-15);
        yPos=(float) ((offset.y+borderInfo.height)-(BLOCK_SIZE>>1)-5);

        if (playerScore.loadFontFromConfig((player==PLAYER1 ? FONT_PATH "P1GAMEFONT.INI" : FONT_PATH
"P2GAMEFONT.INI"),FONT_PATH)==false)
        {
               MessageBox(NULL,"Setting up player score error","*",MB_OK);
        }

        // Now to scale

        imagePtr=playerScore.returnImagePtr();

        playerScore.returnTextSizes(&textXSize,&textYSize);
        playerScore.setCharSet((float) 0.7,(float) 0.7);
        playerScore.setTextSizes((DWORD) textXSize,(DWORD) textYSize);
        playerScore.createText("000000",(float) (offset.x+scorePos),(float) (offset.y-
(BLOCK_SIZE>>1)-3),true);

        playerCountDown.useFontFromElsewhere(imagePtr);
        playerCountDown.setCharSet((float) 0.7,(float) 0.7);
        playerCountDown.setTextSizes((DWORD) textXSize,(DWORD) textYSize);
        playerCountDown.createText("00",(float) (offset.x+5),yPos,true);

        playerLevel.useFontFromElsewhere(imagePtr);
        playerLevel.setCharSet((float) 0.7,(float) 0.7);
        playerLevel.setTextSizes((DWORD) textXSize,(DWORD) textYSize);
        playerLevel.createText("00",(float) ((offset.x+borderInfo.width+2)-
(TIMER_WIDTH*(textXSize+1))),yPos,true);

        multiplier.image=dbRnd(MAX_SHAPEGRAPHICS-1)+IMAGE_SHAPE1;
        multiplier.sprite=findFreeSprite();
        multiplier.total=0;
        control.rotateRight=placeOnRight;

        xPos=(placeOnRight ? (float) (offset.x+borderInfo.width+8) : (float) offset.x-
m_gameShapeWidth-8);
        yPos=(float) (offset.y+8);

        dbSprite(multiplier.sprite,
                        (int) xPos,
                        (int) yPos,
                        (int) multiplier.image);
        dbSetSprite((int) multiplier.sprite,0,1);

        // Create star
        xy.sprite=findFreeSprite();
        xy.x=(DWORD) ((DWORD) xPos+(dbSpriteWidth((int) multiplier.sprite)>>1));
        xy.y=(DWORD) ((DWORD) yPos+14);
        xy.step=0;
        xy.alpha=0.0;
        xy.dir=1.0;
        xy.show=false;
        dbSprite(xy.sprite,xy.x,xy.y,IMAGE_STAR);
        dbOffsetSprite(xy.sprite,dbSpriteWidth(xy.sprite)>>1,dbSpriteHeight(xy.sprite)>>1);
        dbHideSprite(xy.sprite);

        // Create multiplayer value
        playerMultiplier.useFontFromElsewhere(imagePtr);
        playerMultiplier.setCharSet((float) 0.7,(float) 0.7);
        playerMultiplier.setTextSizes(textXSize,textYSize);
        playerMultiplier.createText("00",(float) xPos,(float) (offset.y+36),true);

        // Setup final score sprite
        finalSprite=findFreeSprite();
        dbSprite(finalSprite,0,0,IMAGE_FINALSCORE);
        dbSetSpritePriority(finalSprite,4);
        dbHideSprite(finalSprite);

        // Setup end of game text
        endOfGame.sprite=findFreeSprite();
        dbSprite(endOfGame.sprite,0,0,1);
        dbSetSprite(endOfGame.sprite,0,1);
        dbSetSpritePriority(endOfGame.sprite,4);
        dbHideSprite(endOfGame.sprite);
}

void CPlayer::displaySprite(DWORD x,DWORD y)
{
        SPRITE_BASE(x,y);
        dbSetSprite(grid[x][y].sprite,0,1);
        dbSetSpritePriority(grid[x][y].sprite,1);
}

void CPlayer::SPRITE_BASE(DWORD x,DWORD y)
{
        dbSprite((int) grid[x][y].sprite,
                                (int) (offset.x+(x*SELECTORGRAPHIC_WIDTH))-GAMEGRAPHIC_OFFSETX,
                                (int) (offset.y+(y*SELECTORGRAPHIC_HEIGHT))-GAMEGRAPHIC_OFFSETY,
                                (int) grid[x][y].image);
}

// Initialise grid when starting game
void CPlayer::setupGrid(void)
{
register DWORD x,y;

        for (y=0; y<3; y++)
        {
                for (x=0; x<GRID_WIDTH; x++)
                {
                        // Do we do the player area ?
                        if (y==0)
                        {
                                AddItem(x,PLAYER_AREA);
                        }
                        else
                        {
                                AddItem(x,PLAYER_AREA-y);
                                AddItem(x,PLAYER_AREA+y);
                        }
                }
        }

        // Setup the selector now
        control.nextColumn=0; //(GRID_WIDTH>>1);
        control.y=(double) (PLAYER_AREA*SELECTORGRAPHIC_HEIGHT)+(double) ((double)
m_halfSelectorWidth*2.0);
        control.x=(double) (control.nextColumn*SELECTORGRAPHIC_WIDTH)+(double) ((double)
m_halfSelectorHeight*2.0);
        control.sprite=findFreeSprite();
        control.state=STATE_NONE;
        control.rotate=0.0;

        displaySelector();
        dbSetSprite(control.sprite,0,1);
        dbSetSpritePriority(control.sprite,1);
        dbOffsetSprite(control.sprite,m_halfSelectorWidth,m_halfSelectorHeight);

        for (x=0; x<GRID_WIDTH; x++)
        {
                if (grid[x][PLAYER_AREA].state==STATE_NONE && checkForMatchingItems(x))
                {
                }
        }
}

void CPlayer::displaySelector(void)
{
        dbSprite((int) control.sprite,
                                (int) (control.x+offset.x+m_halfSelectorWidth),
                                (int) ((control.y+offset.y)-m_halfSelectorHeight),
                                IMAGE_SELECTOR);
}

void CPlayer::rotateColumn(bool right)
{
        if (m_numMatching>=MIN_BLOCKSTOMATCH || rotate.state!=STATE_NONE) return;

        rotate.state=(right ? STATE_ROTATERIGHT : STATE_ROTATELEFT);
        rotate.moveLeft=(double) SELECTORGRAPHIC_WIDTH;
        rotate.offset=0.0;
}

// Move a column up or down
void CPlayer::moveColumn(DWORD x,bool up)
{
register int nextPlayerY;

        nextPlayerY=PLAYER_AREA+(up ? 1 : -1);
        if (rotate.state!=STATE_NONE || header->numNodes!=0)       return;

        // Is there space at the top or bottom ? Or is the row before or after the middle line empty
?
        if (grid[x][(up ? 0 : GRID_HEIGHT-1)].image!=NO_OBJECT ||
                grid[x][nextPlayerY].image==NO_OBJECT ||
                grid[x][nextPlayerY].state!=STATE_NONE ||
                m_numMatching>=MIN_BLOCKSTOMATCH ||
                rotate.state!=STATE_NONE)
        {
                return;
        }

        column.state=(up ? STATE_MOVEUP : STATE_MOVEDOWN);
        column.column=x;
        column.moveLeft=SELECTORGRAPHIC_HEIGHT;
        column.offset=0.0;
        m_numberFinishedMoving=0;
}

bool CPlayer::checkForMatchingItems(int x)
{
bool store[GRID_WIDTH];
register DWORD loop;

        SecureZeroMemory(&store,sizeof(store));

        store[x]=true;
        m_numMatching=0;

        if (x-1>=0)                  checkItems(x,-1,grid[x][PLAYER_AREA].image,(bool *) &store);
        if (x+1<GRID_WIDTH)   checkItems(x,+1,grid[x][PLAYER_AREA].image,(bool *) &store);

        for (loop=0; loop<GRID_WIDTH; loop++)
        {
                if (store[loop])
                {
                        m_numMatching++;
                }
        }
//      char t[256];

//      sprintf(t,"[%d]/[%d]/[%d]/[%d]/[%d]\n[%d]/[%d]/[%d]/[%d]",grid[0][PLAYER_AREA].image,grid[1]
[PLAYER_AREA].image,grid[2][PLAYER_AREA].image,m_numMatching,x,
//
        store[0],store[1],store[2],store[4],store[4]);
        //MessageBox(NULL,t,"*",MB_OK);

        if (m_numMatching>=MIN_BLOCKSTOMATCH)
        {
```

```
		for (loop=0; loop<GRID_WIDTH; loop++)
		{
			if (store[loop])
			{
				grid[loop][PLAYER_AREA].state=STATE_ISINUSE;
				SetUpFadeDown(loop,PLAYER_AREA);
			}
		}

		if (grid[x][PLAYER_AREA].image==multiplier.image)
		{
			multiplier.total++;
			xy.show=true;
			xy.dir=1.0;
			xy.step=0.0;
			xy.alpha=0.0;
			dbSprite(xy.sprite,xy.x,xy.y,IMAGE_STAR);
			dbShowSprite(xy.sprite);
		}
		else
		{
			multiplier.total=(multiplier.total>0 ? multiplier.total-1 : 0);
		}

		UpdateMultiplier();
	}

	return ((m_numMatching>=MIN_BLOCKSTOMATCH ? true : false));
}

void CPlayer::SetUpFadeDown(DWORD x,DWORD y)
{
struct __FADE  fade;

	fade.alpha=255.0;
	fade.image=grid[x][y].image;
	fade.sprite=grid[x][y].sprite;
	fade.state=STATE_FADEDOWN;
	fade.x=x;
	fade.y=y;
	if (Add_Node(header,(char *) &fade,sizeof(fade),NODE_ADDLAST)==NULL)
	{
	}

	//MessageBox(NULL,"Added","*",MB_OK);
}

void CPlayer::SetUpHalfScroll(DWORD x,bool up)
{
struct __FADE  fade;

	fade.state=(up ? STATE_MOVEUP : STATE_MOVEDOWN);
	fade.alpha=0.0;
	fade.x=x;

	if (Add_Node(header,(char *) &fade,sizeof(fade),NODE_ADDLAST)==NULL)
	{
	}
//	halfColumn[x].state=(up ? STATE_MOVEUP : STATE_MOVEDOWN);
//	halfColumn[x].moveLeft=(double)
//	halfColumn[x].finished=false;
//	halfColumn[x].offset=0.0;
}

void CPlayer::AddItem(DWORD x,DWORD y,DWORD sprite,DWORD image)
{
	SecureZeroMemory(&grid[x][y],sizeof(struct __GRID));
	grid[x][y].image=image;
	grid[x][y].sprite=sprite;
	grid[x][y].state=STATE_NONE;
	displaySprite(x,y);
	dbSetSpriteAlpha(sprite,255);
}

void CPlayer::AddItem(DWORD x,DWORD y,bool display)
{
struct __FADE  fade;

	grid[x][y].image=dbRnd(MAX_SHAPEGRAPHICS-1)+IMAGE_SHAPE1;
	grid[x][y].sprite=findFreeSprite();
	displaySprite(x,y);
	dbSetSpriteAlpha(grid[x][y].sprite,(display ? 255 : 0));
	if (display==false)
	{
		fade.alpha=0.0;
		fade.image=grid[x][y].image;
		fade.sprite=grid[x][y].sprite;
		fade.state=STATE_FADEUP;
		fade.x=x;
		fade.y=y;
		if (Add_Node(header,(char *) &fade,sizeof(fade),NODE_ADDLAST)==NULL)
```

```
                {
                }
        }
}

void CPlayer::DeleteItem(DWORD x)
{
        DELETE_SPRITE(grid[x][PLAYER_AREA].sprite);
        ClearItem(x,PLAYER_AREA);
}

void CPlayer::ClearItem(DWORD x,DWORD y)
{
        SecureZeroMemory(&grid[x][y],sizeof(struct __GRID));
        grid[x][y].image=NO_OBJECT;
        grid[x][y].sprite=NO_OBJECT;
        grid[x][y].state=STATE_NONE;
}

void CPlayer::checkItems(int x,int dir,DWORD image,bool *store) //,bool destroy)
{
        x+=dir; // So the starting tile is missed
        while (x>=0 && x<GRID_WIDTH)
        {
                if (grid[x][PLAYER_AREA].image==image)
                {
                        store[x]=true;
                }
                else
                {
                        break;
                }

                x+=dir;
        }
}

DWORD CPlayer::countBlocksInColumn(DWORD x,bool up,DWORD *numBlocks)
{
register DWORD y;

        *(numBlocks)=0;

        for (y=0; y<PLAYER_AREA; y++)
        {
                if (grid[x][(up ? PLAYER_AREA-y-1 : PLAYER_AREA+y+1)].image!=NO_OBJECT &&
                        grid[x][(up ? PLAYER_AREA-y-1 : PLAYER_AREA+y+1)].sprite!=NO_OBJECT)
                {
                        *(numBlocks)=*(numBlocks)+1;
                }
                else
                {
                        return (up ? PLAYER_AREA-y-1 : PLAYER_AREA+y+1);
                }
        }

        return (NONE_FREE);
}

void CPlayer::DoWeAddNewItemAfterMatch(DWORD x,bool up)
{
register DWORD firstFree,numBlocks;

        if ((firstFree=countBlocksInColumn(x,!up,&numBlocks))!=NONE_FREE)
        {
                if (numBlocks<MIN_BLOCKSTOMATCH)
                {
                        // If there is nothing in the line, and nothing in the player area, then we
add directly to the player
                        // area instead of the line.  Not doing this would result in an unwanted gap
                        AddItem(x,(numBlocks==0 && grid[x][PLAYER_AREA].image==NO_OBJECT ?
PLAYER_AREA : firstFree),false);
                }
        }
}

void CPlayer::moveColumnHalf(DWORD x,double speed)
{
register int y;
double move;

        if (halfColumn[x].moveLeft<=0.0)
        {
                if (halfColumn[x].state==STATE_MOVEUP)
                {
                        for (y=PLAYER_AREA; y<GRID_HEIGHT-1; y++)
                        {
                                memcpy((char *) &grid[x][y],(char *) &grid[x][y+1],sizeof(struct
__GRID));
                                dbOffsetSprite(grid[x][y].sprite,0,0);
                                SPRITE_BASE(x,y);
                        }
                }
                else
                {
                        for (y=PLAYER_AREA;   y>0; y--)
                        {
                                memcpy((char *) &grid[x][y],(char *) &grid[x][y-1],sizeof(struct
__GRID));
                                dbOffsetSprite(grid[x][y].sprite,0,0);
                                SPRITE_BASE(x,y);
                        }
                }

                ClearItem(x,(halfColumn[x].state==STATE_MOVEUP ? GRID_HEIGHT-1 : 0));
                halfColumn[x].state=STATE_MOVECHECK;
                //displayColumn(x);
        }
        else
        {
                move=10.0*speed;
                if (halfColumn[x].state==STATE_MOVEUP)
                {
                        for (y=PLAYER_AREA+1; y<GRID_HEIGHT; y++)
                        {
                                dbOffsetSprite(grid[x][y].sprite,0,(int) halfColumn[x].offset);
                        }
                }
                else
                {
                        for (y=PLAYER_AREA-1; y>=0; y--)
                        {
                                dbOffsetSprite(grid[x][y].sprite,0,(int) halfColumn[x].offset);
                        }
                }

                halfColumn[x].moveLeft-=move;
                halfColumn[x].offset+=(halfColumn[x].state==STATE_MOVEUP ? 1.0 : -1.0)*move;
        }
}

void CPlayer::moveSelector(bool left)
{
register DWORD currentXPos;

        currentXPos=(DWORD) ((DWORD) control.x/(DWORD) SELECTORGRAPHIC_WIDTH);
        if ((left && currentXPos>0) ||
                (!left && currentXPos<GRID_WIDTH-1))
        {
                control.state=(left ? STATE_MOVELEFT : STATE_MOVERIGHT);
                control.nextColumn=currentXPos+(left ? -1 : 1); //control.nextColumn-1 :
control.nextColumn+1);
                control.moveLeft=SELECTORGRAPHIC_WIDTH;
        }
}

void CPlayer::Update(double speed,CPlayer *player)
{
register double move;
register DWORD x,mult;
register int y;
register struct __NODE *node,*nextNode;
struct __FADE *fade;

        // Show the star when a matched graphic has been found
        if (xy.show)
        {
                xy.step=(double) dbWrapValue((float) xy.step+((float) 0.5*(float) speed));
                xy.alpha+=25.5*speed*xy.dir;
                xy.rotate+=(double) dbWrapValue((float) xy.rotate+((float) 15.5*(float) speed));
                dbSprite((int) xy.sprite,
                                        (int) ((double) xy.x+((double) 20.0*sin(xy.step))),
                                        (int) ((double) xy.y+((double) 20.0*cos(xy.step))),
                                        IMAGE_STAR);
                dbRotateSprite((int) xy.sprite,(float) xy.rotate);
                dbSetSpriteAlpha((int) xy.sprite,(int) xy.alpha);
                if (xy.alpha>=255.0)
                {
                        xy.dir=0.0-xy.dir;
                }
                else
                if (xy.alpha<=0.0)
                {
                        xy.show=false;
                        dbHideSprite(xy.sprite);
                }
        }

        dbRotateSprite((int) control.sprite,(float) control.rotate);
        control.rotate=(double) dbWrapValue((float) (control.rotate+((float) 10.0*(float) speed)));
        if (control.state!=STATE_NONE)
        {
                if (control.moveLeft<=0.0)
                {
                        // Finished moving
                        control.state=STATE_NONE;
                        control.x=(double) (control.nextColumn*SELECTORGRAPHIC_WIDTH);
                }
                else
                {
                        control.x+=(double) (control.state==STATE_MOVELEFT ? -1.0 :
1.0)*CURSOR_MOVE_SPEED*speed;
                        control.moveLeft-=CURSOR_MOVE_SPEED*speed;
                }

                displaySelector();
                //dbSprite((int) control.sprite,(int) (offset.x+control.x),(int)
(offset.y+control.y),IMAGE_SELECTOR);
        }

        switch (column.state) {
                case    STATE_MOVEUP   :
                case    STATE_MOVEDOWN :       // Move line up or down
                                                                        if (column.moveLeft<=0.0)
                                                                        {

                                                                                if
(column.state==STATE_MOVEUP)

                                                                                {
                                                                                        for (y=1;
y<GRID_HEIGHT; y++)

                                                                                        {

        memcpy((char *) &grid[column.column][y-1],

        (char *) &grid[column.column][y],

        sizeof(struct __GRID));
                                                                                        }


        ClearItem(column.column,GRID_HEIGHT-1);


                                                                                }
                                                                                else
                                                                                {
                                                                                        for (y=GRID_HEIGHT-
1; y>0; y--)

                                                                                        {

        memcpy((char *) &grid[column.column][y],

        (char *) &grid[column.column][y-1],

        sizeof(struct __GRID));
                                                                                        }


        ClearItem(column.column,0);


                                                                                }

                                                                                for (y=0; y<GRID_HEIGHT;
y++)

                                                                                {
                                                                                        if
(grid[column.column][y].sprite!=NO_OBJECT)

                                                                                        {

        dbOffsetSprite(grid[column.column][y].sprite,0,0);

        SPRITE_BASE(column.column,y);
                                                                                        }
                                                                                }


        movedUp[column.column]=(column.state==STATE_MOVEUP ? true : false);;
                                                                                column.state=STATE_NONE;

        checkForMatchingItems(column.column);
                                                                        }
                                                                        else
                                                                        {
                                                                                move=20.0*speed;
                                                                                for (y=0; y<GRID_HEIGHT;
y++)

                                                                                {
                                                                                        if
(grid[column.column][y].sprite!=NO_OBJECT)

                                                                                        {

        dbOffsetSprite((int) grid[column.column][y].sprite,
                            0,

                            (int) column.offset);

                                                                                        }
                                                                                }

                                                                                column.moveLeft-=move;

        column.offset+=(column.state==STATE_MOVEUP ? 1.0 : -1.0)*move;
                                                                        }

                                                                        //displayColumn(column.column);
                                                                        break;
                        };

        // Process anything that rotates
        switch (rotate.state) {
                        case    STATE_ROTATERIGHT       :
                        case    STATE_ROTATELEFT        :       if (rotate.moveLeft<=0.0)
                                                                                {
                                                                                struct __GRID  temp;

                                                                                        memcpy((char *)
&temp,

        (char *) &grid[(rotate.state==STATE_ROTATERIGHT ? GRID_WIDTH-1 : 0)][PLAYER_AREA],

        sizeof(struct __GRID));

                                                                                        if
(rotate.state==STATE_ROTATERIGHT)
                                                                                        {
                                                                                                for
(x=GRID_WIDTH-1; x>0; x--)
                                                                                                {

        memcpy((char *) &grid[x][PLAYER_AREA],

                        (char *) &grid[x-1][PLAYER_AREA],

                        sizeof(struct __GRID));

                                                                                                }
                                                                                        }
                                                                                        else
                                                                                        {
                                                                                                for (x=1;
x<GRID_WIDTH; x++)
                                                                                                {

        memcpy((char *) &grid[x-1][PLAYER_AREA],

                        (char *) &grid[x][PLAYER_AREA],

                        sizeof(struct __GRID));

                                                                                                }
                                                                                        }

                                                                                        memcpy((char *)
&grid[(rotate.state==STATE_ROTATERIGHT ? 0 : GRID_WIDTH-1)][PLAYER_AREA],

        (char *) &temp,

        sizeof(struct __GRID));

                                                                                        for (x=0;
x<GRID_WIDTH; x++)
                                                                                        {

        dbOffsetSprite(grid[x][PLAYER_AREA].sprite,0,0);

        SPRITE_BASE(x,PLAYER_AREA);

                                                                                        }

                                                                                        // Check to see if
any matches have been made

        checkForMatchingItems(rotate.state==STATE_ROTATERIGHT ? 0 : GRID_WIDTH-1);

        rotate.state=STATE_NONE;
                                                                                }
                                                                                else
                                                                                {

                                                                                        for (x=0;
x<GRID_WIDTH; x++)
                                                                                        {

        dbOffsetSprite(grid[x][PLAYER_AREA].sprite,(DWORD) rotate.offset,0);
```