

Appendix 1: Essential Code Listings

MenuScene.m

```
//
//  MenuScene.m
//  BioTetris
//
//  Created by Adam Clark, Chad Johnson, and Dylan Gulick on 3/1/09.
//  Copyright 2009 Oregon State University. All rights reserved.
//

#import "MenuScene.h"
#import "GameScene.h"
#import "ScoresScene.h"
#import "OptionsScene.h"
#import "BioTetrisAppDelegate.h"

@implementation MenuScene
- (id) init {
    self = [super init];
    if (self != nil) {
        Sprite * bg = [Sprite spriteWithFile:@"menu1.png"];
        [bg setPosition:cpv(160, 240)];
        [self add:bg z:0];
        [self add:[MenuItem node] z:1];
    }
    return self;
}
@end

@implementation MenuItem
- (id) init {
    self = [super init];
    if (self != nil) {
        [MenuItemFont setFontSize:30];
        [MenuItemFont setFontName:@"Helvetica"];
        MenuItem *start = [MenuItemFont itemFromString:@"Start Game"];
        target:self selector:@selector(startGame:);
        MenuItem *scores = [MenuItemFont itemFromString:@"High Scores"];
        target:self selector:@selector(scores:);
        MenuItem *options = [MenuItemFont itemFromString:@"Options" target:self];
        selector:@selector(options:);
        Menu *menu = [Menu menuWithItems:start, scores, options, nil];
        menu.position = cpv(160, 150);
        [menu alignItemsVertically];
        [self add:menu];

        if([BioTetrisAppDelegate isMusicEnabled])
            [BioTetrisAppDelegate playBackgroundMusic];
    }
    return self;
}
```

```
}  
  
-(void)startGame: (id)sender {  
    [BioTetrisAppDelegate playEffect:soundFXClick];  
    GameScene * gs = [GameScene node];  
    [[Director sharedDirector] replaceScene:gs];  
}  
-(void)scores: (id)sender {  
    [BioTetrisAppDelegate playEffect:soundFXClick];  
    ScoresScene * ss = [ScoresScene node];  
    [[Director sharedDirector] replaceScene:ss];  
}  
-(void)options: (id)sender {  
    [BioTetrisAppDelegate playEffect:soundFXClick];  
    OptionsScene * os = [OptionsScene node];  
    [[Director sharedDirector] replaceScene:os];  
}  
  
@end
```

OptionsScene.m

```
//  
// OptionsScene.m  
// BioTetris  
//  
// Created by Adam Clark, Chad Johnson, and Dylan Gulick on 3/1/09.  
// Copyright 2009 Oregon State University. All rights reserved.  
//  
  
#import "OptionsScene.h"  
#import "MenuScene.h"  
#import "BioTetrisAppDelegate.h"  
  
@implementation OptionsScene  
- (id) init {  
    self = [super init];  
    if (self != nil) {  
        Sprite * bg = [Sprite spriteWithFile:@"menu1.png"];  
        [bg setPosition:cpv(160, 240)];  
        [self add:bg z:0];  
        [self add:[OptionsLayer node] z:1];  
    }  
    return self;  
}  
  
@end  
  
@implementation OptionsLayer  
- (id) init {  
    self = [super init];  
    if (self != nil) {  
        [MenuItemFont setFontSize:30];  
        [MenuItemFont setFontName:@"Helvetica"];  
    }  
}
```

```
NSString *fxState;
NSString *musicState;

if([BioTetrisAppDelegate isEffectsEnabled])
    fxState = @"ON";
else
    fxState = @"OFF";

if([BioTetrisAppDelegate isMusicEnabled])
    musicState = @"ON";
else
    musicState = @"OFF";

fxMenu = [MenuItemFont itemFromString:[NSString stringWithFormat:@"%@"
%@", @"Sound FX: ", fxState] target:self
selector:@selector(menuSoundFXToggle:)];
musicMenu = [MenuItemFont itemFromString:[NSString
stringWithFormat:@"%@" %@", @"Music: ", musicState] target:self
selector:@selector(menuMusicToggle:)];
MenuItem *back = [MenuItemFont itemFromString:@"Back" target:self
selector:@selector(menuBack:)];
Menu *menu = [Menu menuWithItems:fxMenu, musicMenu, back, nil];
menu.position = cpv(160, 150);
[menu alignItemsVertically];
[self add:menu];
}
return self;
}

- (void)menuSoundFXToggle: (id)sender {
[BioTetrisAppDelegate playEffect:soundFXClick];
[BioTetrisAppDelegate setEffectsEnabled:![BioTetrisAppDelegate
isEffectsEnabled]];

NSString *fxState;
if([BioTetrisAppDelegate isEffectsEnabled])
    fxState = @"ON";
else
    fxState = @"OFF";

[[fxMenu label] setString:[NSString stringWithFormat:@"%@" %@", @"Sound
FX: ", fxState]];
}

- (void)menuMusicToggle: (id)sender {
[BioTetrisAppDelegate playEffect:soundFXClick];
[BioTetrisAppDelegate setMusicEnabled:![BioTetrisAppDelegate
isMusicEnabled]];

if([BioTetrisAppDelegate isMusicEnabled])
    [BioTetrisAppDelegate playBackgroundMusic];
else
    [BioTetrisAppDelegate stopBackgroundMusic];

NSString *musicState;
```

```
if([BioTetrisAppDelegate isMusicEnabled])
    musicState = @"ON";
else
    musicState = @"OFF";

[[musicMenu label] setString:[NSString stringWithFormat:@"%@@ %@",
@"Music: ", musicState]];
}

-(void)menuBack: (id)sender {
    [BioTetrisAppDelegate playEffect:soundFXClick];
    MenuScene * ms = [MenuScene node];
    [[Director sharedDirector] replaceScene:ms];
}
@end
```

GameScene.m

GameScene.m will not be shown in its entirety. Instead key aspects of the code that highlight how the game works will be shown.

```
//
// GameScene.m
// BioTetris
//
// Created by Adam Clark, Chad Johnson, and Dylan Gulick on 3/1/09.
// Copyright 2009 Oregon State University. All rights reserved.
//
struct gridNode {
    int x;
    int y;
    bool full;
    int spriteTag;
};

int pieces[31][5][5] =
{
    {
        {0,0,0,0,0},
        {0,0,2,0,0},
        {0,0,3,0,0},
        {0,0,1,0,0},
        {0,0,4,0,0}
    },
    {
        {0,0,0,0,0},
        {0,0,0,0,0},
        {0,0,1,3,0},
        {0,0,4,2,0},
        {0,0,0,0,0}
    },
},

/*
A structure to hold the current piece.
*/
struct piece {
```

```
//4 blocks in a piece
struct gridNode block1;
struct gridNode block2;
struct gridNode block3;
struct gridNode block4;

};

//populate curPiece and set gridArr values to full for first piece
for(int i = 0; i < 5; i++)
{
    for(int j = 0; j < 5; j++)
    {
        switch (pieces[curPieceInt][i][j]) {
            case 1:
                [block1 setPosition:cpv(gridArr[curPos.x - 2 + j][curPos.y
- 2 + i].x,
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].y)];
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].full = YES;
                curPiece.block1.x = curPos.x - 2 + j;
                curPiece.block1.y = curPos.y - 2 + i;
                break;
            case 2:
                [block2 setPosition:cpv(gridArr[curPos.x - 2 + j][curPos.y
- 2 + i].x,
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].y)];
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].full = YES;
                curPiece.block2.x = curPos.x - 2 + j;
                curPiece.block2.y = curPos.y - 2 + i;
                break;
            case 3:
                [block3 setPosition:cpv(gridArr[curPos.x - 2 + j][curPos.y
- 2 + i].x,
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].y)];
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].full = YES;
                curPiece.block3.x = curPos.x - 2 + j;
                curPiece.block3.y = curPos.y - 2 + i;
                break;
            case 4:
                [block4 setPosition:cpv(gridArr[curPos.x - 2 + j][curPos.y
- 2 + i].x,
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].y)];
                gridArr[curPos.x - 2 + j][curPos.y - 2 + i].full = YES;
                curPiece.block4.x = curPos.x - 2 + j;
                curPiece.block4.y = curPos.y - 2 + i;
                break;
            default:
                break;
        }
    }
}

-(void)checkMatches: (int)points
{
```

```
self.isTouchEnabled = NO;
matchFlag = 0;
int matchCounter = 0;
for(int k = 9; k >= 0; k--)
{
    //Only check to second to top row because nothing can match above the
first row
    for(int l = 17; l >= 1; l--)
    {
        if(gridArr[k][l].spriteTag%4+4 == 4 && gridArr[k][l-
1].spriteTag%4 == 3
            && gridArr[k][l].full && gridArr[k][l-1].full)
        {
            [self removeByTag:gridArr[k][l].spriteTag];
            gridArr[k][l].full = NO;
            [self removeByTag:gridArr[k][l-1].spriteTag];
            gridArr[k][l-1].full = NO;
            matchFlag = 1;
            [self addBar:gridArr[k][l-1].spriteTag];
            matchCounter++;
        }
        if(gridArr[k][l].spriteTag%4 == 3 && gridArr[k][l-
1].spriteTag%4+4 == 4
            && gridArr[k][l].full && gridArr[k][l-1].full)
        {
            [self removeByTag:gridArr[k][l].spriteTag];
            gridArr[k][l].full = NO;
            [self removeByTag:gridArr[k][l-1].spriteTag];
            gridArr[k][l-1].full = NO;
            matchFlag = 1;
            [self addBar:gridArr[k][l-1].spriteTag];
            matchCounter++;
        }
        if(gridArr[k][l].spriteTag%4 == 2 && gridArr[k][l-1].spriteTag%4
== 1
            && gridArr[k][l].full && gridArr[k][l-1].full)
        {
            [self removeByTag:gridArr[k][l].spriteTag];
            gridArr[k][l].full = NO;
            [self removeByTag:gridArr[k][l-1].spriteTag];
            gridArr[k][l-1].full = NO;
            matchFlag = 1;
            [self addBar:gridArr[k][l-1].spriteTag];
            matchCounter++;
        }
        if(gridArr[k][l].spriteTag%4 == 1 && gridArr[k][l-1].spriteTag%4
== 2
            && gridArr[k][l].full && gridArr[k][l-1].full)
        {
            [self removeByTag:gridArr[k][l].spriteTag];
            gridArr[k][l].full = NO;
            [self removeByTag:gridArr[k][l-1].spriteTag];
            gridArr[k][l-1].full = NO;
            matchFlag = 1;
            [self addBar:gridArr[k][l-1].spriteTag];
            matchCounter++;
        }
    }
}
```

```
    }  
  }  
}  
if(matchCounter > 0)  
{  
    combo+=matchCounter;  
}  
if(points == 1 && matchCounter > 0)  
{  
    multiplier++;  
}  
}  
}  
  
-(void)pieceDownSched: (ccTime) delta  
{  
    if(mutex)  
    {  
        [self pieceDownSched];  
        return;  
    }  
  
    self.isTouchEnabled = YES;  
    if([self movePieceDown])  
    {  
        return;  
    }  
    else  
    {  
        while(mutex)  
        {  
  
        }  
        mutex = !mutex;  
        self.isTouchEnabled = NO;  
        [self unschedule:@selector(pieceDownSched:)];  
  
        counter1 = 0;  
        counter2 = 0;  
        counter3 = 0;  
        counter4 = 0;  
        [self checkMatches:0];  
        combo2 = combo;  
        combo = 0;  
        for(int i = 1; i <= combo2; i++)  
        {  
            combo+=i;  
        }  
        [self addPoints];  
        curScore-=combo;  
        combo = combo2;  
  
        [self schedule:@selector(blockDownSched:) interval:.1];  
  
        return;  
    }  
}
```

```
-(bool)movePieceDown
{
    while(mutex){

    }
    mutex = !mutex;

    //If there was not enough space available clear the board
    //for some reason if you don't clear the board it still thinks there
    //are blocks if you try to play again.
    if(gameOver == 1)
    {
        for(int x = 0; x <= 9; x++)
        {
            for(int y = 0; y <= 17; y++)
            {
                gridArr[x][y].full = NO;
            }
        }
        GameOverScene *go = [GameOverScene node];
        [[Director sharedDirector] replaceScene:go];
    }

    self.isTouchEnabled = NO;
    for(int i = 0; i < 5; i++)
    {
        for(int j = 0; j < 5; j++)
        {
            if(pieces[curPieceInt][i][j] != 0)
            {
                if(gridArr[curPos.x - 2 + j][curPos.y - 2 + i + 1].full
//block below this is full, is out of scope for piece (i cant be 5)
                && i == 4)
                {
                    mutex = !mutex;
                    return NO;
                }
                if(gridArr[curPos.x - 2 + j][curPos.y - 2 + i + 1].full
//block below is full, isnt part of piece
                && pieces[curPieceInt][i+1][j] == 0)
                {
                    mutex = !mutex;
                    return NO;
                }
                if(curPos.y >= (17 - yStop))//end of the field
                {
                    mutex = !mutex;
                    return NO;
                }
            }
        }
    }

    //if we have not yet returned NO, then this move is valid
    //loop through the array backwards so we can move the blocks down
    for(int k = 4; k >= 0; k--)
```

```
{
    for(int l = 4; l >= 0; l--)
    {
        if(pieces[curPieceInt][l][k] == 4)//found block4
        {
            CocosNode *node = [self
getByTag:gridArr[curPiece.block4.x][curPiece.block4.y].spriteTag];
            [node
setPosition:cpv(gridArr[curPiece.block4.x][curPiece.block4.y+1].x,
gridArr[curPiece.block4.x][curPiece.block4.y+1].y)];
            gridArr[curPiece.block4.x][curPiece.block4.y].full = NO;
            gridArr[curPiece.block4.x][curPiece.block4.y+1].spriteTag =
gridArr[curPiece.block4.x][curPiece.block4.y].spriteTag;
            curPiece.block4.y+=1;
            gridArr[curPiece.block4.x][curPiece.block4.y].full = YES;
        }
        if(pieces[curPieceInt][l][k] == 3)//found block3
        {
            CocosNode *node = [self
getByTag:gridArr[curPiece.block3.x][curPiece.block3.y].spriteTag];
            [node
setPosition:cpv(gridArr[curPiece.block3.x][curPiece.block3.y+1].x,
gridArr[curPiece.block3.x][curPiece.block3.y+1].y)];
            gridArr[curPiece.block3.x][curPiece.block3.y].full = NO;
            gridArr[curPiece.block3.x][curPiece.block3.y+1].spriteTag =
gridArr[curPiece.block3.x][curPiece.block3.y].spriteTag;

            curPiece.block3.y+=1;
            gridArr[curPiece.block3.x][curPiece.block3.y].full = YES;
        }
        if(pieces[curPieceInt][l][k] == 2)//found block2
        {
            CocosNode *node = [self
getByTag:gridArr[curPiece.block2.x][curPiece.block2.y].spriteTag];
            [node
setPosition:cpv(gridArr[curPiece.block2.x][curPiece.block2.y+1].x,
gridArr[curPiece.block2.x][curPiece.block2.y+1].y)];
            gridArr[curPiece.block2.x][curPiece.block2.y].full = NO;
            gridArr[curPiece.block2.x][curPiece.block2.y+1].spriteTag =
gridArr[curPiece.block2.x][curPiece.block2.y].spriteTag;
            curPiece.block2.y+=1;
            gridArr[curPiece.block2.x][curPiece.block2.y].full = YES;
        }
        if(pieces[curPieceInt][l][k] == 1)//found block1
        {
            CocosNode *node = [self
getByTag:gridArr[curPiece.block1.x][curPiece.block1.y].spriteTag];
            [node
setPosition:cpv(gridArr[curPiece.block1.x][curPiece.block1.y+1].x,
gridArr[curPiece.block1.x][curPiece.block1.y+1].y)];
            gridArr[curPiece.block1.x][curPiece.block1.y].full = NO;
            gridArr[curPiece.block1.x][curPiece.block1.y+1].spriteTag =
gridArr[curPiece.block1.x][curPiece.block1.y].spriteTag;

            curPiece.block1.y+=1;
            gridArr[curPiece.block1.x][curPiece.block1.y].full = YES;
        }
    }
}
```

```
    }  
  }  
}  
  
//piece moved down, update curPos  
curPos.y += 1;  
mutex = !mutex;  
return YES;  
}  
  
-(bool)ccTouchesBegan:(NSSet *)touches withEvent:(UIEvent *)event  
{  
  if(inGameScene != 1)  
  {  
    inGameScene = 0;  
    return kEventHandled;  
  }  
  if(mutex)  
  {  
    return NO;  
  }  
  mutex = !mutex;  
  
  //get the location of the touch  
  UITouch *touch = [touches anyObject];  
  CGPoint location = [touch locationInView: [touch view]];  
  
  //check for bottom right corner  
  if(location.x > 280  
    && location.y > 440)  
  {  
    fastDrop = 1;  
    [self unschedule:@selector(pieceDownSched:)];  
    //blockSpeed = .1;  
    [self schedule:@selector(pieceDownSched:) interval:.1];  
  
    mutex = !mutex;  
    return YES;  
  }  
  
  mutex = !mutex;  
  return NO;  
}  
  
/*  
  A method called when a user finishes a touch motion. This method moves  
the block left or right  
  based on where the user touched. It also serves as it's own vaidator,  
ensuring that the block  
  has enough space to move whichever direction the user is requesting.  
*/  
-(bool)ccTouchesEnded:(NSSet *)touches withEvent:(UIEvent *)event  
{  
  if(inGameScene != 1)  
  {  
    inGameScene = 0;
```

```
        return kEventHandled;
    }
    if(mutex)
    {
        return kEventIgnored;
    }
    mutex = !mutex;

    //get the location of the touch
    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInView: [touch view]];

    if(fastDrop == 1)
    {
        [self unschedule:@selector(pieceDownSched:)];
        [self schedule:@selector(pieceDownSched:) interval:blockSpeed];

        mutex = !mutex;
        fastDrop = 0;
        return kEventHandled;
    }

    //check for pause
    if(location.x > 160-61
        && location.x < 160+61
        && location.y < 50)
    {
        mutex = !mutex;
        [self pause:self];
        return kEventHandled;
    }

    //get the blocks
    CocosNode *block1 = [self getByTag:curPiece.block1.spriteTag];
    CocosNode *block2 = [self getByTag:curPiece.block2.spriteTag];
    CocosNode *block3 = [self getByTag:curPiece.block3.spriteTag];
    CocosNode *block4 = [self getByTag:curPiece.block4.spriteTag];

    //If the block is within the board offset confines, allow the move
    if(location.x > gridArr[curPos.x][curPos.y].x + xOffset
        && gridArr[curPiece.block1.x][curPiece.block1.y].x <= 320 -
boardXOffset - xOffset
        && gridArr[curPiece.block2.x][curPiece.block2.y].x <= 320 -
boardXOffset - xOffset
        && gridArr[curPiece.block3.x][curPiece.block3.y].x <= 320 -
boardXOffset - xOffset
        && gridArr[curPiece.block4.x][curPiece.block4.y].x <= 320 -
boardXOffset - xOffset
        && (gridArr[curPiece.block1.x + 1][curPiece.block1.y].full == NO
            || (gridArr[curPiece.block1.x + 1][curPiece.block1.y].spriteTag
== curPiece.block2.spriteTag
                || gridArr[curPiece.block1.x +
1][curPiece.block1.y].spriteTag == curPiece.block3.spriteTag
                || gridArr[curPiece.block1.x +
1][curPiece.block1.y].spriteTag == curPiece.block4.spriteTag))
            && (gridArr[curPiece.block2.x + 1][curPiece.block2.y].full == NO
```

```
    || (gridArr[curPiece.block2.x + 1][curPiece.block2.y].spriteTag ==
curPiece.block1.spriteTag
        || gridArr[curPiece.block2.x + 1][curPiece.block2.y].spriteTag
== curPiece.block3.spriteTag
        || gridArr[curPiece.block2.x + 1][curPiece.block2.y].spriteTag
== curPiece.block4.spriteTag))
    && (gridArr[curPiece.block3.x + 1][curPiece.block3.y].full == NO
        || (gridArr[curPiece.block3.x + 1][curPiece.block3.y].spriteTag ==
curPiece.block1.spriteTag
            || gridArr[curPiece.block3.x + 1][curPiece.block3.y].spriteTag
== curPiece.block2.spriteTag
            || gridArr[curPiece.block3.x + 1][curPiece.block3.y].spriteTag
== curPiece.block4.spriteTag))
    && (gridArr[curPiece.block4.x + 1][curPiece.block4.y].full == NO
        || (gridArr[curPiece.block4.x + 1][curPiece.block4.y].spriteTag ==
curPiece.block1.spriteTag
            || gridArr[curPiece.block4.x + 1][curPiece.block4.y].spriteTag
== curPiece.block2.spriteTag
            || gridArr[curPiece.block4.x + 1][curPiece.block4.y].spriteTag
== curPiece.block3.spriteTag))
    )
    {
        //Touched to the right of the block
        for(int k = 4; k >= 0; k--)
        {
            for(int l = 4; l >= 0; l--)
            {
                if(pieces[curPieceInt][l][k] == 4)//found block4
                {
                    //set current locations .full to NO and new locations
                    to YES
                    gridArr[curPiece.block4.x++][curPiece.block4.y].full
                    = NO;
                    gridArr[curPiece.block4.x][curPiece.block4.y].full =
                    YES;

                    gridArr[curPiece.block4.x][curPiece.block4.y].spriteTag =
                    curPiece.block4.spriteTag;
                }
                if(pieces[curPieceInt][l][k] == 3)
                {
                    gridArr[curPiece.block3.x++][curPiece.block3.y].full
                    = NO;
                    gridArr[curPiece.block3.x][curPiece.block3.y].full =
                    YES;

                    gridArr[curPiece.block3.x][curPiece.block3.y].spriteTag =
                    curPiece.block3.spriteTag;
                }
                if(pieces[curPieceInt][l][k] == 2)
                {
                    gridArr[curPiece.block2.x++][curPiece.block2.y].full
                    = NO;
                    gridArr[curPiece.block2.x][curPiece.block2.y].full =
                    YES;
```

```
    gridArr[curPiece.block2.x][curPiece.block2.y].spriteTag =
curPiece.block2.spriteTag;
        if(pieces[curPieceInt][1][k] == 1)
        {
            gridArr[curPiece.block1.x+1][curPiece.block1.y].full
= NO;
            gridArr[curPiece.block1.x][curPiece.block1.y].full =
YES;

            gridArr[curPiece.block1.x][curPiece.block1.y].spriteTag =
curPiece.block1.spriteTag;
                }
            }

        [block1
setPosition:cpv(gridArr[curPiece.block1.x][curPiece.block1.y].x,
gridArr[curPiece.block1.x][curPiece.block1.y].y)];
        [block2
setPosition:cpv(gridArr[curPiece.block2.x][curPiece.block2.y].x,
gridArr[curPiece.block2.x][curPiece.block2.y].y)];
        [block3
setPosition:cpv(gridArr[curPiece.block3.x][curPiece.block3.y].x,
gridArr[curPiece.block3.x][curPiece.block3.y].y)];
        [block4
setPosition:cpv(gridArr[curPiece.block4.x][curPiece.block4.y].x,
gridArr[curPiece.block4.x][curPiece.block4.y].y)];

        //change curPos to note that the piece is now one unit right
curPos.x++;
    }
    if(location.x < gridArr[curPos.x][curPos.y].x - xOffset
&& gridArr[curPiece.block1.x][curPiece.block1.y].x > boardXOffset +
xOffset
&& gridArr[curPiece.block2.x][curPiece.block2.y].x > boardXOffset +
xOffset
&& gridArr[curPiece.block3.x][curPiece.block3.y].x > boardXOffset +
xOffset
&& gridArr[curPiece.block4.x][curPiece.block4.y].x > boardXOffset +
xOffset
&& (gridArr[curPiece.block1.x - 1][curPiece.block1.y].full == NO
|| (gridArr[curPiece.block1.x - 1][curPiece.block1.y].spriteTag ==
curPiece.block2.spriteTag
|| gridArr[curPiece.block1.x - 1][curPiece.block1.y].spriteTag
== curPiece.block3.spriteTag
|| gridArr[curPiece.block1.x - 1][curPiece.block1.y].spriteTag
== curPiece.block4.spriteTag))
&& (gridArr[curPiece.block2.x - 1][curPiece.block2.y].full == NO
|| (gridArr[curPiece.block2.x - 1][curPiece.block2.y].spriteTag ==
curPiece.block1.spriteTag
|| gridArr[curPiece.block2.x - 1][curPiece.block2.y].spriteTag
== curPiece.block3.spriteTag
|| gridArr[curPiece.block2.x - 1][curPiece.block2.y].spriteTag
== curPiece.block4.spriteTag))
&& (gridArr[curPiece.block3.x - 1][curPiece.block3.y].full == NO
|| (gridArr[curPiece.block3.x - 1][curPiece.block3.y].spriteTag ==
```

```
curPiece.block1.spriteTag
    || gridArr[curPiece.block3.x - 1][curPiece.block3.y].spriteTag
== curPiece.block2.spriteTag
    || gridArr[curPiece.block3.x - 1][curPiece.block3.y].spriteTag
== curPiece.block4.spriteTag))
    && (gridArr[curPiece.block4.x - 1][curPiece.block4.y].full == NO
    || (gridArr[curPiece.block4.x - 1][curPiece.block4.y].spriteTag ==
curPiece.block1.spriteTag
    || gridArr[curPiece.block4.x - 1][curPiece.block4.y].spriteTag
== curPiece.block2.spriteTag
    || gridArr[curPiece.block4.x - 1][curPiece.block4.y].spriteTag
== curPiece.block3.spriteTag))

)
{
//Touched to the left of the block
for(int k = 0; k <= 4; k++)
{
    for(int l = 0; l <= 4; l++)
    {
        if(pieces[curPieceInt][l][k] == 1)//found block4
        {
            //set current locations .full to NO and new locations
to YES
            gridArr[curPiece.block1.x--][curPiece.block1.y].full
= NO;
            gridArr[curPiece.block1.x][curPiece.block1.y].full =
YES;

            gridArr[curPiece.block1.x][curPiece.block1.y].spriteTag =
curPiece.block1.spriteTag;
        }
        if(pieces[curPieceInt][l][k] == 2)//found block3
        {
            gridArr[curPiece.block2.x--][curPiece.block2.y].full
= NO;
            gridArr[curPiece.block2.x][curPiece.block2.y].full =
YES;

            gridArr[curPiece.block2.x][curPiece.block2.y].spriteTag =
curPiece.block2.spriteTag;
        }
        if(pieces[curPieceInt][l][k] == 3)//found block2
        {
            gridArr[curPiece.block3.x--][curPiece.block3.y].full
= NO;
            gridArr[curPiece.block3.x][curPiece.block3.y].full =
YES;

            gridArr[curPiece.block3.x][curPiece.block3.y].spriteTag =
curPiece.block3.spriteTag;
        }
        if(pieces[curPieceInt][l][k] == 4)//found block1
        {
            gridArr[curPiece.block4.x--][curPiece.block4.y].full
= NO;
```

```
gridArr[curPiece.block4.x][curPiece.block4.y].full =
YES;

    gridArr[curPiece.block4.x][curPiece.block4.y].spriteTag =
curPiece.block4.spriteTag;
    }
}

[block1
setPosition:cpv(gridArr[curPiece.block1.x][curPiece.block1.y].x,
gridArr[curPiece.block1.x][curPiece.block1.y].y)];
[block2
setPosition:cpv(gridArr[curPiece.block2.x][curPiece.block2.y].x,
gridArr[curPiece.block2.x][curPiece.block2.y].y)];
[block3
setPosition:cpv(gridArr[curPiece.block3.x][curPiece.block3.y].x,
gridArr[curPiece.block3.x][curPiece.block3.y].y)];
[block4
setPosition:cpv(gridArr[curPiece.block4.x][curPiece.block4.y].x,
gridArr[curPiece.block4.x][curPiece.block4.y].y)];

    //change curPos to note that the block is now one unit left
    curPos.x--;
}
mutex = !mutex;
return kEventHandled;
}
```

Appendix 2: Anything Else

Introduction To The Cocos2d Framework

This guide is intended to get you started using the cocos2d framework. The framework is mostly intended for making 2D games, but it could still be used for any application you have in mind.

Introduction to the iPhone

Everything you see in an iPhone program works through what is called a window. Every thing that is added to the program is attached into the window. Usually you will attach what are called views. For example, the settings menu of an iPhone would be a table view, which is already predefined in the standard framework. All you would have to do is add the view and tell it how many rows you need, what they say, and what they do when pressed.

This is really all our group had experience in with the standard iPhone frameworks. If your project is going to need to use the standard frameworks you should definitely google basic tutorials.

Cocos2d Programming

Before starting you will need to visit <http://code.google.com/p/cocos2d-iphone/> to get the tar file for the framework. Also check out their guides and download a demo program if you do not want to hassle with getting the framework into your project.

This section will show you how to make a simple menu. Keep in mind that syntax will change over time so check cocos2d API if terms in this guide are not correct. A link to the API can be found at the previously mentioned website.

This guide is going to assume you are familiar with Xcode. It will not detail small things like how to add a file to a project.

Making a Menu

To start create a new NSObject subclass to your project. You can name it whatever you want but this class will be your menu. I will refer to the files as MenuScene.h and MenuScene.m. Be sure to import cocos2d.h in all files.

When you created this project AppDelegate should have been created for you. They would be called "NameOfYourProjectAppDelegate.h" and also a .m file.

In the .m AppDelegate add the function:

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
```

In this function we need to set up our window and attach our director into the window. Director is the function in cocos2d that manipulates what you will see on the iPhone screen. What is currently running in the window is called a Scene. So a director is used to stop, pause, or change the scene.

Add the following code to your project:

```
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
mainScreen] bounds]];
[window setUserInteractionEnabled:YES];
[window setMultipleTouchEnabled:YES];

[[Director sharedDirector] attachInWindow:window];

[window makeKeyAndVisible];

MenuScene *ms = [MenuScene node];

[[Director sharedDirector] runWithScene:ms];
```

Function names in Objective-C are very long and usually self explanatory so this guide will mention key lines of codes and explain what they are doing.

The second to last line of code declares the MenuScene and the last line tells the program to start with that scene.

Now we need to implement MenuScene. In MenuScene.h add the following code:

```
@interface MenuScene : Scene {

}

@end

@interface MenuLayer : Layer {}
-(void)startGame: (id)sender;
-(void)scores: (id)sender;
-(void)options: (id)sender;
@end
```

Notice the type of the MenuScene interface was changed to Scene. Also notice there is a second interface for the menu we will add into this scene. Menus are known as Layers. The three functions you see in MenuLayer are actually going to be what is called when that specific item in the menu is selected.

Now switch to MenuScene.m and implement the menu. Add the following code for the implementation of MenuScene:

```
@implementation MenuScene
- (id) init {
    self = [super init];
    if (self != nil) {
        [self add:[MenuItem node] z:1];
    }
    return self;
}
@end
```

This will let the scene know that it is going to have a menu called MenuItem in it. The z:1 means that it will be added on layer 1 of the scene. Layers are varying levels of plains in the scene. So something added to z:2 would show up on top of some on layer z:1.

Now add the following code for the implementation for MenuItem:

```
@implementation MenuItem
- (id) init {
    self = [super init];
    if (self != nil) {
        [MenuItemFont setFontSize:30];
        [MenuItemFont setFontName:@"Helvetica"];
        MenuItem *start = [MenuItemFont itemFromString:@"Start Game"];
        MenuItem *scores = [MenuItemFont itemFromString:@"High Scores"];
        MenuItem *options = [MenuItemFont itemFromString:@"Options"];
        target:self selector:@selector(startGame:);
        target:self selector:@selector(scores:);
        target:self selector:@selector(options:);
        Menu *menu = [Menu menuWithItems:start, scores, options, nil];
        menu.position = cpv(160, 150);
        [menu alignItemsVertically];
        [self add:menu];

        return self;
    }

    -(void)startGame: (id)sender {
    }
    -(void)scores: (id)sender {
    }
    -(void)options: (id)sender {
    }
}
@end
```

For each item that is going to be in the menu you create a MenuItem. In each MenuItem you tell it what selector it belongs to. After you have your MenuItems you create Menu and add the MenuItems. Now that you have the Menu you add the Menu into the scene.

The three empty functions are your selectors. You would add code here to perform tasks you want to do when that item has been clicked. For example, when the scores item is selected in the menu you would tell the director to switch to a scene that has high score information.

That's all there is to making a cocos2d menu. There is a lot more you can do with this framework so go take a look at the API to get an idea of what can be accomplished.

If you are having trouble getting this to work for you there are many guides online.

Various Helpful Websites for Learning iPhone Development

Basic iPhone Programming:

<http://www.iphonedevcentral.org/tutorials.php?page=ViewTutorial&id=16&uid=85906760>

Cocos2d: <http://cocos2d.org/>

CocosLive: <http://groups.google.com/group/cocoslive-discuss>

<http://code.google.com/p/cocos2d-iphone/source/browse/trunk/tests/cocosLive/cocosLiveDemo.m>

http://groups.google.com/group/cocoslive-discuss/browse_thread/thread/c19257cd16d21dbf#

iCodeBlog: <http://icodeblog.com/>

iPhone SDK Tutorial:

<http://theappleblog.com/2008/08/04/tutorial-build-a-simple-rss-reader-for-iphone/>

Our Project Is Up On The App Store Under The Title "DNA Blocks". Check It Out!