
Checkers Program

Contents

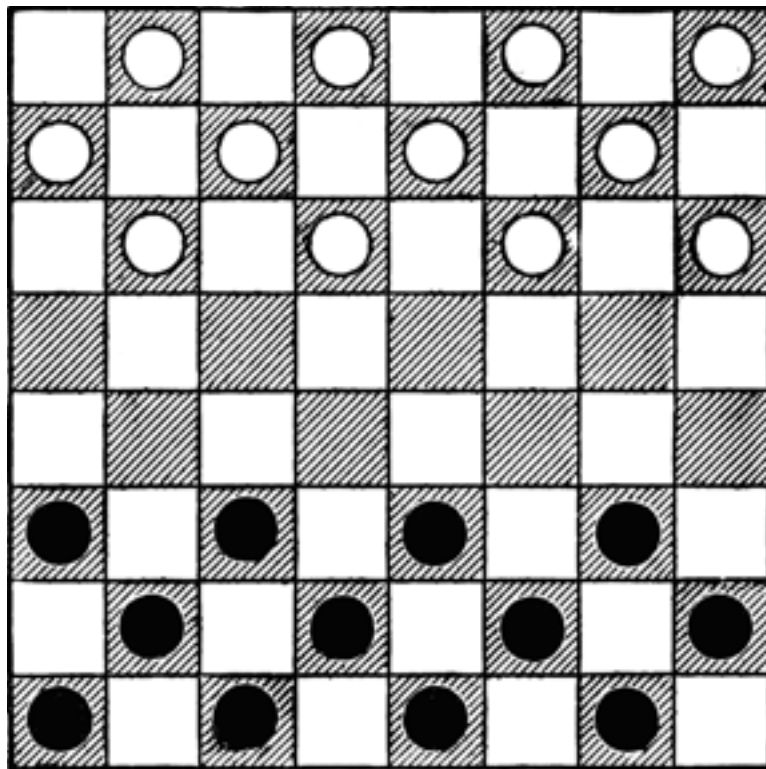
3	Project Definition
4,5	Investigation & Analysis
6	System Requirements
7	Design
8-11	Storyboard
12	Variable Table
13	Class Diagram
14	Flowchart
15	Test Plan
16-33	Software Development
33-34	Testing
35	User Testing
36-37	Screenshots
38	Evaluation

Definition

Using X-code (an IDE capable of simulating iPhone and mac applications written in either swift or objective-C, the latter will be the language i will use, I am going to create a computer game based around the classic board game Checkers. The game consists of each player taking turns to move one of their pieces or 'checkers' one space diagonally along the checkered board or move their piece over an enemy piece that has a gap on the other side in order to 'capture' the piece. The game ends when one player has captured all of the other players pieces, the remaining player wins.

While the concept of checkers has origins as far back as 3000 BC, the modern version os checkers was first constructed in the 1100s in France when an aristocrat got the idea to play the ancient game on a checkers board and changing the number of pieces to 12 each side in order to accommodate.

The end users of this project will most likely be students aged from 16-18 but may be played by anyone who wishes to. As such I shall correspond with students from my college in order to ascertain the features that i may wish to include or omit from the game.



An example of the layout of the traditional game of checkers

Investigation and Analysis

In order to investigate the end-user requirements I intend to interview students between the ages 16-18, the questions I intend to ask are:

Q1) Have you ever played a game of Checkers or Draughts?

Q2) What features, if any, of this do you like or think to be complementary to the style of game?

Q3) What features, if any, of this do you dislike or think to be detrimental to the style of game?

Initial test results

I have interviewed several college/ sixth form students aged between 16 and 18. Their names are Adam Griffiths, Samuel Chambers, Mathew Daly, Andrew Stainton, Kyle Cunningham and Hadrian Hughes.

Q1) Have you ever played a game of Checkers or Draughts?

-yes(5)

-no(1)

Q2) What features, if any, of this do you like or think to be complementary to the style of game?

-Turn Timer (2)

-A.I. Opponent (3)

Q3) What features, if any, of this do you dislike or think to be detrimental to the style of game?

-Turn Timer(3)

-Interacting with Others (1)

-Rules are Confusing (1)

Analysis of initial test results

All 5 out of the 6 interviewed persons had played a form of this game. However due to this style of game being popular in the past it means that they are not necessarily talking about the same game or game franchise when stating liked or disliked features. Due to this it would not be appropriate to include all of the liked features, or to omit all of the disliked features. Some of the features may also contradict each other and so including all of them may be impossible.

One such feature would be the turn timer, I will not be adding this feature due to the the returned difference of opinion in accordance to the inclusion of this feature.

I will also be omitting the inclusion of the requested feature of an A.I Opponent due the the technical difficulty that is included with the creation of the feature.

Instead I will create and 'flesh out' a new version of this game including and omitting a select number of the requested features.

Based on the gathered results of the interview, the related analysis and from collaboration with the end user I have decided on these requirements.

System Requirements

Basic requirements

- Game ends when one player has no pieces remaining in play
- Game is turn based so actions happen only upon user input
- User is able to select and execute certain actions

Variable features

- Which players turn it is
- The 'king' state of each piece still in play
- Which player has won

Hardware/Software requirements

- OS X Mavericks or later (minimum needed to run x-code)
- intel i5 processor or better (minimum needed to run x-code)
- VGA 640x480 or higher-resolution screen supported by OS X (minimum needed to run x-code)
- 4GB RAM (minimum needed to run x-code)
- Computer keyboard and Mouse (For user input)
- VGA monitor or higher resolution (For program output)

Design

Visual Necessities

- Screen Size : 1080px x 1920px
- Background : Grey Bordered Screen
- Realistic Checker Pieces
- Simple UI
- Screen must be able to fit 8x8 grid

Functional Necessities


- Checker pieces rendered in correct places at the beginning of play
- User able to interact with the pieces in play
- Pieces removed from board when jumped by opposing pieces
- Game ends when only one type of checker piece left on the board

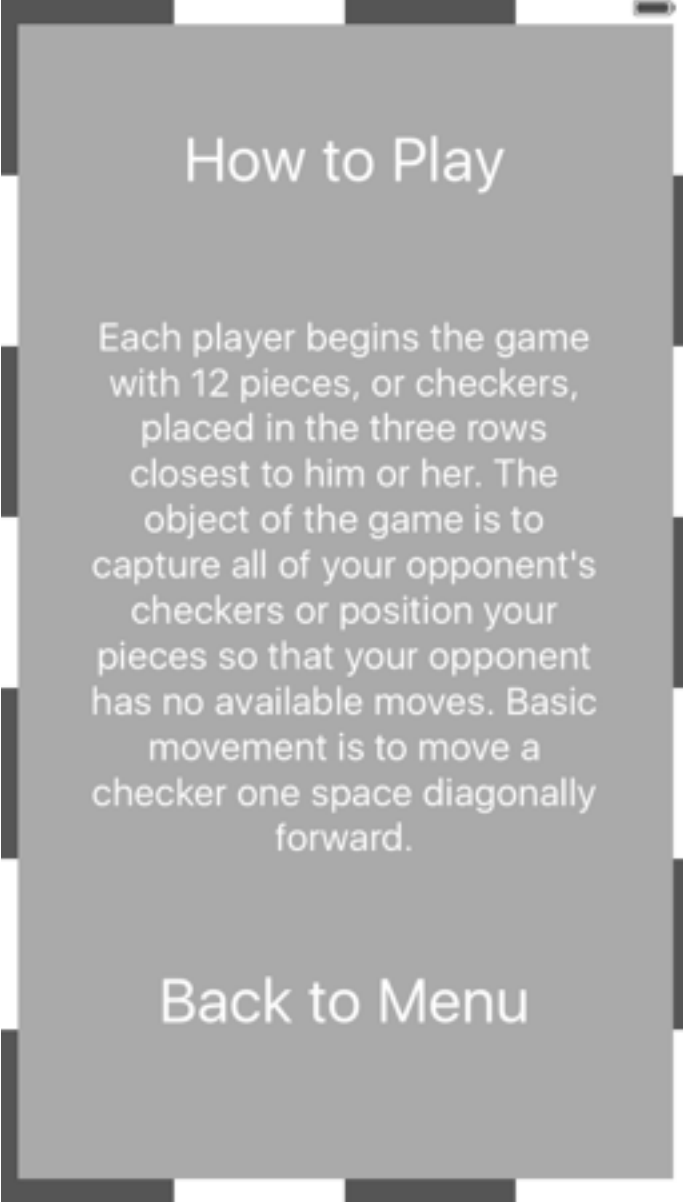
I will provide image backgrounds and assets sized and scaled so that they will accurately fit the desired screen size on the iPhone six plus or on a screen of the same resolution. I will also provide the assets for the sprite kit related items such as the rendered realistic checker pieces or the sprite kit buttons.

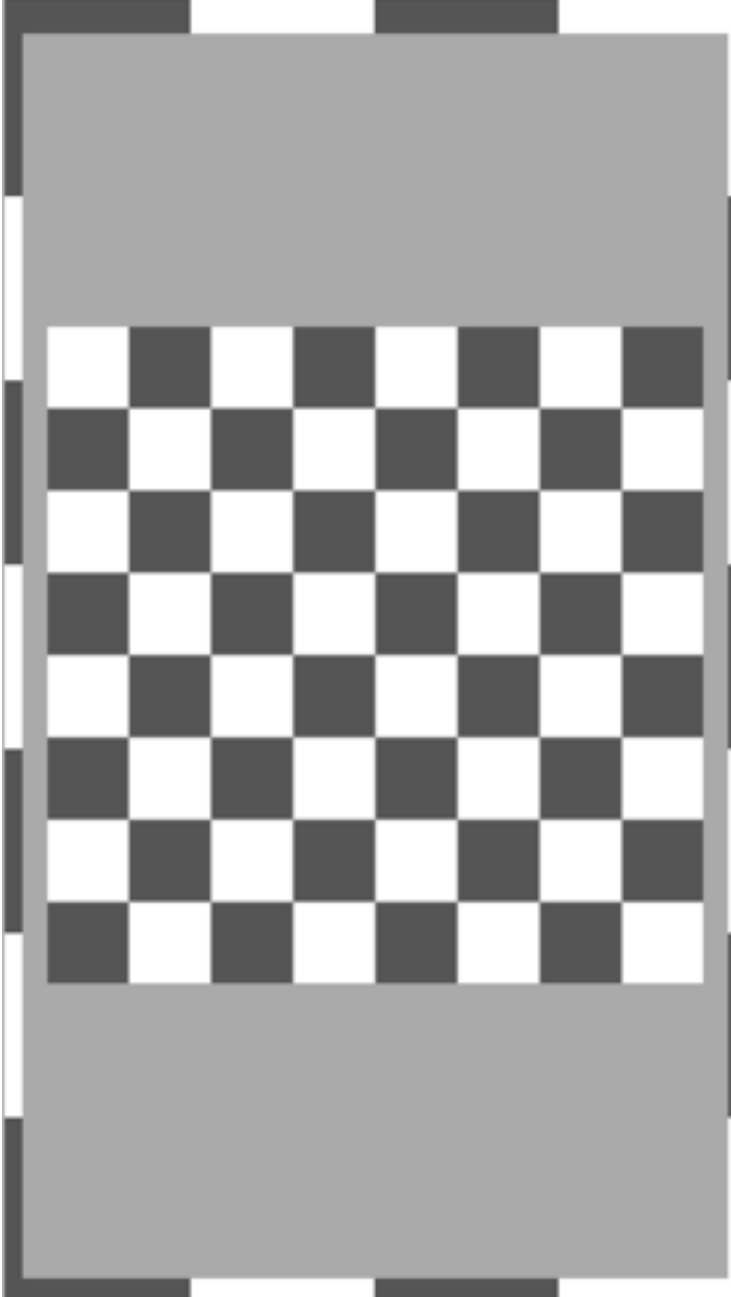
I will design the UI so it will be able to accommodate all of my requested features without created a cluttered game environment for the user in order to keep the application pleasing to the eye while still retaining the required functionality.

I will initiate the program by creating the grid out of pre-assigned assets and attaching these assets to an array with the evenly spaced grid like positions assigned as values. And layered on top of the relevant grid tiles I will render and initiate the 24 starting checker pieces in their relevant grid cells.

Storyboard

	<p>Menu Screen</p> <p><No.1></p> <p>Purpose of this screen</p> <p>Launch Screen for the game, initial menu used for navigation</p> <p>Colour of back ground</p> <p>Black & White</p> <p>Font colour(s) used</p> <p>Black & White</p> <p>Explain how the user will use it and what happens next</p> <p><i>The user will interact with the page by clicking or 'tapping' the buttons on the screen, e.g.</i></p> <p><i>The start game button will link to the board screen.</i></p> <p><i>The how to play button will redirect the user to the how to play screen which will show the user the rules and instructions of the game.</i></p> <p><i>The Quit Button will simply exit the application.</i></p>
------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>Rules Screen</p> <p><No.2></p> <p>Purpose of this screen</p> <p>Screen used to instruct the user of the basic mechanics of the game.</p> <p>Colour of back ground</p> <p>Black & White</p> <p>Font colour(s) used</p> <p>White</p> <p>Explain how the user will use it and what happens next</p> <p><i>The user will interact with the page by clicking or 'tapping' the buttons on the screen, e.g.</i></p> <p><i>The back to menu button will return the user to the previous screen (in this case, the menu screen)</i></p>
------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<h2>Game Screen</h2> <h3><No.3></h3> <p>Purpose of this screen</p> <p>Screen used for the actual playing of the game</p> <p>Colour of back ground</p> <p>Black & White</p> <p>Font colour(s) used</p> <p>N/A</p> <p>Explain how the user will use it and what happens next</p> <p><i>The user will interact with the page by clicking or 'tapping' the buttons on the screen, e.g.</i></p> <p><i>The user will touch the relevant game piece (either black or white), this will then add small overplayed x's on the valid spaces that the piece can be moved to, touching one of these x's will move the piece and continue on to the next players turn.</i></p> <p><i>When someone meets the win condition of the game they will be redirected to the win screen.</i></p>
------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>Win Screen</p> <p><No.4></p> <p>Purpose of this screen</p> <p>To display the outcome of the game and provide a way back to the menu screen</p> <p>Colour of back ground</p> <p>Black & White</p> <p>Font colour(s) used</p> <p>Black & White</p> <p>Explain how the user will use it and what happens next</p> <p><i>The user will interact with the page by clicking or 'tapping' the buttons on the screen, e.g.</i></p> <p><i>It will output the result of the game.</i></p> <p><i>The back to menu button will return the user to the menu screen.</i></p>
------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

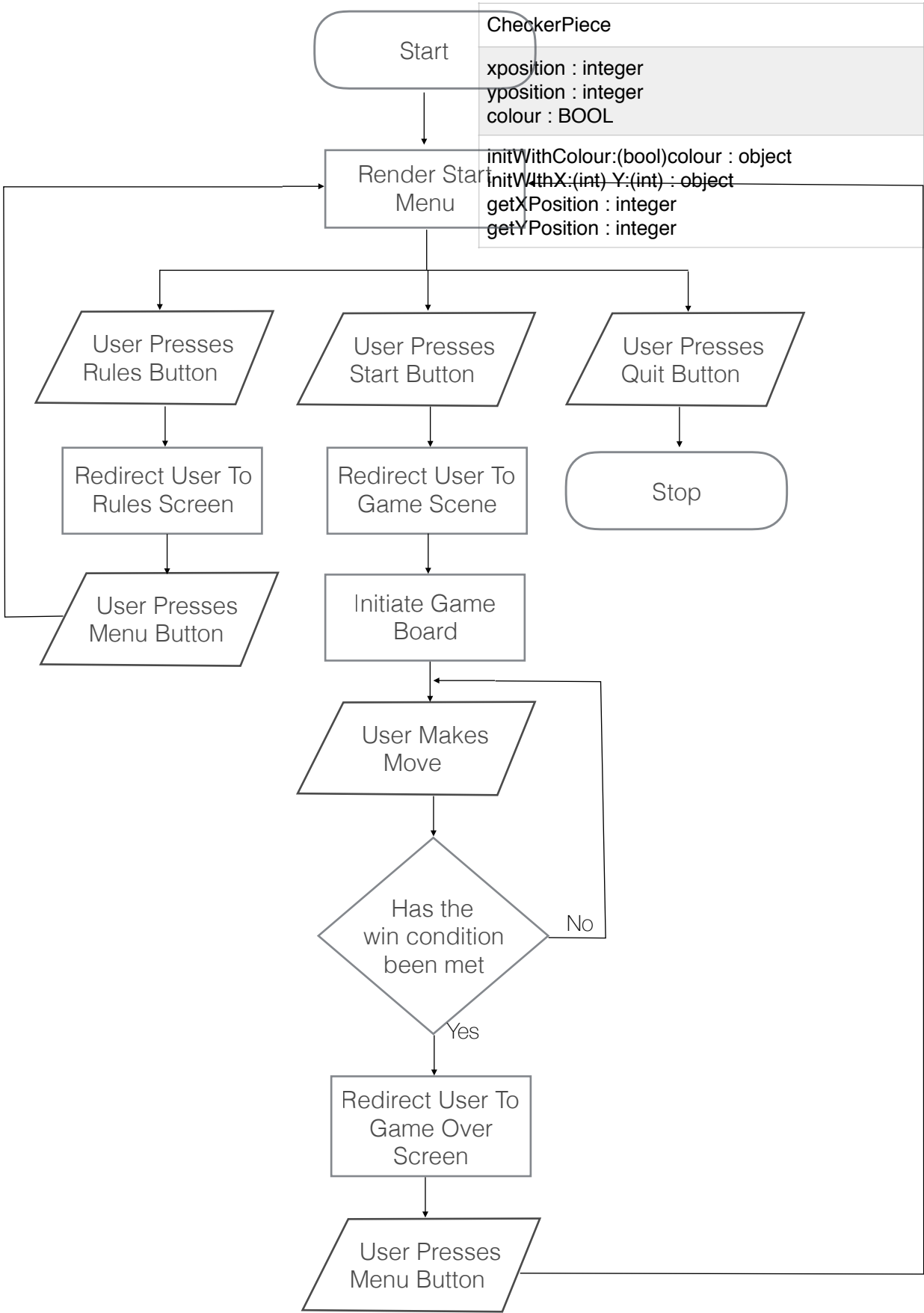
Variable Table

Name	Type/Storage	Scope	Comment
gridCell[65]	SKSpriteNode	local	Used to create and declare the game board both visually and logically
piecePosition[8][8]	integer	local	Used to store the state of the piece in each gridCell and allow for easier and more logical reference
CheckerPiece	SKNode	global	Used as a class reference to assign, get values to and initiate the checker pieces
Vectors	SKNode	global	Used as a class reference to assign, get values to and initiate the checker pieces
cellWidth	integer	global	Used to define a set value for the width of every grid cell
cellHeight	integer	global	Used to define a set value for the height of every grid cell
squares	integer	global	Used to set an amount of grid cells to be created
leftX	integer	global	Used to define the co-ordinates of the far left border of the grid
topY	integer	global	Used to define the co-ordinates of the top border of the grid

Class Diagram

Vector
xposition : integer yposition : integer
initWithX:(int) Y:(int) : object getXPosition : integer getYPosition : integer

Flowchart



Test Plan

Function Being tested	Method of Input	Input	Expected Result
Changing Scene	Pressing button	Rules Button Pressed	Scene changes from Menu Scene to Rules Screen
		Menu Button Pressed	Scene changes from Rules Screen to Menu Screen
		Start Button Pressed	Scene changes from Menu Scene to Game Scene
	Win Condition Met	last white piece 'taken'	Scene changes from Game Scene to Game Over Screen
	Pressing Button	Menu Button Pressed	Scene changes from Game Over Screen to Menu Screen
Gameplay	Press Piece and Vector	Clicked adjacent top left vector	Piece is moved to the adjacent top left piece
		Clicked adjacent top right vector	Piece is moved to the adjacent top right piece
		Clicked adjacent bottom left	Piece is moved to the adjacent bottom left piece
		Clicked adjacent bottom right	Piece is moved to the adjacent bottom right piece
		Clicked top left vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed
		Clicked top right vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed
		Clicked bottom left vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed
		Clicked bottom right vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed

Software Development

I will be using Xcode as my interactive development environment and using objective-c (an object oriented superset of C) as my programming language of choice. The first section of code i will be working on will be the GameScene.h file provided by Xcode on creation of a Sprite Kit Project.

```
@interface GameScene()
{
    //Image View Array Declaration
    SKSpriteNode *gridCell[65];
    int piecePosition[8][8]; //0 empty, 1 white, 2 black

    CheckerPiece *whitePieces[12], *blackPieces[12], *touchedPiece; //Piece objects declared
    NSMutableArray *pieceContainer;

    Vector *vectors[4]; //4 possible places a piece can move to
}
```

In this snippet of code I have declared the gridCell array which will be used for creating and referencing the different spaces on the grid or board. It has been given 65 slots within the array so that it may hold values for each of the required 64 grid slots while starting at 1 instead of zero for the sake of simplicity.

The array piecePosition is also declared in order for us to be able to assign the current state of a tile (unoccupied, occupied by player one, occupied by player 2) and get this value later on.

The variables whitePieces, blackPieces and touched pieces are being imported from the Checker Piece class which i will talk about later on. And the pieceContainer is used to convert a node to a different data type. The array vectors array is used for the definition of the 4 possible locations of a vector and is imported from the vector class that i will talk about later.

```
@property BOOL contentCreated;
@end

int cellWidth; //Gap between checkers
int cellHeight;
int squares = 64; //Amount of squares on board
int leftX; //Left border of board
int topY; //Top border of board

@implementation GameScene
- (void)didMoveToView:(SKView *)view
{
    if ([self contentCreated])
    {
        [self createSceneContents]; //Separate method so it can be called again later
        self.contentCreated = YES;
    }
}
```

The contentCreated Boolean property is used to ensure that the createSceneContents method is only referenced once during the didMoveToView method. The variables defined here are all global variables within the game scene file used to give default values to and to initialise the game board.


```
#pragma mark - Scene Contents
-(void)createSceneContents
{
    /* Setup your scene here */
    pieceContainer = [[NSMutableArray alloc] init]; //Initialise array

    [self setAnchorPoint:CGPointMake(0.5, 0.5)]; //Sets anchor point to center
#pragma mark Background Setup
    //Background
    SKSpriteNode *backgroundImage = [SKSpriteNode spriteNodeWithImageNamed:@"CheckersBackground"];
    backgroundImage.position = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY(self.frame)); //Centers the sprite
    backgroundImage.size = self.frame.size;
    [self addChild:backgroundImage];
    self.scaleMode = SKSceneScaleModeAspectFit;
}
```

This code snippet shows the declaration and definition of the before mentioned createSceneContents method that is used as a work around to ensure that the viewDidLoad method and its contents can essentially be re-called as needed in order to solicit a reset.

At the beginning of this method the pieceContainer array is initialised for use. The default anchor point is also set at the middle of the screen both by the x and y plains.

After this we create an SKSprite node with the name backgroundImage and assign it to the image by the reference "CheckersBackground". We then assign various values to this node such as its position and size. We then add this as a child of the frame in order to add it to the scene.

```
#pragma mark Variable declaration and definition
int index;
int row, column;
row = 0;
column = 0;
int firstColour = 0;

//Define variables from header
cellWidth = 115;
cellHeight = cellWidth;
topY = 380-4*cellHeight;
leftX = 207-4*cellWidth;
```

We then declare and define local variables for the current method, these variables are the index, row, column and firstColour integers. As well as defining the variables that were defined in the header by assigning values to them.

```
#pragma mark Grid Creation and Initialisation
for(int index = 0; index < 8; index++)
{
    for(int jindex = 0; jindex < 8; jindex++)
    {
        piecePosition[index][jindex] = 0; //Define whole array
    }
}

for(int index = 0; index < 12; index++)
{
    whitePieces[index] = [[CheckerPiece alloc] initWithColour:true]; //True means white
    blackPieces[index] = [[CheckerPiece alloc] initWithColour:false]; //False means black
}

for(index=1; index<=64; index++)
{
    if(firstColour == 0) //Assigns the first colour for the first square of the board and continues in an alternating pattern
    {
        gridCell[index] = [SKSpriteNode spriteNodeWithImageNamed:@"WhiteSquare"];
    }
    else
    {
        gridCell[index] = [SKSpriteNode spriteNodeWithImageNamed:@"BlackSquare"];
    }

    //Assigns a position value to the gridCell array for each space within this array
    gridCell[index].position = CGPointMake((leftX + cellWidth * column)-(cellWidth*1.26), (topY + cellHeight * row)-(cellHeight*3));
    gridCell[index].zPosition = 1;
    gridCell[index].size = CGSizeMake(115, 115);
    //Gives a few values that every instance of one of the board square node shares
}
```

Within this snippet we cycle through two nested for loops in order to assign a default value to every position in the piecePosition array, from [0][0] to [7][7]. The next for loop is then used to cycle from 0 to 12 in order to initiate all twelve values in each of the whitePieces array and the blackPieces array by giving them using the checker piece class in order to assign a different sprite to each of them.

Next a new for loop is started in which it will cycle through the current code 64 times, once for each square in the grid. At the start of this for loop we use a conditional if statement to check which firstColour is 0 and if it is it creates a white square, else it creates a black square.

After this a position value is assigned based on the number created for the for loop in order from the grid cells to be in the correct spaces through the following formula:

x co-ordinate = (left border of the grid + width of a cell * column number) - (cellWidth * 1.26)

y co-ordinate = (top border of the grid + height of a cell * row number) - (cellHeight * 3)

Then more values are assigned to the currently indexed grid cell, such as setting the z position and the size.

```

switch(index)
{
    //Individual initialisation and declaration of the default pieces for their default positions
#pragma mark White Piece Declaration
    case 2:
        piecePosition[1][0] = 1;
        [whitePieces[0] setPositionX:1 Y:0];
        whitePieces[0].position = gridCell[index].position;
        whitePieces[0].zPosition = 2;
        [self addChild:whitePieces[0]];
        break;

    case 4:
        piecePosition[3][0] = 1;
        [whitePieces[1] setPositionX:3 Y:0];
        whitePieces[1].position = gridCell[index].position;
        whitePieces[1].zPosition = 2;
        [self addChild:whitePieces[1]];
        break;

    case 6:
        piecePosition[5][0] = 1;
        [whitePieces[2] setPositionX:5 Y:0];
        whitePieces[2].position = gridCell[index].position;
        whitePieces[2].zPosition = 2;
        [self addChild:whitePieces[2]];
        break;

    case 8:
        piecePosition[7][0] = 1;
        [whitePieces[3] setPositionX:7 Y:0];
        whitePieces[3].position = gridCell[index].position;
        whitePieces[3].zPosition = 2;
        [self addChild:whitePieces[3]];
        break;

    case 9:
        piecePosition[0][1] = 1;
        [whitePieces[4] setPositionX:0 Y:1];
        whitePieces[4].position = gridCell[index].position;
        whitePieces[4].zPosition = 2;
        [self addChild:whitePieces[4]];
        break;

    case 11:
        piecePosition[2][1] = 1;
        [whitePieces[5] setPositionX:2 Y:1];
        whitePieces[5].position = gridCell[index].position;
        whitePieces[5].zPosition = 2;
        [self addChild:whitePieces[5]];
        break;

    case 13:
        piecePosition[4][1] = 1;
        [whitePieces[6] setPositionX:4 Y:1];
        whitePieces[6].position = gridCell[index].position;
        whitePieces[6].zPosition = 2;
        [self addChild:whitePieces[6]];
        break;

    case 15:
        piecePosition[6][1] = 1;
        [whitePieces[7] setPositionX:6 Y:1];
        whitePieces[7].position = gridCell[index].position;
        whitePieces[7].zPosition = 2;
        [self addChild:whitePieces[7]];
        break;

    case 18:
        piecePosition[1][2] = 1;
        [whitePieces[8] setPositionX:1 Y:2];
        whitePieces[8].position = gridCell[index].position;
        whitePieces[8].zPosition = 2;
        [self addChild:whitePieces[8]];
        break;

    case 20:
        piecePosition[3][2] = 1;
        [whitePieces[9] setPositionX:3 Y:2];
        whitePieces[9].position = gridCell[index].position;
        whitePieces[9].zPosition = 2;
        [self addChild:whitePieces[9]];
        break;

    case 22:
        piecePosition[5][2] = 1;
        [whitePieces[10] setPositionX:5 Y:2];
        whitePieces[10].position = gridCell[index].position;
        whitePieces[10].zPosition = 2;
        [self addChild:whitePieces[10]];
        break;

    case 24:
        piecePosition[7][2] = 1;
        [whitePieces[11] setPositionX:7 Y:2];
        whitePieces[11].position = gridCell[index].position;
        whitePieces[11].zPosition = 2;
        [self addChild:whitePieces[11]];
        break;
}

```

```

#pragma mark Black Piece Initialisation
case 41:
    piecePosition[0][5] = 2;
    [blackPieces[0] setPositionX:0 Y:5];
    blackPieces[0].position = gridCell[index].position;
    blackPieces[0].zPosition = 2;
    [self addChild:blackPieces[0]];
    break;

case 43:
    piecePosition[2][5] = 2;
    [blackPieces[1] setPositionX:2 Y:5];
    blackPieces[1].position = gridCell[index].position;
    blackPieces[1].zPosition = 2;
    [self addChild:blackPieces[1]];
    break;

case 45:
    piecePosition[4][5] = 2;
    [blackPieces[2] setPositionX:4 Y:5];
    blackPieces[2].position = gridCell[index].position;
    blackPieces[2].zPosition = 2;
    [self addChild:blackPieces[2]];
    break;

case 47:
    piecePosition[6][5] = 2;
    [blackPieces[3] setPositionX:6 Y:5];
    blackPieces[3].position = gridCell[index].position;
    blackPieces[3].zPosition = 2;
    [self addChild:blackPieces[3]];
    break;

case 50:
    piecePosition[1][6] = 2;
    [blackPieces[4] setPositionX:1 Y:6];
    blackPieces[4].position = gridCell[index].position;
    blackPieces[4].zPosition = 2;
    [self addChild:blackPieces[4]];
    break;

case 52:
    piecePosition[3][6] = 2;
    [blackPieces[5] setPositionX:3 Y:6];
    blackPieces[5].position = gridCell[index].position;
    blackPieces[5].zPosition = 2;
    [self addChild:blackPieces[5]];
    break;

case 54:
    piecePosition[5][6] = 2;
    [blackPieces[6] setPositionX:5 Y:6];
    blackPieces[6].position = gridCell[index].position;
    blackPieces[6].zPosition = 2;
    [self addChild:blackPieces[6]];
    break;

case 56:
    piecePosition[7][6] = 2;
    [blackPieces[7] setPositionX:7 Y:6];
    blackPieces[7].position = gridCell[index].position;
    blackPieces[7].zPosition = 2;
    [self addChild:blackPieces[7]];
    break;

case 57:
    piecePosition[0][7] = 2;
    [blackPieces[8] setPositionX:0 Y:7];
    blackPieces[8].position = gridCell[index].position;
    blackPieces[8].zPosition = 2;
    [self addChild:blackPieces[8]];
    break;

case 59:
    piecePosition[2][7] = 2;
    [blackPieces[9] setPositionX:2 Y:7];
    blackPieces[9].position = gridCell[index].position;
    blackPieces[9].zPosition = 2;
    [self addChild:blackPieces[9]];
    break;

case 61:
    piecePosition[4][7] = 2;
    [blackPieces[10] setPositionX:4 Y:7];
    blackPieces[10].position = gridCell[index].position;
    blackPieces[10].zPosition = 2;
    [self addChild:blackPieces[10]];
    break;

case 63:
    piecePosition[6][7] = 2;
    [blackPieces[11] setPositionX:6 Y:7];
    blackPieces[11].position = gridCell[index].position;
    blackPieces[11].zPosition = 2;
    [self addChild:blackPieces[11]];
    break;
}

```

In this section of code it initiates the 24 individual checker pieces and their starter locations in the piecePosition array with the same positions of the currently indexed grid cell array. It also assigns the z positions for each of the checkers. At also creates each instance as a child of the frame in order to create these aforementioned instances. This is done twelve times for each colour set as well as only running for the predefined grid cell locations.

```
[self addChild:gridCell[index]];

//Used to alternate the Colours of the game board by toggling the value of the first colour variable when needed
column = column + 1;
firstColour = 1 - firstColour;
if(column > 7)
{
    row = row + 1;
    firstColour = 1 - firstColour;
    column = 0;
}

//Pretty self explanatory, enables user interaction for each grid cell
gridCell[index].userInteractionEnabled = YES;
}
```

Within this code snippet it creates the instance of the grid cell array in order to render the board by creating it as a child of the overall frame.

It then toggles the state of the firstColour array by using the formula:

$\text{firstColour} = 1 - \text{firstColour};$

This formula inverts the value from one to zero and vice-versa, this happens as is firstColour is equal to 1 then $\text{firstColour} = 1 - 1$ which is equal to zero, however if firstColour is equal to 0 then $\text{firstColour} = 1 - 0$ which is equal to one, creating an arithmetical toggle to the integer that is being used as a pseudo-boolean.

Using this value the row and column are incremented in a nested if statement in order to ensure that the first colour is un toggled in the case of a new row represented in the array.

Finally within the snippet the last line of code ensures that each grid cell may be interacted with by the user, before closing the previously mentioned for loop as well as the createSceneContents method.

```
#pragma mark - Touch Method
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    /* Called when a touch begins */
    UITouch *touch = [[event allTouches] anyObject];
    CGPoint location = [touch locationInView:self];

    SKNode *node = [self nodeAtPoint:location];
}
```

Within this snippet the touchesBegan method is defined allowing for user interaction through tapping to the included contents. This method is provided by the sprite kit template available with Xcode. It begins with the declaration and definition of the UITouch object with the reference of touch and allows for the interaction between user input and code snippets. The touched location is also saved to the 'location' CGPoint value. This value is then used to select a specific location of a clicked node, this is done so by the node object.

```
if(node != nil)
{
    //Remove all vectors already on screen
    [[self childNodeWithName:@"vector1"] removeFromParent];
    [[self childNodeWithName:@"vector2"] removeFromParent];
    [[self childNodeWithName:@"vector3"] removeFromParent];
    [[self childNodeWithName:@"vector4"] removeFromParent];

    //If the node doesn't have a name
    if(node.name == nil)
    {
        int xposition, yposition;

        //Converts node to CheckerPiece
        [pieceContainer addObject:node.parent];
        touchedPiece = pieceContainer[0];
        [pieceContainer removeLastObject];

        xposition = [touchedPiece getXPosition];
        yposition = [touchedPiece getYPosition];
    }
}
```

The first conditional statement of this snippet is in place to ensure that the enclosed code only runs if the touched object is a node. This helps to ensure that the relevant code is only executed when the relevant type of object (in this case a node) is touched.

Nested within this conditional statement is a set of statements than clear any 'vectors' on the screen so that more aren't created while others still exist, this helps to declutter the game scene interface and alleviate potential confusion from the users mind.

Preceding this is another conditional statement that is only executed if the node in question doesn't have an assigned name.

Nested within this is a variable declaration for the xposition and yposition variables as well as a small (yet inefficient/unconventional way) of converting a node to another data type, while i wouldn't recommend converting a node this way it is the only way i am currently able to execute with my level of technical knowledge.

Finally within this snippet we are assigning values to these variables by taking a value from the touchedPiece method (I will talk about this method later).

```

        if(!((xposition - 1) < 0) && !((yposition + 1) > 7)) //If the selected square is not out of bounds
        {
            if(piecePosition[xposition - 1][yposition + 1] == 0) //If the top-left square is empty
            {
                vectors[0] = [[Vector alloc] initWithX:xposition - 1 Y:yposition + 1]; //Initialises the vector with the
                position of the empty space
                SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
                vector.text = @"X";
                vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition - 1)), gridCell[1].position.y + (115 *
                (yposition + 1))); //Sets the graphical position of the vector
                vector.name = @"vector1";
                vector.zPosition = 100; //Brings the sprite to the front
                vector.fontSize = 60;
                [self addChild:vector];
            }
            else
            {
                if((piecePosition[xposition - 2][yposition + 2] == 0) && (piecePosition[xposition - 1][yposition + 1] !=
                piecePosition[xposition][yposition])) //If the piece is an enemy piece and the space top-left of that is
                empty
                {
                    //Place a vector
                    vectors[0] = [[Vector alloc] initWithX:xposition - 2 Y:yposition + 2];
                    SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
                    vector.text = @"X";
                    vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition - 2)), gridCell[1].position.y +
                    (115 * (yposition + 2)));
                    vector.name = @"vector1";
                    vector.zPosition = 100;
                    vector.fontSize = 60;
                    [self addChild:vector];
                }
            }
        }
    }
}

```

This snippet shows the initialisation for the two potential values for the top left vector and renders the 'X' on the relevant space on the game board in its place, this position can either be x-1 and y+1 from the touched nodes location which would be diagonally adjacent to the top left or it could potentially be x-2 and y+2 from the touched nodes location if there is an 'enemy' checker in the x-1, y+1 slot and an empty space in the x-2 and y+2 slot.

```

        if(!((xposition + 1) > 7) && !((yposition + 1) > 7))
        {
            if(piecePosition[xposition + 1][yposition + 1] == 0)
            {
                vectors[1] = [[Vector alloc] initWithX:xposition + 1 Y:yposition + 1];
                SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
                vector.text = @"X";
                vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition + 1)), gridCell[1].position.y + (115 *
                (yposition + 1)));
                vector.name = @"vector2";
                vector.zPosition = 100;
                vector.fontSize = 60;
                [self addChild:vector];
            }
            else
            {
                if((piecePosition[xposition + 2][yposition + 2] == 0) && (piecePosition[xposition + 1][yposition + 1] !=
                piecePosition[xposition][yposition]))
                {
                    vectors[1] = [[Vector alloc] initWithX:xposition + 2 Y:yposition + 2];
                    SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
                    vector.text = @"X";
                    vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition + 2)), gridCell[1].position.y +
                    (115 * (yposition + 2)));
                    vector.name = @"vector2";
                    vector.zPosition = 100;
                    vector.fontSize = 60;
                    [self addChild:vector];
                }
            }
        }
    }
}

```

This snippet shows the initialisation for the two potential values for the top right vector and renders the 'X' on the relevant space on the game board in its place, this position can either be x+1 and y+1 from the touched nodes location which would be diagonally adjacent to the top right or it could potentially be x+2 and y+2 from the touched nodes location if there is an 'enemy' checker in the x+1, y+1 slot and an empty space in the x+2 and y+2 slot.


```

if(!((!(xposition - 1) < 0)) && (!(yposition - 1) < 0))
{
    if(piecePosition[xposition - 1][yposition - 1] == 0)
    {
        vectors[2] = [[Vector alloc] initWithX:xposition - 1 Y:yposition - 1];
        SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
        vector.text = @"X";
        vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition - 1)), gridCell[1].position.y + (115 * (yposition - 1)));
        vector.name = @"vector3";
        vector.zPosition = 100;
        vector.fontSize = 60;
        [self addChild:vector];
    }
    else
    {
        if((!(piecePosition[xposition - 2][yposition - 2] == 0) && (piecePosition[xposition - 1][yposition - 1] != piecePosition[xposition][yposition])))
        {
            vectors[2] = [[Vector alloc] initWithX:xposition - 2 Y:yposition - 2];
            SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
            vector.text = @"X";
            vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition - 2)), gridCell[1].position.y + (115 * (yposition - 2)));
            vector.name = @"vector3";
            vector.zPosition = 100;
            vector.fontSize = 60;
            [self addChild:vector];
        }
    }
}
}

```

This snippet shows the initialisation for the two potential values for the bottom left vector and renders the 'X' on the relevant space on the game board in its place, this position can either be x-1 and y-1 from the touched nodes location which would be diagonally adjacent to the bottom left or it could potentially be x-2 and y-2 from the touched nodes location if there is an 'enemy' checker in the x-1, y-1 slot and an empty space in the x-2 and y-2 slot.

```

if(!((!(xposition + 1) > 7) && (!(yposition - 1) < 0))
{
    if(piecePosition[xposition + 1][yposition - 1] == 0)
    {
        vectors[3] = [[Vector alloc] initWithX:xposition + 1 Y:yposition - 1];
        SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
        vector.text = @"X";
        vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition + 1)), gridCell[1].position.y + (115 * (yposition - 1)));
        vector.name = @"vector4";
        vector.zPosition = 100;
        vector.fontSize = 60;
        [self addChild:vector];
    }
    else
    {
        if((!(piecePosition[xposition + 2][yposition - 2] == 0) && (piecePosition[xposition + 1][yposition - 1] != piecePosition[xposition][yposition])))
        {
            vectors[3] = [[Vector alloc] initWithX:xposition + 2 Y:yposition - 2];
            SKLabelNode *vector = [SKLabelNode labelNodeWithFontNamed:@"Wingdings"];
            vector.text = @"X";
            vector.position = CGPointMake(gridCell[1].position.x + (115 * (xposition + 2)), gridCell[1].position.y + (115 * (yposition - 2)));
            vector.name = @"vector4";
            vector.zPosition = 100;
            vector.fontSize = 60;
            [self addChild:vector];
        }
    }
}
}

```

This snippet shows the initialisation for the two potential values for the bottom right vector and renders the 'X' on the relevant space on the game board in its place, this position can either be x+1 and y-1 from the touched nodes location which would be diagonally adjacent to the bottom right or it could potentially be x+2 and y-2 from the touched nodes location if there is an 'enemy' checker in the x+1, y-1 slot and an empty space in the x+2 and y-2 slot.


```

else //else a vector was pressed
{
    //Get the number from the vector's name
    int vectorNumber;
    if(![[node.name rangeOfString:@"1"].location == NSNotFound])
    {
        vectorNumber = 0;
    }
    else if(![[node.name rangeOfString:@"2"].location == NSNotFound])
    {
        vectorNumber = 1;
    }
    else if(![[node.name rangeOfString:@"3"].location == NSNotFound])
    {
        vectorNumber = 2;
    }
    else if(![[node.name rangeOfString:@"4"].location == NSNotFound])
    {
        vectorNumber = 3;
    }
}

```

This else statement linked to the conditional statement 'if(node.name == nil)' will occur if a vector has been pressed by the user. Nested within this statement is the declaration of the vectorNumber variable and several linked if and else if statements that detect the name of the touched vector by checking the range of the string for the relevant and update the vectorNumber variable relatively.

```

int oldX, oldY;
oldX = [touchedPiece getXPosition];
oldY = [touchedPiece getYPosition];
piecePosition[oldX][oldY] = 0;
[touchedPiece setPositionX:[vectors[vectorNumber] getXPosition] Y:[vectors[vectorNumber] getYPosition]];
if([touchedPiece colour])
{
    piecePosition[[[touchedPiece getXPosition]][[touchedPiece getYPosition]]] = 1;
}
else
{
    piecePosition[[[touchedPiece getXPosition]][[touchedPiece getYPosition]]] = 2;
}

switch(vectorNumber)
{
    case 0:
        [[self findPieceX:[touchedPiece getXPosition] + 1] Y:[touchedPiece getYPosition] - 1] removeFromParent; //
        Removes the piece that was passed over
        NSLog(@"%d", [self findPieceX:[touchedPiece getXPosition] + 1] Y:[touchedPiece getYPosition] - 1] );
        piecePosition[[touchedPiece getXPosition] + 1][touchedPiece getYPosition] - 1] = 0; //Sets the appropriate
        array index to 0
        break;
    case 1:
        [[self findPieceX:[touchedPiece getXPosition] - 1] Y:[touchedPiece getYPosition] - 1] removeFromParent;
        NSLog(@"%d", [self findPieceX:[touchedPiece getXPosition] - 1] Y:[touchedPiece getYPosition] - 1] );
        piecePosition[[touchedPiece getXPosition] - 1][touchedPiece getYPosition] - 1] = 0;
        break;
    case 2:
        [[self findPieceX:[touchedPiece getXPosition] + 1] Y:[touchedPiece getYPosition] + 1] removeFromParent;
        NSLog(@"%d", [self findPieceX:[touchedPiece getXPosition] + 1] Y:[touchedPiece getYPosition] + 1] );
        piecePosition[[touchedPiece getXPosition] + 1][touchedPiece getYPosition] + 1] = 0;
        break;
    case 3:
        [[self findPieceX:[touchedPiece getXPosition] - 1] Y:[touchedPiece getYPosition] + 1] removeFromParent;
        NSLog(@"%d", [self findPieceX:[touchedPiece getXPosition] - 1] Y:[touchedPiece getYPosition] + 1] );
        piecePosition[[touchedPiece getXPosition] - 1][touchedPiece getYPosition] + 1] = 0;
        break;
    default:
        NSLog(@"Error");
        break;
}
[self winCheck];
}
}
}

```

This snippet begins with a way to store the values of the location of the current touched node and to set the piecePosition value for this location to 0.

Then a statement makes use of custom methods in order to change the location of the touched piece to the place of the touched vector and is followed by condition statements that have been put in place to assign the relevant colour value to this location in the piecePosition array.

Afterwards this switch statement cases the vectorNumber variable in order to ascertain which vector has been pressed and according remove a piece that was passed over (if a piece was passed over)

After at the end of the method the win check method is called.

```
-(void)winCheck
{
    bool whiteOnBoard = false;
    bool blackOnBoard = false;
    for(int index=0;index<7;index++)
    {
        for(int jindex=0;jindex<7;jindex++)
        {
            if(piecePosition[index][jindex] == 1)
            {
                whiteOnBoard = true;
            }
            else if(piecePosition[index][jindex] == 2)
            {
                blackOnBoard = true;
            }
        }
    }
    if(((whiteOnBoard)&&(!blackOnBoard))||((blackOnBoard)&&(!whiteOnBoard)))
    {
        SKScene *gameOverScene = [[GameOverScene alloc] initWithSize:self.size];
        SKTransition *fade = [SKTransition fadeWithDuration:0.5];
        [self.view presentScene:gameOverScene transition:fade];
    }
}
```

This is the method definition for the winCheck method. For the beginning of this method two boolean variable and declared and defined as well as containing a set of nested for loops in able to cycle through its contents for every grid cell on the board. If this finds a white checker in one of the grid cells it changes the whiteOnBoard variable to true, the same setup also exists so that is this finds a black checker in one of the grid cells it changes the blackOnBoard variable to true. These variables are used to check is only one type of checker exists on the board, if this condition is met the application is redirected to the game over scene.

```

-(CheckerPiece *)findPieceX:(int)x Y:(int)y
{
    CheckerPiece *foundPiece;
    for(int index = 0; index < 12; index++) //Once for each array index
    {
        if([[whitePieces[index] getXPosition] == x] && [[whitePieces[index] getYPosition] == y]) //If the current position matches
            the desired coordinates
        {
            foundPiece = whitePieces[index]; //Set found piece
            break; //Stop looping
        }
        if([[blackPieces[index] getXPosition] == x] && [[blackPieces[index] getYPosition] == y])
        {
            foundPiece = blackPieces[index];
            break;
        }
    }
    return foundPiece;
}

```

This findPiece method uses methods and variables provided by the CheckerPiece class in order to function. This method loops one for each index value in the array and for each of these values it will use conditional statements to check if the current position being scanned matches the desired coordinates, it does this for both the whitePieces array and the blackPieces array, if these conditions are met it sets the found piece to the relevant found value.

```

-(void)update:(CFTimeInterval)currentTime
{
    /* Called before each frame is rendered */
    for(int i = 0; i < 12; i++)
    {
        //Set new position of all white pieces
        double xpos, ypos;
        xpos = gridCell[i].position.x + (115 * [whitePieces[i] getXPosition]);
        ypos = gridCell[i].position.y + (115 * [whitePieces[i] getYPosition]);
        whitePieces[i].position = CGPointMake(xpos, ypos);

        //And all black pieces
        xpos = gridCell[i].position.x + (115 * [blackPieces[i] getXPosition]);
        ypos = gridCell[i].position.y + (115 * [blackPieces[i] getYPosition]);
        blackPieces[i].position = CGPointMake(xpos, ypos);
    }
}
@end

```

This is the update method provided by Xcode's sprite kit template and loops through all the checker pieces in order to render the sprites in the correct locations. It calculates this needed value through the formula:

$xpos = gridCell[1].position.x + (115 * [whitePieces[i] getXPosition]);$

$ypos = gridCell[1].position.y + (115 * [whitePieces[i] getYPosition]);$

$whitePieces[i].position = CGPointMake(xpos, ypos);$

And a similar formula is repeated for the other array.

```
#import <Foundation/Foundation.h>

@interface Vector : NSObject
{
    int xposition, yposition;
}

- (id)initWithX:(int)x Y:(int)y;
- (int)getXPosition;
- (int)getYPosition;
@end
```

This is the header value for the before mentioned Vector class in which the methods and variables shown in the vector class diagram are defined.

```
#import "Vector.h"

@interface Vector ()
@end

@implementation Vector

- (id)initWithX:(int)x Y:(int)y
{
    if([self = [super init]])
    {
        xposition = x;
        yposition = y;
    }

    return self;
}

- (int)getXPosition
{
    return xposition;
}

- (int)getYPosition
{
    return yposition;
}

@end
```

This is the implementation file for that same class and contains the definition for the initWithX:(int)x Y:(int)y method as well as the definitions for the getXPosition and getYPosition methods.

```
#import <SpriteKit/SpriteKit.h>

@interface CheckerPiece : SKNode
{
    int xposition, yposition;
}

@property(readonly) bool colour;
-(id)initWithColour:(bool)colour;
-(void)setPositionX:(int)x Y:(int)y;
-(int)getXPosition;
-(int)getYPosition;

@end
```

This is the header file for the CheckerPiece class in which the integer variables xposition and yposition are declared as well as declaration for the initWithColour, setPosition, getXPosition and getYPosition methods.

```
#import "CheckerPiece.h"

@interface CheckerPiece ()
{
    SKSpriteNode *pieceSprite;
}

@end

@implementation CheckerPiece

-(id)initWithColour:(bool)colour
{
    if(self = [super init])
    {
        _colour = colour;

        if(!_colour)
        {
            pieceSprite = [SKSpriteNode spriteNodeWithImageNamed:@"WhitePiece"];
        }
        else
        {
            pieceSprite = [SKSpriteNode spriteNodeWithImageNamed:@"BlackPiece"];
        }

        [self addChild:pieceSprite];
    }
    return self;
}

-(void)setPositionX:(int)x Y:(int)y
{
    xposition = x;
    yposition = y;
}

-(int)getXPosition
{
    return xposition;
}

-(int)getYPosition
{
    return yposition;
}

@end
```

This is the implementation file for this same class and is used to declare the pieceSprite SKSpriteNode as well as add context and definition to the before mentioned methods that were declared in the header of this class.

```

#import "MenuScene.h"
#import "GameScene.h"
#import "RulesScreen.h"

@interface MenuScene()

@property BOOL contentCreated;

@end

@implementation MenuScene

-(void)didMoveToView:(SKView *)view
{
    if (!self.contentCreated)
    {
        [self createSceneContents]; //Separate method so it can be called again later
        self.contentCreated = YES;
    }
}

-(void)createSceneContents
{
    //Background
    SKSpriteNode *backgroundImage = [SKSpriteNode spriteNodeWithImageNamed:@"Menu Background"];
    backgroundImage.position = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY(self.frame)); //Centers the sprite
    backgroundImage.size = self.frame.size;
    [self addChild:backgroundImage];
    self.scaleMode = SKSceneScaleModeAspectFit;

    //Title
    SKSpriteNode *titleImage = [SKSpriteNode spriteNodeWithImageNamed:@"TitleImage"];
    titleImage.position = CGPointMake(CGRectGetMidX(self.frame), 150);
    titleImage.zPosition = 2;
    titleImage.xScale = 1.2;
    titleImage.yScale = 1.2;
    [self addChild:titleImage];
    self.scaleMode = SKSceneScaleModeAspectFit;

    [self addChild:[self StartButtonNode]];
    [self addChild:[self RulesButtonNode]];
    [self addChild:[self QuitButtonNode]];
}

-(SKSpriteNode *)StartButtonNode
{
    //Start Button
    SKSpriteNode *startButton = [SKSpriteNode spriteNodeWithImageNamed:@"StartGameButton"];
    startButton.position = CGPointMake(CGRectGetMidX(self.frame)+1, 950);
    startButton.zPosition = 2;
    startButton.xScale = 1.2;
    startButton.yScale = 1.2;
    startButton.name = @"StartButtonNode";
    self.scaleMode = SKSceneScaleModeAspectFit;
    return startButton;
}

-(SKSpriteNode *)RulesButtonNode
{
    //How to play button
    SKSpriteNode *rulesButton = [SKSpriteNode spriteNodeWithImageNamed:@"RulesButton"];
    rulesButton.position = CGPointMake(CGRectGetMidX(self.frame)-2, 700);
    rulesButton.zPosition = 2;
    rulesButton.xScale = 1.2;
    rulesButton.yScale = 1.2;
    rulesButton.name = @"RulesButtonNode";
    self.scaleMode = SKSceneScaleModeAspectFit;
    return rulesButton;
}

-(SKSpriteNode *)QuitButtonNode
{
    //Quit Button
    SKSpriteNode *quitButton = [SKSpriteNode spriteNodeWithImageNamed:@"QuitButton"];
    quitButton.position = CGPointMake(CGRectGetMidX(self.frame), 420);
    quitButton.zPosition = 2;
    quitButton.xScale = 1.2;
    quitButton.yScale = 1.2;
    quitButton.name = @"QuitButtonNode";
    self.scaleMode = SKSceneScaleModeAspectFit;
    return quitButton;
}

```

This (and the previous page) is the code extracted from the start menu game scene put in place to add a more simple and user friendly hub for the application in order to help remove confusion and difficulty caused by unclear instruction.

```
#import "RulesScreen.h"
#import "MenuScene.h"

@interface RulesScreen()
@property BOOL contentCreated;
@end

@implementation RulesScreen

-(void)didMoveToView:(SKView *)view
{
    if (!self.contentCreated)
    {
        [self createSceneContents]; //Separate method so it can be called again later
        self.contentCreated = YES;
    }
}

-(void)createSceneContents
{
    //Background
    SKSpriteNode *backgroundImage = [SKSpriteNode spriteNodeWithImageNamed:@"CheckersBackground"];
    backgroundImage.position = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY(self.frame)); //Centers the sprite
    backgroundImage.size = self.frame.size;
    [self addChild:backgroundImage];
    self.scaleMode = SKSceneScaleModeAspectFit;

    //Title
    SKLabelNode *titleLabel = [SKLabelNode labelNodeWithFontNamed:@"Helvetica"];
    titleLabel.text = @"How to Play";
    titleLabel.fontSize = 100;
    titleLabel.position = CGPointMake(CGRectGetMidX(self.frame), 150);
    titleLabel.zPosition = 2;
    [self addChild:titleLabel];

    //Body Text
    SKSpriteNode *bodyNode = [SKSpriteNode spriteNodeWithImageNamed:@"RulesBodyText"];
    bodyNode.position = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY(self.frame)); //Centers the sprite
    bodyNode.zPosition = 2;
    [self addChild:bodyNode];

    [self addChild:[self MenuButtonNode]];
}

-(SKSpriteNode *)MenuButtonNode
{
    //Return to menu
    SKSpriteNode *menuButton = [SKSpriteNode spriteNodeWithImageNamed:@"ReturnButton"];
    menuButton.position = CGPointMake(CGRectGetMidX(self.frame)-2, 500);
    menuButton.zPosition = 2;
    menuButton.xScale = 1.2;
    menuButton.yScale = 1.2;
    menuButton.name = @"MenuButtonNode";
    self.scaleMode = SKSceneScaleModeAspectFit;
    return menuButton;
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];

    if ([node.name isEqualToString:@"MenuButtonNode"])
    {
        SKScene *menuScene = [[MenuScene alloc] initWithSize:self.size];
        SKTransition *fade = [SKTransition fadeWithDuration:0.5];
        [self.view presentScene:menuScene transition:fade];
    }
}

@end
```

This is the implementation file linked to the rules screen accessible from the start menu of the application and simply creates three images and a label and displays them on the screen, one of the images also functions as a button to return to the menu screen.


```

#import "GameOverScene.h"
#import "MenuScene.h"

@interface GameOverScene()
@property BOOL contentCreated;
@end

@implementation GameOverScene

-(void)didMoveToView:(SKView *)view
{
    if (!self.contentCreated)
    {
        [self createSceneContents]; //Separate method so it can be called again later
        self.contentCreated = YES;
    }
}

-(void)createSceneContents
{
    //Background
    SKSpriteNode *backgroundImage = [SKSpriteNode spriteNodeWithImageNamed:@"CheckersBackground"];
    backgroundImage.position = CGPointMake(CGRectGetMidX(self.frame), CGRectGetMidY(self.frame)); //Centers the sprite
    backgroundImage.size = self.frame.size;
    [self addChild:backgroundImage];
    self.scaleMode = SKSceneScaleModeAspectFit;

    //Game Over
    SKLabelNode *titleLabel = [SKLabelNode labelNodeWithFontNamed:@"Helvetica"];
    titleLabel.text = @"Game Over";
    titleLabel.fontSize = 100;
    titleLabel.position = CGPointMake(CGRectGetMidX(self.frame), 150);
    titleLabel.zPosition = 2;
    [self addChild:titleLabel];

    [self addChild:[self MenuButtonNode]];
}

-(SKSpriteNode *)MenuButtonNode
{
    //Return to menu
    SKSpriteNode *menuButton = [SKSpriteNode spriteNodeWithImageNamed:@"ReturnButton"];
    menuButton.position = CGPointMake(CGRectGetMidX(self.frame)-2, 500);
    menuButton.zPosition = 2;
    menuButton.xScale = 1.2;
    menuButton.yScale = 1.2;
    menuButton.name = @"MenuButtonNode";
    self.scaleMode = SKSceneScaleModeAspectFit;
    return menuButton;
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    UITouch *touch = [touches anyObject];
    CGPoint location = [touch locationInNode:self];
    SKNode *node = [self nodeAtPoint:location];

    if ([node.name isEqualToString:@"MenuButtonNode"])
    {
        SKScene *menuScene = [[MenuScene alloc] initWithSize:self.size];
        SKTransition *fade = [SKTransition fadeWithDuration:0.5];
        [self.view presentScene:menuScene transition:fade];
    }
}

```

This is the implementation file for the game over scene that is accessible when the winCheck condition is met within the game scene. This scene simply consists of a single label message and a button to return to the start menu.

Testing

Function Being tested	Method of Input	Input	Expected Result	Actual Result
Changing Scene	Pressing button	Rules Button Pressed	Scene changes from Menu Scene to Rules Screen	Nothing *fix create sprite as separate method instead of inside create scene contents method
		Menu Button Pressed	Scene changes from Rules Screen to Menu Screen	Scene changes from Rules Screen to Menu Screen
		Start Button Pressed	Scene changes from Menu Scene to Game Scene	Scene changes from Menu Scene to Game Scene
	Win Condition Met	last white piece 'taken'	Scene changes from Game Scene to Game Over Screen	Scene changes from Game Scene to Game Over Screen
	Pressing Button	Menu Button Pressed	Scene changes from Game Over Screen to Menu Screen	Scene changes from Game Over Screen to Menu Screen
Gameplay	Press Piece and Vector	Clicked adjacent top left vector	Piece is moved to the adjacent top left piece	Piece moved to space [0][0] *fix create find piece method and move piece separate to piece removal
		Clicked adjacent top right vector	Piece is moved to the adjacent top right piece	Piece is moved to the adjacent top right piece
		Clicked adjacent bottom left	Piece is moved to the adjacent bottom left piece	Piece is moved to the adjacent bottom left piece
		Clicked adjacent bottom right	Piece is moved to the adjacent bottom right piece	Piece is moved to the adjacent bottom right piece

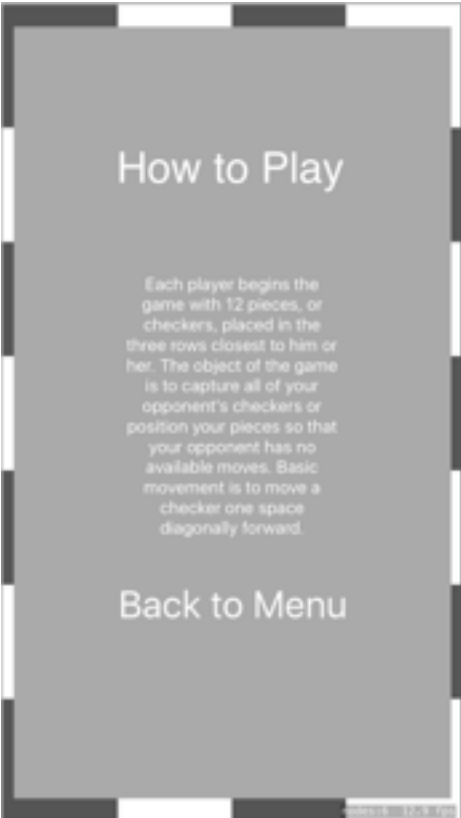
		Clicked top left vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed	Piece is moved to the vectors place. *fix correct spelling in find piece method
		Clicked top right vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed	Piece is moved to the vectors place and enemy piece that is hopped is removed
		Clicked bottom left vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed	Piece is moved to the vectors place and enemy piece that is hopped is removed
		Clicked bottom right vector on the other side of an enemy piece	Piece is moved to the vectors place and enemy piece that is hopped is removed	Piece is moved to the vectors place and enemy piece that is hopped is removed

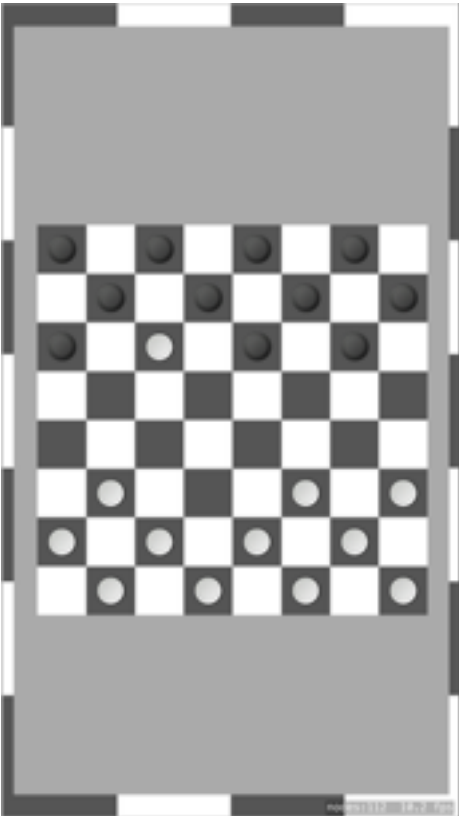
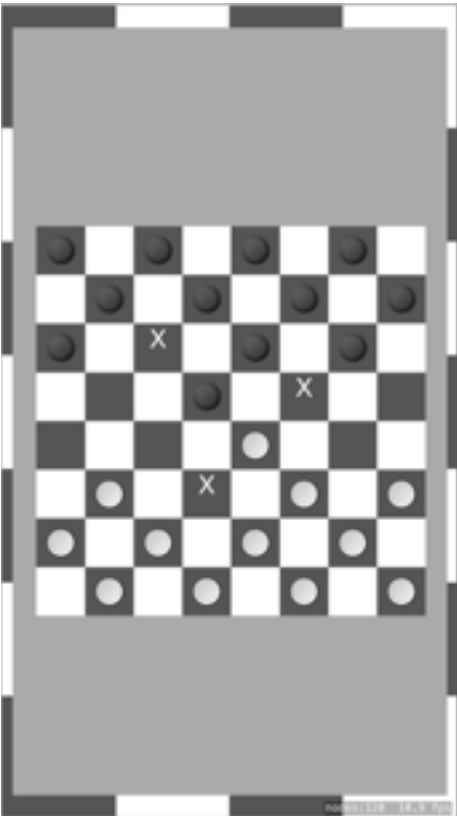
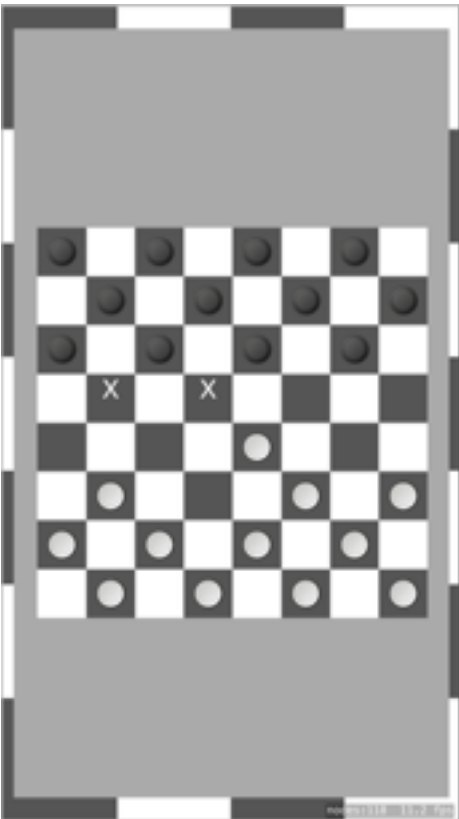
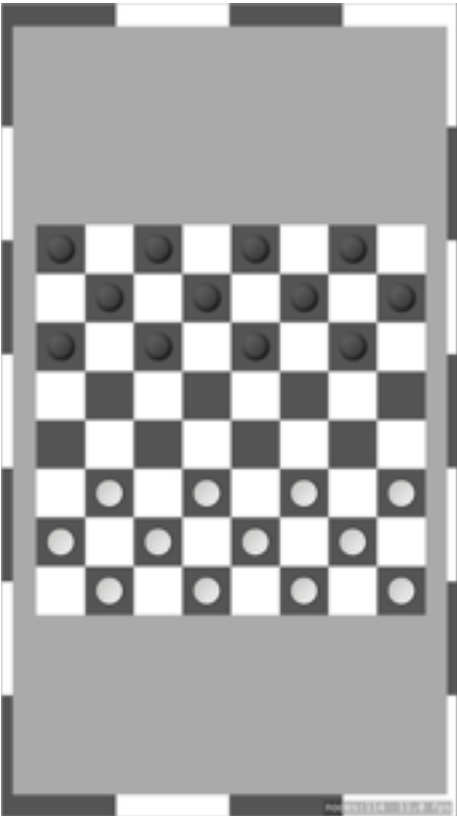
User Testing

Question number	Question	Response Y/N
1	Did all of the buttons on the start menu work	Y (5) N (0)
2	Did the button on the rules menu work	Y (5) N (0)
3	Did the game play as expected	Y (4) N (1) - Redirected to game over scene before win condition was met
4	Did the application provide enough instruction	Y (2) N (3) - Instructions could be made more in depth
5	Did the application fit the expected screen size	Y (5) N (0)

I dealt with the response to question 4 by asking the questions what they thought needed to be included within the How to Play section and they returned that perhaps it would be a good idea on which sections of the screen are intractable by the user or perhaps an image for how to move a checker piece. Due to my main target audience being students ages 16 - 18 i have decided to take no action against this remark as the difficulty of understanding may cause the game to be less appealing to younger users helping to keep my primary target audience at the decided upon point.

Screenshots of the working program





Evaluation

In conclusion of this project i do believe that i have managed to meet the criteria of my initial objectives and to accurately market the application to the stated target audience. The program appears to function in accordance with the rules of checkers as well as provide a relatively friendly user experience. My statement about the program being in accordance with the rules of checkers is only partially true as the feature of being able to king a checker piece was never properly implemented and as a result was omitted from the final program

If i were to repeat this project i would place a higher focus on efficient use of accessible code so that it may be increasingly feasible to implement any requested or desired rules and features to the game.

A desired extension that was deemed unfeasible was the implementation of a settings page that could allow the user to alter various game features such as sound when playing, or a turn timer, this feature was left out of the current application in order to comply with deadlines due to my incomplete technical knowledge of objective c specific syntax as well as the lack of audio assets available to me.