**Part A: Final Project Description and Documentation**

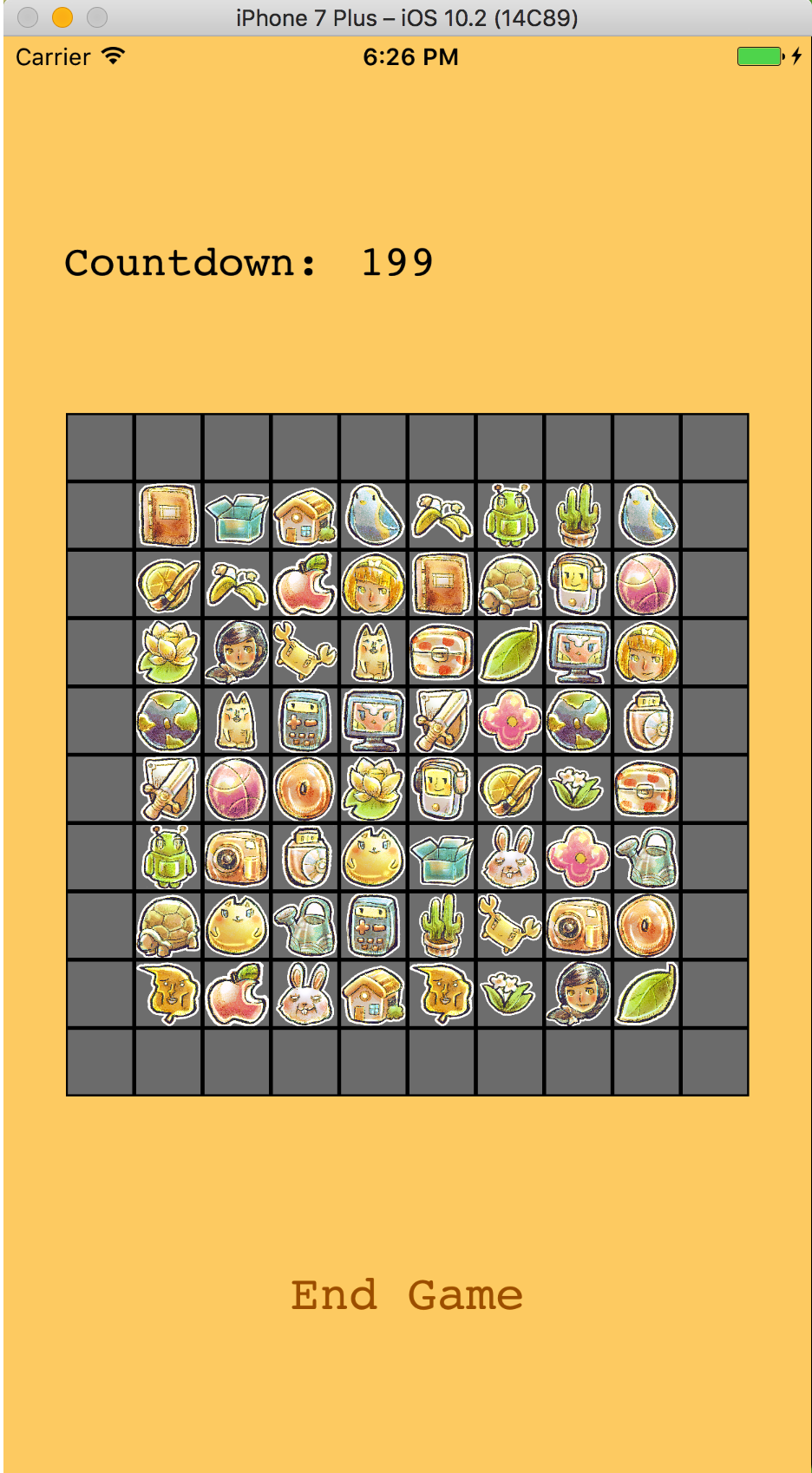
**Title:** Link the Squares

**Name:** Qianru Wu

**Type:** Individual project

**Description:**

It is a game application. The target is to remove all squares with images on the screen within a certain amount of countdown. The playground for this game is a 1010 board, which is separated into two parts. One part is squares with 88 = 64 images (called "image squares" thereafter), and the other part is the "path" with gray background (called "path squares" thereafter). Every image square is assigned a picture, and for any image square, there is one and only one image square that has the same image as itself. All squares have the same size. See the figure below:



The play can remove image squares with the help of the path squares by the following rules:

1. The player must remove only two image squares at a time.

2. To remove squares, the player first selects one image square using one finger, and drag-move to another image square with the same picture. The moving finger must falls in either target image squares or any path squares. Once a move is successful, two image squares will be removed and become two path squares.

3. At any time, if the players finger falls in different image squares or outside the playground, the image square that he choose will not be removed.

4. The player wins when he remove all the image squares while the countdown has not reached to 0. The player loses when the countdown reaches 0 but there is at least one image square (technically, two image squares, because the player removes two at a time) left on the board.

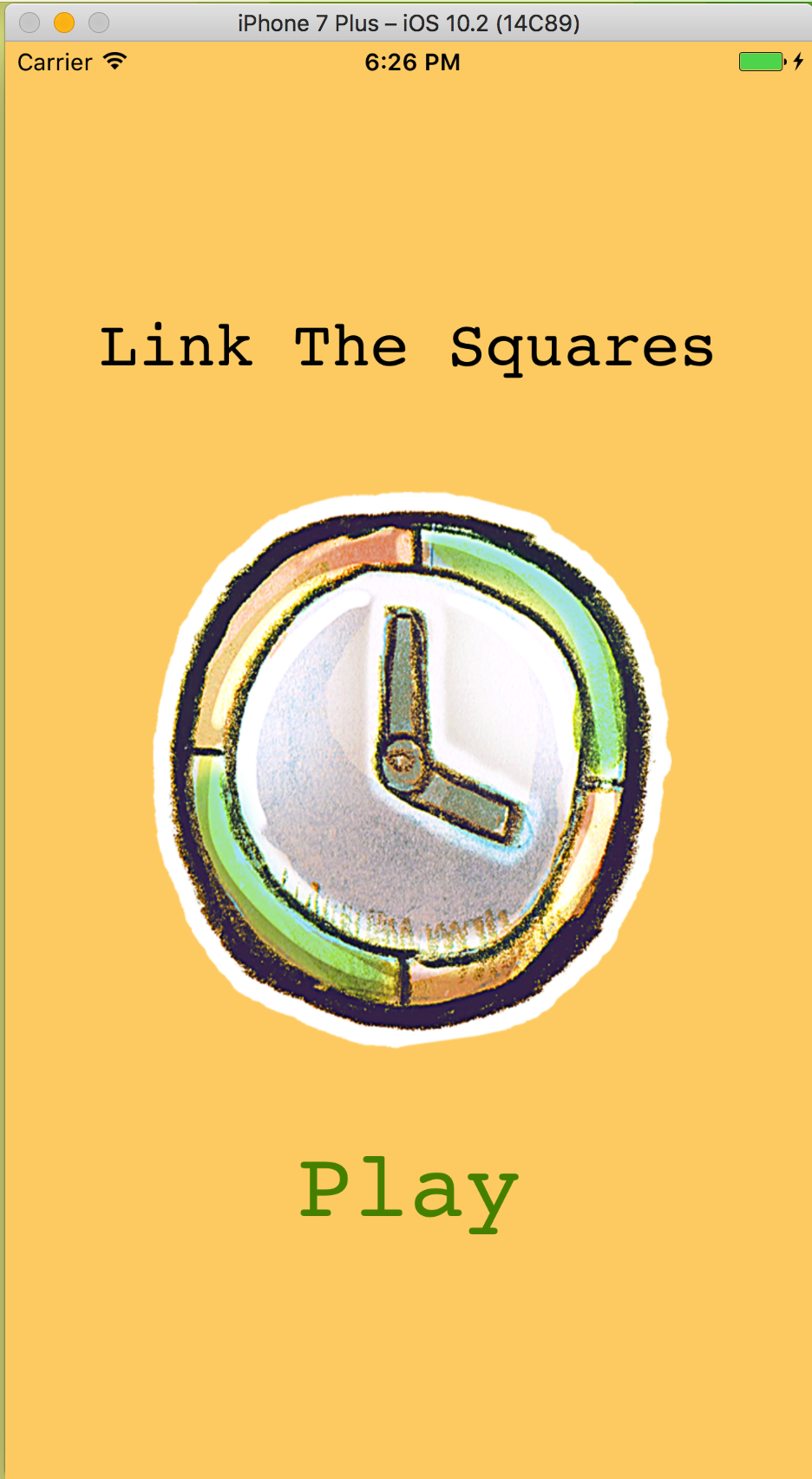
This figure shows the playground while the player is playing:



When the player wins or loses, the application will automatically jump to another screen of win or lose, like these:

Besides, there is a welcome screen before level selection:



**Acknowledgment:** Thank the artist Teekatas Suwannakrua ([http://raindropmemory.deviantart.com](http://raindropmemory.deviantart.com/" \t "_blank)) for providing icons, which are from an open source website IconAchive (http://www.iconarchive.com/).

**Part B: Final Project Discussion**

**API Implement:**

1. Quartz 2D drawing: draw playground on the view

(1) Programmatically draw the squares on the view.

a. Download images (icons), name them in a sequence, and put them in Assets. There are 32 images and one "gray" image, which is to represent the path squares.

b. Define the initial point, which is the left-upper point of the board, and the size of the squares (35 points). The initial point is defined according to the screen size, so the board lies in the center of the screen. Using the position of the initial point and side length, we can easily deduct the frame for every square.

c. Put indices (numbered 1-32) of the images into an array, and then put another 1...32 into the same array, because every image needs to be duplicated. Shuffle the index array. Then add in the index for the path square, which defined as -1. This new array is called "allIndex" in the program.

d. Use this newly constructed array as a guide to draw the squares. Path squares should be in the position where the array element value is -1, and image squares should be in the position where the array element value is the same as indicated in their names. For example, suppose the 25th element in the "allIndex" has the value 6. We know from 25 that this square is in the row 3 and column 6 of the board. Therefore, its frame starts from (initial\_x + 5 \* side\_length, initial\_y + 2 \* side\_length), and its height and width are both one side length.

e. Draw properties, like border and background color, to the image. Enable user interactions and multiple touches.

(2) Update squares on the board programmatically, assuming the player made a successful move. If a move fails, there will be no image update.

a. Identify in which squares the first and last gestures are, by calculating the distance of them to the initial point.

b. Calculate the frames of the first and last squares, and draw a "gray" image on the calculated frame.

c. Update the "allIndex" array of the first and last squares, and set their element values to -1.

2. Multi-touch events: play with one finger

Override four touch event functions in the program to handle the touch event.

(1) In the function touchesBegan, track the first touch and save it a class property called "first". Before any touch, assume the following move will be success, by setting a property "perform" to true. Doing this will ensure that it goes back to the initial state after a failure move. Then check if the player starts from the image square or not. If the play starts from the path squares or outside the board, then this move is defined failure.

(2) In the function touchesMoved, track the immediate position of the touch. Then check if the touch is within any image square whose image is not as same as the first image. If at any time, the finger moves into such a different image square, this move is defined failure. If at any time the player moves out of the board, it is also defined failure.

(3) In the function touchesEnded, track the last point and its indicated square. Since we have filtered out the situation that the player moves into the different image square, we must not repeat it here. The other two failure checks must be made in this function. One is to identify whether the last touch falls in the same square as the first one. The other is to check if the last touch is in the path square.

Filtering out all defined failure situations, we can have the successful condition, which is starting from A, moving within A or A' (A' id A's twin square) or path square, and stopping at A'. If after these filters, the "perform" property is true, then this move is successful. After a successful move, the images and the "allIndex" array will be updated.

3. NSTimer: keep track of time

This class is to create timers. The timer object sends message to the window when certain time interval has met. In the application, timer is used to determine if a player loses. The timer is set up in the viewDidLoad function, and it represented by the countdown numbers in a label. The countdown numbers updates every one second. When the countdown number reaches 0, a "lose" segue will automatically shown.

4. UIKit and Foundation: basic views and displays

Extend an array method: shuffle.

**Challenges:**

The major challenge is to check whether a move is successful or not. Omit some the criterions, and put them in one touch function at first. Solve it by drawing a table listing all success and failure situations, and then separating criterions to different touch functions. Have to better understand what are the three touch functions for.

**Limitations of the application:**

I only make a basic version of this game, comparing to the proposal. There is a situation that although there are image squares left in the board, there is no path to link them. At this time, the squares should shuffle again, or the player is definitely losing the game. The ideal version should check that, but I didn't figure out the algorithm. This is a must-in feature that will be added into the application in the future.

Another feature that mentioned in the proposal is to have an upgraded version of the game that the images can go to the four directions after a successful move, and the player can choose before the game starting which direction he wants to go. However, this feature is omitted in this version because of the time. But it's fun to add them in the future.

**Limitations of the iOS SDK and Xcode:**

It doesn't provide a shuffle method. I have to do some online study to figure it out. It needs an extra extension encoded by ourselves.

**Overall experience:**

It's my first mobile development experience, so there is no comparison between other mobile development SDK. However, comparing to other languages, Swift is more light-weighted than Java and more functional than C. As for the view part, it is also very convenient and straightforward, like adding an outlet, action, or segue. What you see is what you will get. I really enjoy working with Swift and Xcode.