Complements

Documentation

Final Project
iOS programming 2014
Professor: Nathan Hull

Group Members: Jason Leung, James Philips, Kevin Yao

Introduction to Complements

Our group, Jason, James and Kevin, was formed one day in class when professor Nathan Hull asked the class: who doesn't have a group yet?

So as a newly formed group, we talked about our common interests and passion, and found one thing that we truly shared – games.

Tracing from our childhood to college life, we studied the games that stood out to us as being remarkable – and we figured out that the best games *may not have the best graphics, or the most crazy game play*.

Instead, good games are about delivering a *unique experience* to the user. So we decided to create Compliments – a puzzle game that is a carefully designed experience for the player.

Executive Summary of Documentation

As of every project, this one was full of hurdles and choices – which really are the most interesting parts of the project. In this documentation, we would like the highlight the interesting things we encountered and how we tried to resolve it.

Our project can be logically divided into three main components, and we will go through in the following order:

1. Game Design and Implementation
2. Visual Design and Implementation
3. Other Stuff

Finally, we will go over a list of techniques we used in our project and our work distribution.

A Quick Guide Complements

When you open the game, you'll see a title screen with three options. We'd recommend going to Options first and inputting a name, so your scores will show up on our High Scores table. From there, you can also control the sound settings, if you'd like.

Next, we'd recommend playing a round, since the High Scores table will be empty at the moment. The controls are simple. Just swipe the screen to send a wave of colored dots cascading across the board. Primary colors will mix into secondary colors, and secondary colors will mix into whatever primary those colors share. If the colors are complementary, they will for a white dot, each of which is worth 10 points. The game will end when no moves are possible.

At the end of the game, the High Score table will be displayed, including your most recent score.

Game Design

Game design is an integral part of our project. While our end product is just a simple game, the choices and variables we could have picked from is huge.

For example, we had to decide between:
1. Should white dots disappear when they are merged, or should they stay?
2. Should there be a timer to "pressure" players to move, if so, how long should it last?
3. Should the user know multiple colors in advance, or just one color
4. Will players need a color wheel at the bottom to guide them which colors match and mix?

To decide between these variables, we had to have lots of versions of the game to iterate through and play test them. After careful deliberation, we decided that iOS is not the best platform for rapid prototyping and play testing, mainly because it was messy to maintain lots of different versions of projects, compiling and switching between projects were a hassle, and generally it took more effort to visualize things quickly.

Our solutions was to build a first version of the game using HTML5 and JavaScript, keep prototyping in our browser, and decide on the rules we would use going forward.

Lesson #1: We learned that, especially for simple games, iOS may not be the best platform for rapid prototyping and optimizing – by prototyping it in other lighter weight platform and porting it over, we saved a lot of time.

For the actually code implementation, the game logic is nothing super complicated or hard. However, we do take pride in having well-organized code and using a lot of helper functions and abstractions to make the code for the game logic relatively easier to understand and go through.

Visual Design and Implementation

A huge emphasis of our project is the visual design. We didn't just want it to look good – we wanted it look *great*. And more importantly, we wanted the design to complement the gameplay.

We referenced a lot of other apps and games to get inspiration for our design. Examples would be:
Candy Crush Saga – how the central theme of the design should be clear
2048 – how the UI and "swiping" motion can be made to feel responsive
Threes – minimalistic design like ours, with a large emphasis on typography/ font selection

Moreover, in our first iterations, using bright sharp colors, the game just looked *terrible*. By toning down the saturation of the colors, and choosing colors from pre-selected online color palettes, we believe we have now came up with something much more visually appealing.

Lesson #2: Design is all about inspiration. There are plenty of apps that are great design inspiration for us, and many online resources (eg color palettes) that helped us a lot.

One of the most technically challenging parts of our project was implementing the animations into the game. The core iPhone library provided us with many powerful graphics features, and we decided that it was sufficient for our project.

The problems we had to solve were mainly:

1. How to draw and animate the ball?
2. How to deal with user input? How do we buffer the animations so that the animation runs correctly even when the player input multiple moves before one animation is complete?

The basic idea behind the graphics are as follows:

1. Each **Ball** is a separate **CALayer** that is stored in a 2D array. We choose CALayer over UIView, because it is much more light weight.
2. Each **Move** is a collection of **Animation+Ball** pairs that are grouped together in a **CATransaction** so they are animated together.
3. Those **CATransactions** are implemented into an **Animation Queue** so they are executed one-by-one and don't overlap.

The main work-flow of our animation works as follows:

Registering Moves:

1. Swiping action is registered.
2. We generate the animation steps required. For example:
    1. Create 3 orange balls at the bottom.
    2. More 3 orange balls from bottom to row 4, 5, and 5 respectively (balls can move to different rows in the same move)
    3. After ball reaches location, animate color to white.
3. Each **Step** above is grouped together in an array of **Animation+Ball object** and put into the animation queue.
4. Call Execute Moves, which is detailed below

Execute Moves:

1. If there is **no animation going on** and **animation queue is not empty**
    1. Mark as **animation in progress**

2. Pop out an array of **Animation+Ball object**
3. Group of these animations in a **CATransaction**
4. Execute the **CATransaction**
5. When **CATransaction** ends, mark as **no animation in progress** and “Execute Moves” again

Lesson #3: Making clean and smooth animations is certainly hard than it looks! Luckily, iPhone comes with lots of drawing and animation tools (CALayer, CAAnimations, CATranscations) that made our life much easier!

Other Stuff

There are a lot of other neat and thought-out things we did in our project that we would like to mention.

Audio

We used an AVAudioPlayer for our background music, and a SystemSound for our sound effect, both pretty standard implementations. However, we also have an options screen that lets the user mute the background music, the sounds, or both. To do this, we write the user's preferences to memory, so they're preserved both across views and the next time the user opens the game. We also keep track of whether the player has opened the game before, and if not, make sure that the audio defaults to on.

Game Logic and Graphics Separation

After some thought, we decided to completely separate the game logic and graphics. Although in the end we put everything in the same class – logically, they are completely separate. Each move calls a game logic function, which then calls an independent graphics function.

The benefits of this approach is:

1. Game logic/ graphic components can be completely revamped without affecting the other side.
2. Save time by developing graphics section and game logic section in parallel
3. Much easier to debug each component and check it's correctness compared to a combined version.

Lesson #4: By keeping things logically modular, we can update/change things easier, and develop things in parallel.

General Code Design and Organization

We tried to maintain the code we had as well as possible:

1. Using a lot of Objects to keep things organized and maintainable
2. Comments, comments and comments
3. Using lots of helper functions so that the core logic is super readable

We really found that keeping the code well-organized and commented helped our collaboration a lot and saved us time in the long run.

The Big List of Techniques We Used and Learned

Game Logic:

1. Use of **Objects** to store game Board, and Timers etc
2. Use of iOS **Timers** to count time
3. Generating **Random Numbers** to come up with next move
4. Extending **NSMutableArray** so that it acts as a **Queue**

Visual Design

1. Use of **CALayers** to render simple shapes (cirlces and game board)
2. Extensive use of **CAAnimation** to move things around
3. Use of **CATransaction** to group animations
4. Loading and using **Custom Fonts**, which had to be done in code and not in GUI
5. Various iPhone components: **UIView, UITableView, Buttons, TIMERBAR and FLIPBUTTON**

Other

1. Saving and Loading **Persistent Data** for use in the high-score table and audio settings
2. Playing **Sound Files**
3. Moving between **UIViews** in code, and **passing data across them**

Work Distribution

As a group, we were all heavily involved in most processes together and helped each other. But the general work distribution/'who was in charge of what' would be as follows:

Game Design (how the game is played and what are the rules)
Porting/Implementing Game from Prototype to Objective C
Music and Sound Effects
James Philips

Game Prototype (rapid prototyping in JS + HTML5 for testing)
Visual Design, Layout and Font Designs
Implementation of Graphics and Animation
Jason Leung

Navigation and View Flow of Application
Prototyping with and Testing Different Graphic Libraries
Saving and Display Scores in High-Score Table
Kevin Yao

Presentations and Documentation
Collaboration

Finally...

We would like to end our Complements documentation with a compliment – thank you a lot professor and TA for this course – it was certainly one of the best CS courses we've taken!

We learned a lot from doing this project and really gained confidence in our application development skills.

We are still in discussions some final touch ups we want to do before we releasing Complements into the wild – and when we do, we will hope you will enjoy Complements as much as we have.