Mickey's Memory Game:

A children's memory game designed for Sifteo cubes

A Senior Project
presented to
the Faculty of the Computer Engineering Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science

by

Anjelica Concepcion

March 2013

© 2013 Anjelica Concepcion

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank...

Dr. Oliver for being my advisor and allowing me to take on the experience of Sifteo game development. I think I've taken 19 units with you in the past two quarters... are you sick of me yet?

Ken Ugo for being my Sifteo partner this year and for staying to help me debug, even when your parking meter was up.

Karina Cordon and Dereck Quock for setting an example with your previous work in Sifteo development. You have both helped me so much in the past several years and I'm so grateful that the support didn't stop once you left Cal Poly!

Mickey boy for being the cutest, most loveable dog ever and for being the game mascot. Ant for all your feedback after play testing this game over and over. And a huge thank you to Dwe Dwe and my entire family for the endless support!

Enjoy the game!

Abstract

Mickey's Memory Game is a children's memory game developed on the Sifteo platform. Sifteo cubes are small, interactive devices, which can display colored images and sense neighboring cubes or detect user actions, such as clicking, shaking, or tilting the cubes. Although many classic video games or mobile games can be recreated on the Sifteo cubes, the goal of this project was to utilize capabilities that are unique to the Sifteo gaming platform, while also creating a useful game for child development.

Introduction

Problem Statement

Attention-Deficit/Hyperactivity disorder, or ADHD, is one of the most common mental disorders affecting children and adolescents. According to the Centers for Disease Control and Prevention (CDC), nine percent of American children ages 3-17 have been diagnosed with ADHD, which is about 5 million children [1]. In surveys taken since 1999, there has been a clear upward trend in the number of parent-reported ADHD diagnoses, as well as an increasing number of FDA-approved ADHD medications [2]. While treating ADHD has become a big focus for many parents and doctors, it is important to also consider methods of prevention for this disorder.

Solution

Working memory is a type of memory that involves holding recent information in your mind while being able to actively manipulate it. Active working memory is used for things like following directions or solving math problems in your head. Studies have shown that there is a correlation between children with ADHD and children with a working memory deficit [3]. By regularly exercising a child's working memory, it is likely that you are increasing their ability to selectively enhance or ignore the environment around them. Memory games are a popular method of improving a child's working memory, as it presents the learning experience to them in a fun, desirable manner. The more a child enjoys playing the memory game, the more effective it can be as a development tool to prevent or lessen the effects of ADHD.

Game Proposal

I had two main criteria when trying to decide the concept of my game: utilize as many of the unique Sifteo capabilities as possible and design a game that targets child development. I downloaded several games available through the Sifteo store and played them to get an idea of what I felt worked best and worst about these games. I found that, due to the small size of the Sifteo cubes, the most successful games were the ones with simplistic graphics and little to no text.

I liked the idea of creating a follow-the-leader game, which involved displaying a particular action and then requiring the player to perform the action. It would be similar to the kids' game "Bop It" where players are told to either "bop it", "flick it", "pull it", or "twist it". The Sifteo commands would instead use "shake it", "flip it", "click it" and "tilt it". To develop this into a memory game, I chose to implement a repeating chain of commands for the player to follow. The game starts with one random action command. Once the player correctly performs the action, the second round starts and repeats the first command, then adds a second command. For each successful round the player completes, a new command is added onto the chain and the sequence becomes harder to remember. The game ends when the player performs the wrong action or performs an action on the wrong cube.

The mascot for this game is a cartoon Maltese puppy. He is based on my own dog, Mickey, and serves as a friendly face for children.

Background

The original Sifteo cubes were an interactive gaming platform released in September 2011. Created by Sifteo, Inc., these blocks have a length and width of 1.5 inches and a height of about .5 inches. They each have a clickable full-color LCD screen, a 3-axis accelerometer, and near-field object sensing technology. Sifteo cubes were sold in sets of 3 or 6 cubes and came with a charging dock, which also served as a protective carrying case. To use the original cubes, it was necessary to have either a Mac or PC, as well as the ability to download the Sifteo software. Games could be purchased and downloaded through the online Sifteo store and then installed onto cubes using a USB connector. The computer application Siftrunner was used to run games on the set of cubes.

In November 2012, Sifteo, Inc. released the second generation of Sifteo cubes, known as the Sifteo Cubes Interactive Game System. The new system came with a base unit that stored games available to the cubes and had speakers for game audio, thus eliminating the use of a computer when running games. The base unit can still sync with the computer via USB cable so that new games can be installed from the desktop software. The second generation also removed the need for a charging station by powering the base and the cubes with AAA batteries, as opposed to the Lithium Polymer batteries used in the original Sifteo cubes. Sifteo, Inc. also released a new SDK for use with the new system.

Figure 1 demonstrates the different possible moves a player can perform on either version of Sifteo cubes. Although the new Sifteo cubes are functionally similar to the original ones, they cannot be used together. The original Sifteo cubes will not sync with the new system's base unit, and the new cubes cannot run games through Siftrunner. Also, games designed on the old SDK will not work with the new cubes. The original Sifteo cubes can support gameplay with up to 6 cubes, while the new system doubles that amount and can support 12 cubes. Mickey's Memory Game was designed for the original Sifteo cubes.

Each original Sifteo cube contains:
32-bit ARM CPU
128x128 color TFT LCD
3-axis accelerometer
8MB Flash
Lithium Polymer rechargeable battery
2.4 GHz wireless radio
Sifteo's near field object sensing technology

Figure 1: Illustration of Sifteo's sensing capabilities [4]

Requirements

System Requirements

In order to run the Sifteo desktop software, computers must meet the following requirements:

Windows
2.0 GHz Intel Pentium 4 or faster processor
Windows XP SP3 with 512 MB of RAM, or
Vista/Windows 7 with 1GB of RAM

Mac
1.5 GHz or faster Intel Core processor
Leopard 10.5 or Snow Leopard 10.6 with 1GB of RAM

General
1024x768 or larger display
Available USB 2.0 port
200 MB disk space (500 MB recommended)
Internet connection for software download and setup

Developing games for Sifteo also requires the use of an IDE, such as MonoDevelop, which is designed for C# and other .NET languages. Downloading the original Sifteo SDK provides the developer with a C# API, four demo programs, and documentation for the API and demos.

Game Requirements

The game should be designed for children ages 7 and up. The difficulty level should also account for the target age audience. The game should utilize all (or as many as possible) of the cube's interactive abilities, including shaking, flipping, tilting, button pressing, and "neighboring". The graphics of the game should make use of the full-color LCD on each cube. Any text that appears in the game should be concise, easy to read, and easy to comprehend. The game should also demonstrate the use of audio in a meaningful way (i.e. indicating when a player makes a correct move). The game can require a minimum of 3 cubes, but it must be capable of supporting up to 6 cubes.

Learning Requirements

One of the big ideas behind the creation of Sifteo cubes was for them to be used as a unique learning tool for children. The cubes combine characteristics of classic hands-on children games, like jigsaw puzzles and Legos, with modern technology that children are becoming more interested in playing with, like video games or iPad apps. Sifteo games center around the idea of "intelligent play", which focuses on several areas of thinking skills. Games also fall into a number of categories, which often cross fun with learning.

Mickey's Memory Game primarily targets the following thinking areas:

- *Patterns and Perception* - the game requires focus to remember a pattern that is constantly getting longer
- *Strategy and Planning* - there are many different approaches to pattern memorization and as the pattern gets more complicated, players must quickly adjust their memorization techniques to make them more manageable and effective
- *Cooperation and Collaboration* - this is a game that can become easier when children collaborate and help each other; if one player forgets the next move, there is a greater chance that someone else can still recall it

Spatial Reasoning	Understand relationships between objects in terms of distance and orientation
Logic and Computation	Build awareness of associations between numbers
Language and Literacy	Comprehend and use words, story elements, and literary devices
Cooperation and Collaboration	Build and refine social skills to solve problems with other people
Expression and Emotion	Explore and understand emotions and how they are expressed
Strategy and Planning	Hone high-level thinking and problem solving skills
Creativity and Design	Build and create all kinds of compositions
Patterns and Perception	Sort and classify to fine-tune your ability to spot trends and patterns

Table 1: List of Sifteo's Thinking Skills with descriptions [4]

Mickey's Memory Game falls under the following game genres:

- *Learning* - teaches children how to focus their attention and follow directions as shown
- *Puzzle* - allows users to make decisions on how to approach their pattern memorization
- *Social* - users may benefit from playing in a group or they can add a competitive edge to the game by comparing high scores

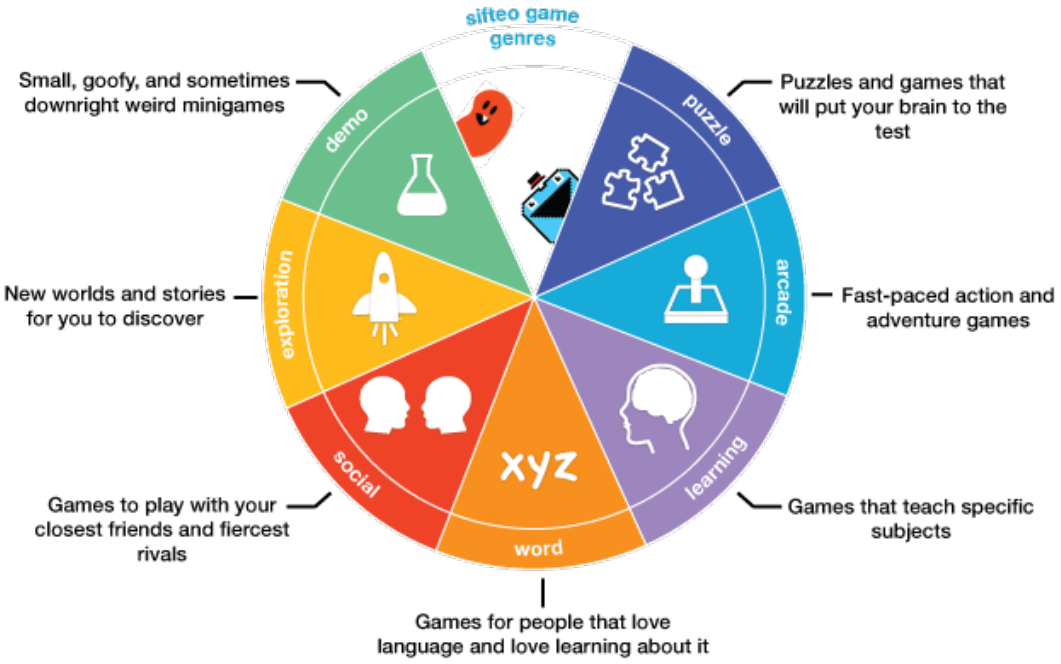


Figure 2: Sifteo Game Genres [4]

Design

Figure 3 shows a top-level view of how the user interacts with the cubes and how the cubes communicate with the computer. The cube set comes with a USB link that allows Siftrunner to detect available cubes, using Bluetooth. Figure 4 looks further into components within the computer and Sifteo cubes. The computer uses MonoDevelop, Siftrunner, and speakers. The cubes contain a 32-bit ARM CPU, a 128x128 color TFT LCD, a 3-axis accelerometer, 8MB Flash memory, Lithium Polymer rechargeable battery, 2.4 GHz wireless radio, and near field object sensing technology. In order to play a game, you must run the program in MonoDevelop, load the game into Siftrunner, then select and play the game with the computer's volume on.

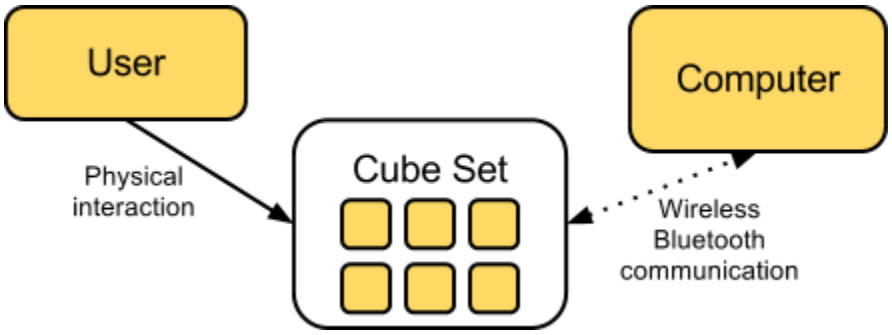


Figure 3: Top level connections

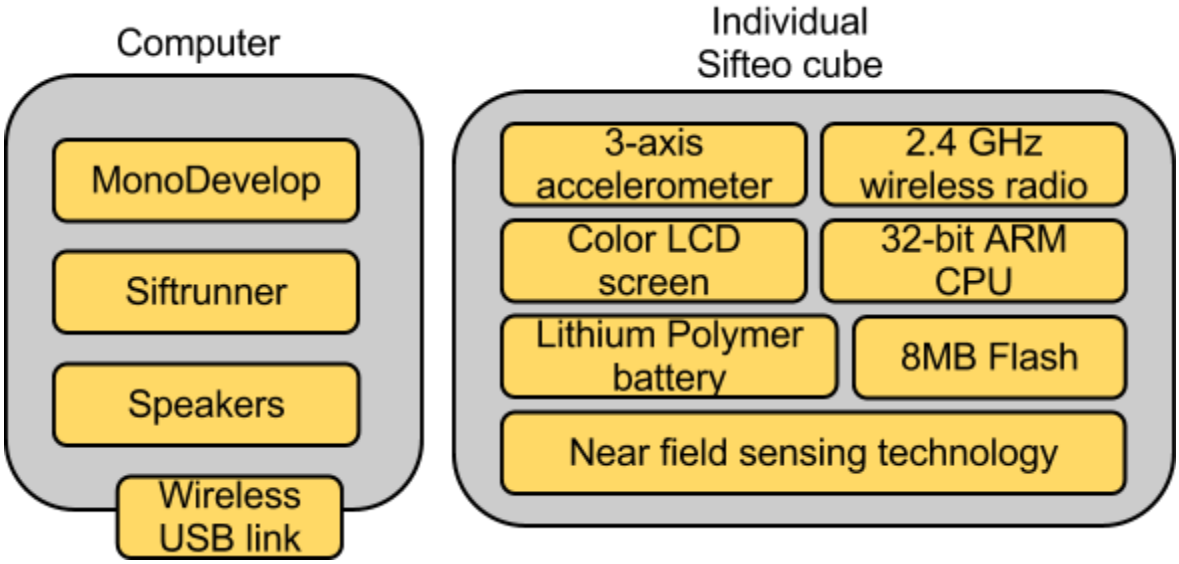


Figure 4: Close up look at individual components

Figure 5 is a state diagram that shows the game flow, as designed in the software. Upon start up, three cubes are chosen to display the three images that make up the title screen. When the user places them next to each other in a way that forms the title "Mickey's Memory Game", the state moves to the "pattern state". The pattern state is used to display the current action sequence that the user needs to reenact. Once it is done displaying the sequence, it moves to the game state where it is the users turn to create the same sequence. If the player gets it right, the state moves back to the pattern state, which displays the same sequence and then adds one new action to the end. When a player makes an error in the game state, the score screen is shown for a few seconds and then the state returns to the title screen, which the user must again put in order.

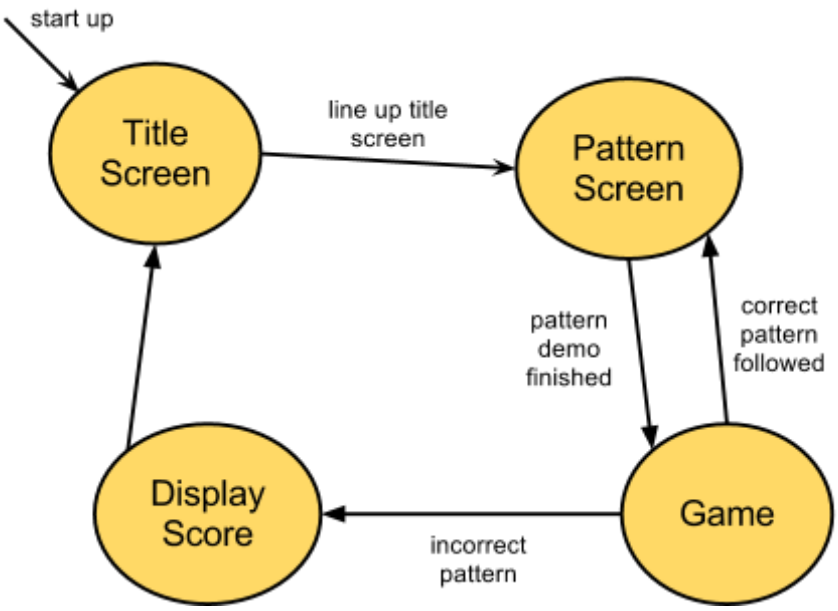


Figure 5: State flow diagram

Development

The development process was not as clear-cut as I had imagined. I had expected the game functionality to be the focus of development, but I could not have predicted how much time I would spend debugging game graphics. My main areas of development involved functionality, graphics, scoring, and audio. Additional examples of what I found helpful throughout the development process can be found in Appendix C.

Functionality

I found that the best way to start developing my game was to first try running and editing one of the existing Sifteo demo applications. The SlideShow app is a game that displays images and manipulates the graphics based on user interaction. After exploring the existing code, I grasped a better idea of how the Sifteo API worked and how event handlers were used to perform actions specific to user interaction. From there, my game development progressed through 3 distinct stages.

Stage 1:

This version of the game was done using one class, known as the base app. At this stage, the entire cube set was programmed to display the same image upon start up. Each of the cubes was set up with event handlers. The first image displayed the command “shake”. If a shake event was caught on a cube that was displaying the shake image, the

image on that particular cube would advance to a new image that said "click". Once that cube was clicked, it would advance to a "flip" command. If a click event was caught on a cube displaying "shake", nothing would happen. This stage tested my ability to verify that an action matches up with the image on the given cube.

Stage 2:

The next stage involved selecting one cube, at random, to display the action image. Once the right action was done on the right cube, another random cube would display the next random action image. This stage lead to the usage of a cube variable and and action variable that were set with a randomizer and then compared against in the event handler functions. This was also the first version that incorporated state controllers. However, upon moving the PaintImage() calls from the BaseApp to a state controller, I ran into new errors that stated my images could not be found. The imaging issue was the biggest setback in my entire project, and due to limited resources for Sifteo development, I was unable to find any explanation as to why my images stopped working.

To continue development without running into imaging issues, I removed all PaintImage() calls and used a combination of Log.Debug() messages and ClearScreen() calls. ClearScreen() is a function in Sean Voisen's CubePainter class, which allows you to paint a cube's screen with one solid color. In testing, I used yellow to represent "click", pink to represent "flip", and blue to represent "shake".

Stage 3:

The final version of this game required me to separate the main game into two different states: one to display the sequence and one to listen for events from the user. This version involved passing a list of [action, cube] pairs between the state controllers. The TitleController was used to reset the list, the PatternController displayed the list and added to it, and the GameController made comparisons between the list and user input.

Graphics

Once the game play was functioning correctly, it was time to fix the lack of graphics. The Image Helper tool is a feature of SiftDev, which converts user images into a Sifteo-friendly format. To convert a single file, the user clicks "Select File" and chooses the file from

its directory. After selecting the file, the Image Helper displays it on the left side, under “Original”. Once you click “Convert”, the converted image will appear on the right side, under “Converted”, as shown in Figure 6. However, when I followed this procedure with graphics that I had created myself, clicking the “Convert” button would yield no resulting image on the right side, indicating that the conversions were not successful. At this point, I set aside the goal of using images and instead learned a technique that I saw used in the Sifteo demo program, HelloWorldApp.

HelloWorldApp demonstrated a way to display the words “Hello World” on each cube by simply coloring pixels. FillRect() is a function in the Paint API, which colors in a rectangle of a specified height and width to a specified x and y location. I used FillRect() to draw block letters of different letters; for example, the letter “S” was created with 5 calls to FillRect(), using different x-y coordinates and dimensions. I created functions to print the words “shake”, “click”, and “flip”. I also used a function to progressively draw a colorful circle around the word. After successfully implementing these functions into my program, I still felt determined to get real images on my cubes. Eventually, I found that if I used existing Sifteo images that were the size of a single cube, I could edit the image into my own graphic, rename it, and successfully convert it. Although I was glad to finally have images working, I didn’t want to omit the work I had done with the pixel coloring. For my final program, I incorporated real graphics in the TitleController and GameController but left the PatternController displaying the word drawings. I also used “GO” and “WATCH” images to differentiate between when a player should be interacting with the cubes versus when they should be observing. Although the variety of graphics may feel incohesive, I believe that they served to demonstrate the flexibility in Sifteo graphics.

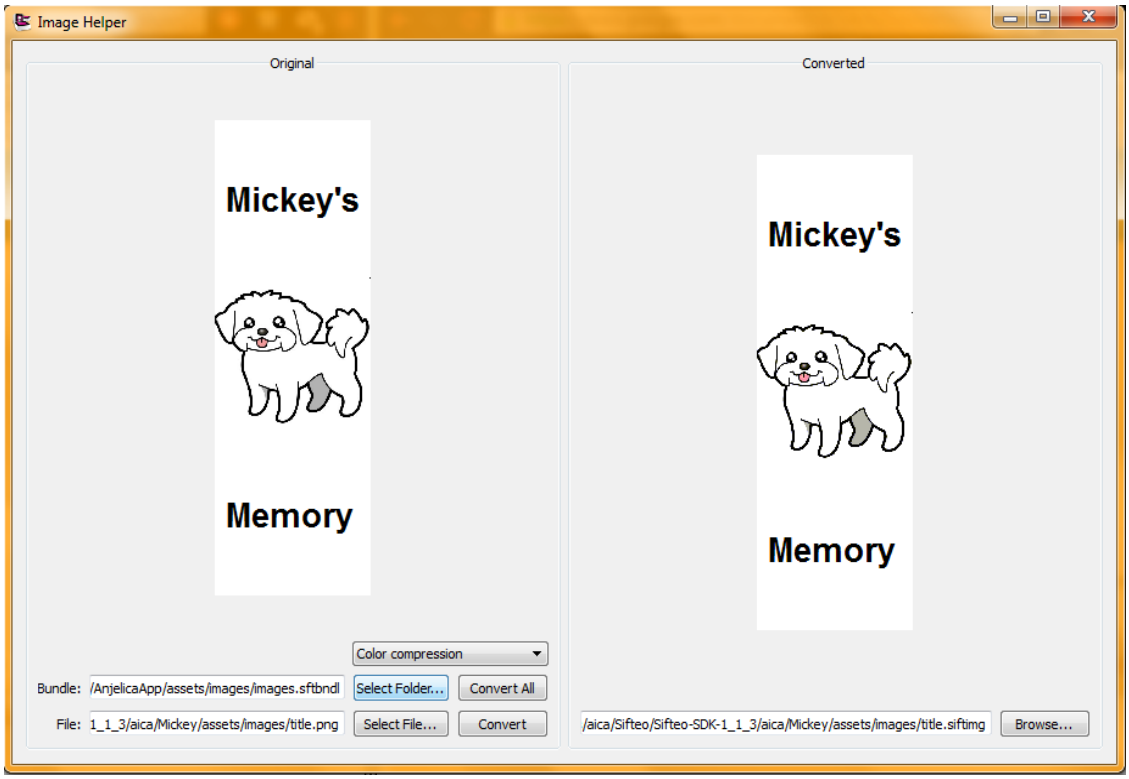


Figure 6: Sifteo's Image Helper tool with original title menu design

Figure 7: Example of graphics done by coloring pixels

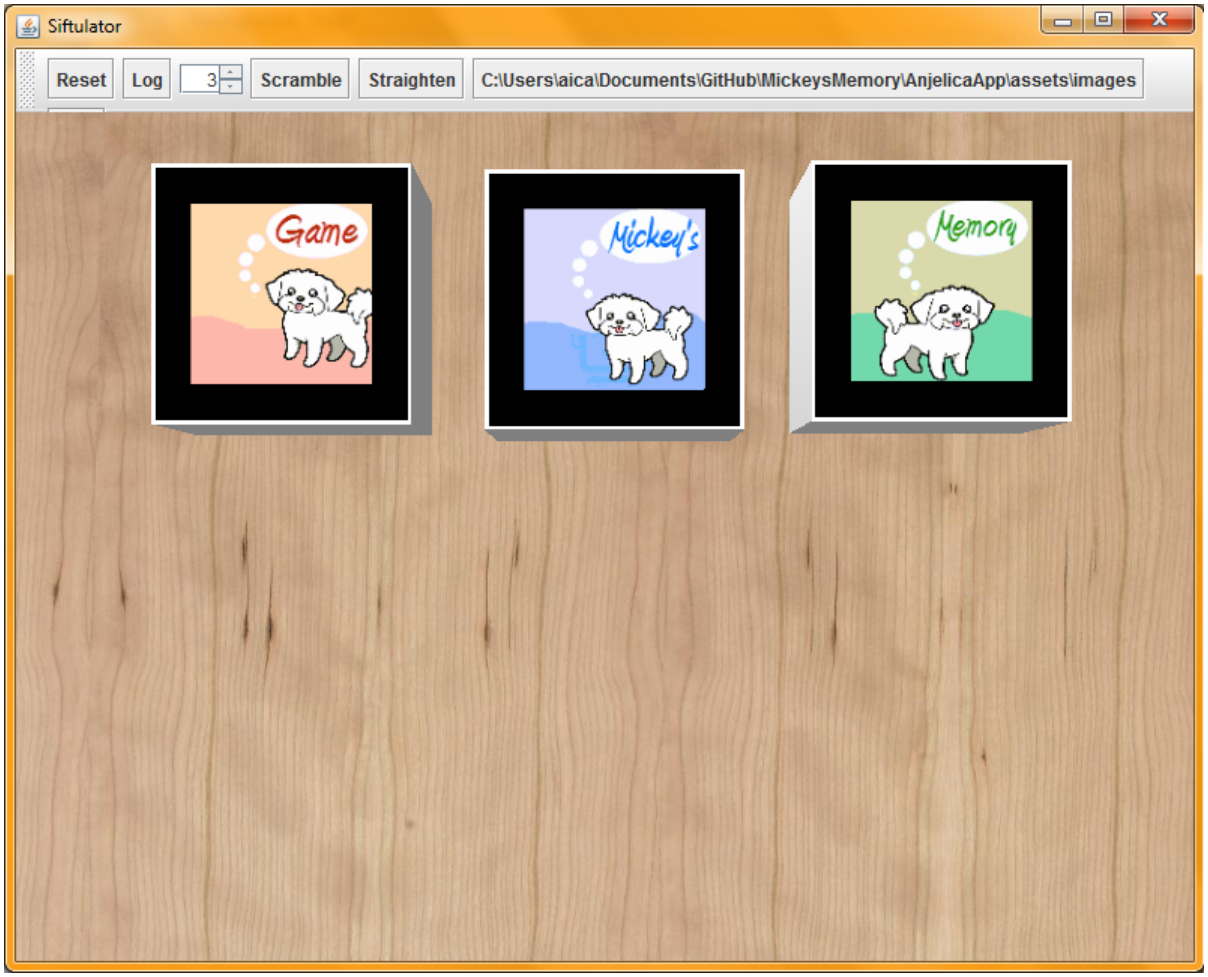


Figure 8: Screenshot of title menu images (out of order)

Figure 9: Additional images used

Displaying Scores

Scores were initially designed to be in a separate controller, but when it was time to implement it, I realized I could easily include it as part of the setup code in the TitleController. I created 17 images displaying scores. I chose to color score images in a manner that represented how well the player did. The red zone includes scores 1-5,

the yellow zone is 6-9, and the green zone is 10-17. Although it is possible to score above 18, I chose not to support images past a certain point. However, this does not cause the program to stop running. A score above 17 will simply not display a score image before the title screen, but the score can still be read on the Log console.

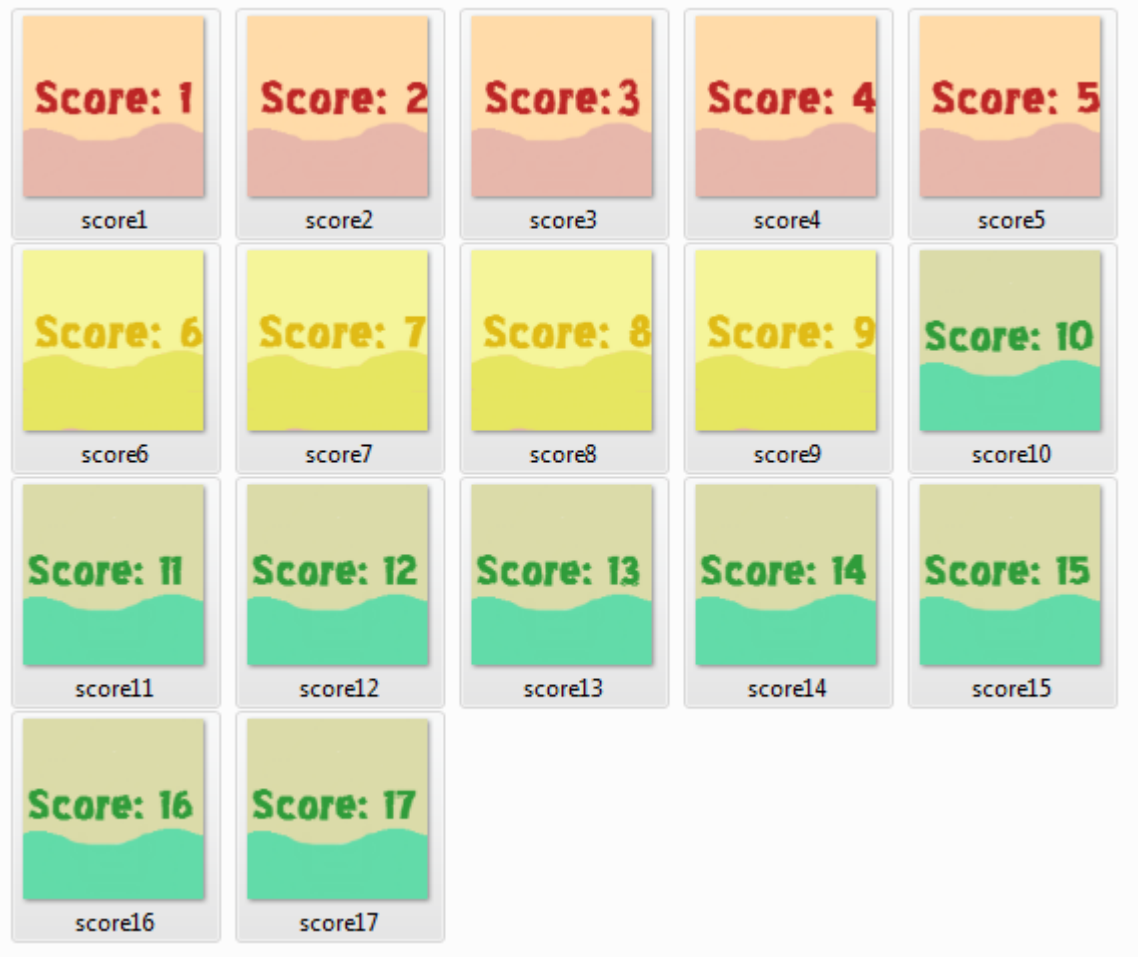


Figure 10: Score images

Audio

The final addition to my game was adding sounds. As I did not have the equipment or experience to create custom sounds for this game, I used Joshua Lee's music from Sifteo's SorterApp demo as my background music and "incorrect" sound. The "correct" sound in my game is taken from Karina Cordon's Mac's Fiesta game, which had sound engineering done by Ritchie Ly and Sven Le.

Testing

This project received ongoing testing throughout the development process and I made several significant observations along the way. The first thing to note is the availability of two testing platforms. Sifteo applications can be tested on either a physical set of Sifteo cubes or on Sifteo's own cube simulation program, Siftulator. Mickey's Memory Game was tested on both and I found it beneficial to do so. Secondly, it was important to have a distinct set of test cases for each state controller. Each state is responsible for executing a unique part of the game, so it is important to know what you are expecting to see when a game is in a given state. Lastly, there were two areas of my game that did not function perfectly when tested. I was not able to get tilting and timing to work in an ideal way, so they remain as areas for future improvement.

Siftulator vs. Sifteo Cubes

Siftulator is an application included in the Sifteo SDK, which allows developers to simulate cube functionality on the computer screen. It was an incredibly convenient tool, because it eliminated the need to bring the physical cube set around. Whenever changes are made to an application, Siftulator must install the updated game onto the physical cubes before enabling play. For bigger applications with a lot of data, this can take several minutes. In testing, it was very useful to have changes visible in Siftulator within seconds.

However, there were a few downfalls to using Siftulator. I only used the physical cube set in the first and final weeks of my project. I ran into an incident toward the end where I was testing on both the physical cubes and Siftulator, and I found that the Siftulator was displaying images incorrectly. The application was displaying old images that I had already erased from my project files. Although I couldn't figure out why this bug was occurring, it lead me to believe that the image problems I had faced throughout development may have been issues with Siftulator, rather than with my code or image files. For this reason, I would recommend that others test on both platforms regularly. Also, when using Siftulator, it can be easy to forget some of the basic rules for running games. The Siftulator folder must always point to the /assets/images directory of the running program. If you are running a program that does not correspond to the Siftulator's image folder, you will receive no warning. Another rule

when developing is that whenever a change is made to the assets folder of a project (i.e. adding a sound file, renaming an image), the game must be reloaded into Siftdev by going to the Developer menu and selecting “Load Apps”. To load an app, you must select the folder that holds the assets folder, otherwise it will not show up in your “my games” section.

Test Cases

The following tables list the cases which each of the three state controllers were tested for:

Category	**Expected Behavior**
Display	On setup, three title images display on three random cubes
Functionality	Neighboring three title cubes in correct order queues transition to next state
Audio	Background sound begins playing on setup and loops infinitely, even in other controllers
Score	When transitioning from GameController, display accurate score

Table 2: Test cases for TitleController

Category	**Expected Behavior**
Display	Displays “WATCH” before looping through and displaying each action in the current sequence
Functionality	Ignores any user interaction; queues transition to GameController after displaying entire sequence

Table 3: Test cases for PatternController

Category	Expected Behavior
Display	Displays “GO” on setup, displays action image on cube once a user interacts with it, regardless of whether it is correct or not
Functionality	Detects whether a user has followed the correct pattern; transitions to TitleController if incorrect; transitions to PatternController if correct
Audio	Plays “correct” sound once for each correct action; plays “incorrect” sound once when wrong

Table 4: Test cases for GameController

Tilt Event

The initial prototype for Mickey’s Memory included a fourth action, “tilt”. When playtesting, the game would often end erroneously when a player was correctly executing a “flip” command. Debug messages helped me to discover that when flipping a cube, it was common for cubes to detect a tilt event. This makes sense because flipping a cube involves tilting it. To avoid compromising the accuracy of the game, I decided to remove the tilt event handler. Although this limited the game variety a bit, I believe the existing game is a worthy challenge with only three action options.

Timing

Timing was a difficult thing for me control in this project. In the GameController, there is a lot of information that must be processed whenever an event is received. The cube catches the event, checks to verify it is correct, and then calls functions to display the corresponding image for half a second and play the appropriate sound. Because they are holding a memorized pattern in their mind, players often want to zoom through the GameController quickly by performing each action in the sequence as fast as they can. The good thing is that event handlers can keep a queue of the actions that were just

performed and process them one by one. The downside is that the user will see a delay between their interactions and the cubes' responsiveness. The action queue can also backfire in events that have no set start and stop time. For example, shaking a cube for a longer-than-normal period of time could register as shaking the cube twice.

Conclusion

Designing a Sifteo game was a great way to explore a new technology in gaming. I ran into a lot of unexpected challenges and although the lack of information for Sifteo developers was frustrating, it lead to a more rewarding end result. I may have imagined that this project would be mainly about development and game design, however a great deal of my effort was spent just learning about the technology. When I first played Sifteo games, it was easy for me to criticize some of the features that didn't work well. In designing and playtesting my game, I discovered just how hard it is to make an intuitive game on a platform that's completely new to people. However, once you discover the capabilities of these small cubes, it's very exciting to see the creative ways that developers utilize them.

Although I feel that this was a successful project, I am not sure whether I believe it would be a good choice for future students. Throughout the course of my project, Sifteo, Inc. was making big changes to their product. In an area that already had little support, it seemed that I was in fact losing resources along the way. For periods of time, Sifteo web pages regarding the original Sifteo cubes would be out of service. Links previously used for the original cubes were being replaced with information about the new cubes only. Since the hardware is already available, it may still be a project of interest, but in the long run, games created for original Sifteo cubes will become outdated and unsupported.

That being said, Sifteo is still accepting developer submissions for their original Sifteo store. These games will all be offered to consumers for free, so developers will not be paid. However, Sifteo has stated that if they do make your game available on their store, they will send you a set of the new Sifteo cubes as compensation. This may be a good way to secure future Sifteo projects at a reasonable cost.

Future work:

- Eliminate timing delays in GameController so that cubes respond in real time to user interactions
- Add multi-player game modes
- Create unique sounds for each action as a way to involve more senses in the memorization process
- Animating the puppy cartoon to move throughout the blocks
- Find a way to incorporate a tilt event
- Replace all audio with custom music

Appendices

Appendix A: References

[1] Bloom B, Cohen RA, Freeman G. "Summary health statistics for U.S. children: National Health Interview Survey, 2009." National Center for Health Statistics. Vital Health Stat 10(247). December 2010. Web. <http://www.cdc.gov/nchs/data/series/sr_10/sr10_247.pdf>
[2] "CDC - ADHD, Timeline - NCBDDD." 2013. 25 Mar. 2013. Web. <http://www.cdc.gov/ncbddd/adhd/timeline.html>
[3] Diamond, Adele. "Attention-deficit disorder (attention-deficit/hyperactivity disorder without hyperactivity): A neurobiologically and behaviorally distinct disorder from attention-deficit/hyperactivity disorder (with hyperactivity)." *Development and psychopathology* 17.3 (2005): 807. Web. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1474811/>
[4] Sifteo, Inc. Web. <https://www.sifteo.com/>
[5] Cordon, Karina. "Mac's Fiesta: A Foreign Language Game for the Sifteo Platform." June 2012. Web. <http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1065&context=cpesp>

Appendix B: Senior Project Analysis

Summary of Functional Requirements
My project is a children's game played with 3 or more Sifteo cube. It starts with a title menu, which the player must place in order to start the game. During the "watch" phase, the game shows a random action name on a random cube. Then, during the "go" phase, the player performs that action on the correct cube. The game switches back and forth between "watch" and "go" states, each time adding a new action and making the memory sequence longer for the player. When the player gets the action or cube wrong, their score is shown and the game resets to the title menu.

Primary Constraints

Most of my issues were regarding graphics for this project. I was never able to convert my own images into a Sifteo-friendly format. I also had problems with my IDE updating to a version that crashed upon being opened. I was never able to redownload a working copy of MonoDevelop and spent my last few weeks on the project developing on a trial version of Visual Studio. Whenever new problems occurred, the biggest difficulty was the limited number of resources available for Sifteo development.

Economic

The cost of the 6 Sifteo cube set is about $250. For this project, no additional money was spent, as the cubes were previously purchased for another project. The Sifteo software is free to download and the MonoDevelop IDE is free. If MonoDevelop is unavailable, it may cost an additional amount to download an alternate IDE, such as Visual Studio. I was able to develop on a trial version of Visual Studio for the end of this project. The span of this project was 22 weeks, with approximately 3-8 hours of research or development done each week.

Environmental

The biggest environmental impact related to this project is the need to keep the batteries for all 6 cubes charged. Also, the game cannot be played with the cubes alone, it always requires the use of a computer to run the game.

Manufacturability

No manufacturing was done for this project.

Sustainability

The entire project can easily be passed on to others who wish to continue the development of the game. As long as someone has a set of Sifteo cubes and a computer that can run the Sifteo software, they can play the game. However, a newer Sifteo SDK was released in October 2012, which supports C++ projects instead of C#. Although the website still says that the old SDK is available for download, it does not seem to be supported anymore.

Ethical

Many video games that kids are exposed to involve violence or other negative exposures. This project utilizes cheery audio and colorful cartoon images to create a positive experience for users.

Health and Safety

As with many electronics, using this game for long periods of time could cause strain on the eyes and possibly the hands. It is important to take breaks when engaging in game play.

Social and Political

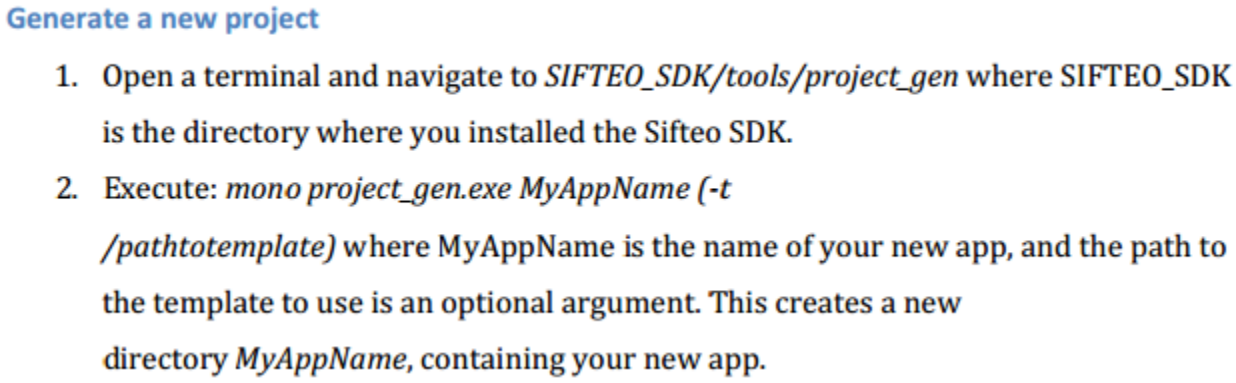
This game can be used as a fun social activity. Although there is no "multiple player" mode, many players can work together to play the same game, or players can compete to get the highest score between them.

Development

This project was my first exposure to coding in C# and developing an MVC project. I also learned a lot from using the Sifteo API. Although I had studied MVC in the past, it never really made sense until I put it into practice, and using state controllers felt like a very obvious choice for designing a Sifteo game.

Appendix C: Tips for possible issues

- In Karina Cordon's "Getting Started Guide" [5], the section under "Generate a new project" must be done from a mono command prompt:



Generate a new project

1. Open a terminal and navigate to *SIFTEO_SDK/tools/project_gen* where SIFTEO_SDK is the directory where you installed the Sifteo SDK.
2. Execute: *mono project_gen.exe MyAppName (-t /pathtotemplate)* where MyAppName is the name of your new app, and the path to the template to use is an optional argument. This creates a new directory *MyAppName*, containing your new app.

- Error when building project: "cannot access Sifteo.bll"
  - Game may be currently running in SiftDev; stop game to rebuild
- Build unsuccessful
  - Try adding Sifteo.dll to project references:
    - Right click "References", click "Edit References", go to ".NET assembly", find Sifteo.dll and click "ADD", then click "OK"
- MonoDevelop crashes repeatedly or won't open
  - Download a trial version of Visual Studio as a temporary solution

- Receiving: “A first chance exception of type 'System.NullReferenceException' occurred in [game.exe]” in visual studio:
    1. Navigate to "Debug/Exceptions"
    2. Expand the "Common Language Runtime Exceptions" tree.
    3. Expand the "System" branch.
    4. Scroll down to where "NullReferenceException" is, and check the "throw" checkbox, and uncheck the "user-handled".
    5. Debug your project.

Appendix D: Source Code

```
/* AnjeilicaApp is  the BaseApp, which is where the program begins*/
using Sifteo;
using System.Collections;
using System;
using System.Collections.Generic;
using Sifteo.Util;

namespace AnjelicaApp
{
   public class AnjelicaApp : BaseApp
  {
    private TitleController titleController;
    private PatternController patternController;
    private GameController gameController;
    private CubePainter cubePainter;
    private StateMachine sm;
    private Boolean canvasDirty;
    private List<Actions> acts = new List<Actions>();
    private Sound bgMusic;

    public String[] mImageNames;
    public Random mRandom = new Random();
    public int magicId, magicIndex;

    // Here we initialize our app.
    public override void Setup()
    {
      cubePainter = new CubePainter();

      // Creates background music Sound object
      bgMusic = Sounds.CreateSound("music");
      // Plays music at max volume and continuous loop
      bgMusic.Play(1, -1);
      SetupStateMachine();
    }



    public void SetupStateMachine()
    {
      sm = new StateMachine();
```

```
      // Initialize Controllers
      titleController = new TitleController (this.CubeSet, this.cubePainter, sm, acts);
      patternController = new PatternController(this.CubeSet, this.cubePainter, sm, acts);
      gameController = new GameController (this.CubeSet, this.cubePainter, sm, acts, Sounds);

      // Set states and transitions
      sm.State("title", titleController);
      sm.State("pattern", patternController);
      sm.State("game", gameController);

      sm.Transition("null", "nullToTitle", "title");
      sm.Transition("title", "titleToPattern", "pattern");
      sm.Transition("pattern", "patternToGame", "game");
      sm.Transition("game", "gameToTitle", "title");
      sm.Transition("game", "gameToPattern", "pattern");

      sm.SetState("title", "nullToTitle");
    }



    public override void Tick()
    {
      // Call current state's OnTick() function
      sm.CurrentState.OnTick (1);

      if (canvasDirty)
      {
        sm.Paint (canvasDirty);
        canvasDirty = false;
      }
    }
  }
}



/* Title Controller - Displays three images that make up the title menu
   The 3 image cubes must be placed in proper order to start the game */
using System;
using System.Collections;
using Sifteo;
using System.Collections.Generic;
using Sifteo.Util;

namespace AnjelicaApp
{
  public class TitleController : IStateController
  {
    private CubeSet cubeSet;
    private CubePainter cubePainter;
    private StateMachine sm;
    private StateMachineLock smLock;
    private bool found;
    private List<Actions> acts;

    public TitleController(CubeSet cubeSet, CubePainter cubePainter, StateMachine sm,
        List<Actions> acts)
    {
```

```
  this.cubeSet = cubeSet;
  this.cubePainter = cubePainter;
  this.sm = sm;
  smLock = new StateMachineLock(sm);
  this.acts = acts;
}

public void OnSetup(string transition)
{
  Log.Debug("In TitleController");
  if (transition == "gameToTitle")
  {
    Log.Debug("YOUR SCORE WAS: {0}", acts.Count-1);
    cubePainter.printScore(cubeSet, acts.Count - 1);
    cubePainter.Commit(cubeSet);
    System.Threading.Thread.Sleep(2000);
  }
  found = false;
  Paint();
  ListenForEvents();
}

public void OnTick(float n)
{
  if (sm.Current != "title")
    return;

  // Increments tick counter when state is locked
  if (smLock.Locked)
  {
    smLock.Tick();
  }
  else
  {
    if (found)
    {
      // Queues transition when cubes are placed in correct order
      sm.QueueTransition("titleToPattern");
      sm.Tick(1);
    }
  }
}
public void OnPaint (bool canvasDirty)
{
  if (canvasDirty)
  {
    Paint();
  }
}

public void OnDispose ()
{
  cubeSet.ClearEvents ();
}

  /* Private Methods */
private void ListenForEvents()
{
  cubeSet.NeighborAddEvent += NeighborAddHandler;
```

```
      cubeSet.NeighborRemoveEvent += NeighborRemoveHandler;
    }

    private void NeighborAddHandler(Cube cube1, Cube.Side side1, Cube cube2, Cube.Side side2)
    {
      Log.Debug("title cube add");
      CheckCubes();
    }

    /* CheckCubes()
    * Checks if cubes are placed in correct order
    */
    private void CheckCubes()
    {
      Cube[] row = CubeHelper.FindRow(cubeSet);
      if (row.Length == 3)
      {
        if (row[0] == cubeSet[0] &&
            row[1] == cubeSet[1] &&
            row[2] == cubeSet[2])
        {
          found = true;
          smLock.LockForTickCount(10);
        }
      }
    }

    private void NeighborRemoveHandler(Cube cube1, Cube.Side side1, Cube cube2, Cube.Side side2)
    {
      // Does nothing, but can check if program recognizes removed cubes
      // with a call to Log.Debug()
    }

    /* Paint()
    * Puts image information on specific cubes that will be used to display the title screen
    */
    private void Paint()
    {
      cubePainter.ClearScreen(cubeSet);

      cubePainter.PaintFullImage(cubeSet[0], "mickeys");
      cubePainter.PaintFullImage(cubeSet[1], "memory");
      cubePainter.PaintFullImage(cubeSet[2], "game");

      cubePainter.Commit(cubeSet);
    }
  }
}

/* PatternController - used to display the current pattern sequence */
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Sifteo;
using Sifteo.Util;

namespace AnjelicaApp
```

```
{
  public class PatternController : IStateController
  {
    private CubeSet cubeSet;
    private CubePainter cubePainter;
    private StateMachine sm;
    private StateMachineLock smLock;
    private Cube actionCube;
    private Random random = new Random();
    private int actionIndex;
    private String[] actions = new String[] {"shake", "flip", "click"};
    private List<Actions> acts;

    public PatternController (CubeSet cubeSet, CubePainter cubePainter, StateMachine sm,
                List<Actions> acts)
    {
      this.cubeSet = cubeSet;
      this.cubePainter = cubePainter;
      this.sm = sm;
      smLock = new StateMachineLock(this.sm);
      this.acts = acts;
    }

    public void OnSetup(string transition)
    {
      Log.Debug("PatternController Setup");
      /* Randomly select a cube and action */
      actionCube = cubeSet[random.Next(cubeSet.Count)];
      actionIndex = random.Next(3);

      if (transition.Equals("titleToPattern"))
      {
        // Clear actions array for new game session
        acts.Clear();
      }
      /* Add new cube/action combo to actions array */
      acts.Add(new Actions(actions[actionIndex], actionCube));
      System.Threading.Thread.Sleep(1000);
      cubePainter.PaintAllImage(cubeSet, "watch");
      cubePainter.Commit(cubeSet);
      System.Threading.Thread.Sleep(1000);
      Paint();
    }

    public void OnTick(float n)
    {
    }

    public void OnPaint(bool dirtyCanvas)
    {
      if (dirtyCanvas)
      {
        Paint();
      }
    }

    public void OnDispose()
    {
      cubeSet.ClearEvents();
```

```
    }

    public void Paint()
    {
      cubePainter.ClearScreen(cubeSet);
      foreach (Actions act in acts)
      {
        foreach (Cube cube in cubeSet)
        {
          if (!cube.Equals(act.Cube))
          {
            cubePainter.ClearScreen(cube, new Color(0, 0, 0));
          }
          else
          {
            Log.Debug("{0} me!", act.Action);
            cubePainter.PaintAction(cube, act.Action);
          }
        }
        cubePainter.Commit(cubeSet);
        /* paint each action for 1 second */
        System.Threading.Thread.Sleep(1000);
      }


      sm.QueueTransition("patternToGame");
      sm.Tick(1);
    }
  }
}


/* GameController - listens for events and checks if they are correct or incorrect*/
using System;
using System.Collections;
using Sifteo;
using System.Collections.Generic;
using Sifteo.Util;

namespace AnjelicaApp
{
  public class GameController : IStateController
  {
    private CubeSet cubeSet;
    private CubePainter cubePainter;
    private StateMachine sm;
    private StateMachineLock smLock;
    private Random random = new Random();
    private int eventCount;
    private String[] actions = new String[] {"shake", "flip", "click"};
    private List<Actions> acts;
    private Sound incorrect;
    private Sound correct;
    private SoundSet sounds;

    public GameController(CubeSet cubeSet, CubePainter cubePainter, StateMachine sm,
                List<Actions> acts, SoundSet sounds)
    {
```

```
  this.cubeSet = cubeSet;
  this.cubePainter = cubePainter;
  this.sm = sm;
  smLock = new StateMachineLock(this.sm);
  this.acts = acts;
  this.sounds = sounds;
}

public void OnSetup(string transition)
{
  Log.Debug("GameController Setup");
  /* set audio for correct and incorrect actions */
  correct = sounds.CreateSound("correct");
  incorrect = sounds.CreateSound("incorrect");
  eventCount = 0;
  cubePainter.PaintAllImage(cubeSet, "go");
  cubePainter.Commit(cubeSet);
  listenForEvents();

}

public void OnTick(float n)
{
  if (sm.Current != "game")
    return;
}

public void OnPaint(bool dirtyCanvas)
{
  if (dirtyCanvas)
  {
    //Paint();
  }
}

public void OnDispose()
{
  cubeSet.ClearEvents();
}

/* private methods */
void Paint(Cube cubey, string action)
{
  cubePainter.ClearScreen(cubeSet);
  foreach (Cube cube in cubeSet)
  {
    if (!cube.Equals(cubey))
    {
      cubePainter.ClearScreen(cube, new Color(0, 0, 0));
    }
    else
    {
        Log.Debug("you did: {0}", action);
        cubePainter.PaintFullImage(cube, action);
        System.Threading.Thread.Sleep(800);
        cube.Paint();

    }
```

```
  }
  cubePainter.Commit(cubeSet);
}

private void OnButton(Cube cube, bool pressed)
{
  if (pressed)
  {
    Log.Debug("Button pressed");
  }
  else
  {
    Log.Debug("Button released");
    Paint(cube, "click");
    if (acts[eventCount].Action.Equals("click"))
    {
      if (cube.Equals(acts[eventCount].Cube))
      {
        correct.Play(1, 0);
        Log.Debug("Correct!");
        checkEventCount();
      }
      else
      {
        Log.Debug("Wrong cube!");
        incorrect.Play(1, 0);
        sm.QueueTransition("gameToTitle");
        sm.Tick(1);
      }
    }
    else
    {
      Log.Debug("Wrong action!");
      incorrect.Play(1, 0);
      sm.QueueTransition("gameToTitle");
      sm.Tick(1);
    }
  }
}

private void OnShakeStarted(Cube cube)
{
  Log.Debug("Shake start");
  Paint(cube, "shake");
  if (acts[eventCount].Action.Equals("shake"))
  {
    if (cube.Equals(acts[eventCount].Cube))
    {
      correct.Play(1, 0);
      Log.Debug("Correct!");
      checkEventCount();
    }
    else
    {
      Log.Debug("Wrong cube!");
      incorrect.Play(1, 0);
      sm.QueueTransition("gameToTitle");
      sm.Tick(1);
    }
  }
```

```
  else
  {
    Log.Debug("Wrong action!");
    incorrect.Play(1, 0);
    sm.QueueTransition("gameToTitle");
    sm.Tick(1);
  }
}

private void OnShakeStopped(Cube cube, int duration)
{
  Log.Debug("Shake stop: {0}", duration);

}

private void OnFlip(Cube cube, bool newOrientationIsUp)
{
  if (newOrientationIsUp)
  {
    Log.Debug("Flip face up");
    Paint(cube, "flip");
    if (acts[eventCount].Action.Equals("flip"))
    {
      if (cube.Equals(acts[eventCount].Cube))
      {
        correct.Play(1, 0);
        Log.Debug("Correct!");
        checkEventCount();
      }
      else
      {
        Log.Debug("Wrong cube!");
        incorrect.Play(1, 0);
        sm.QueueTransition("gameToTitle");
        sm.Tick(1);
      }
    }
    else
    {
      Log.Debug("Wrong action!");
      incorrect.Play(1, 0);
      sm.QueueTransition("gameToTitle");
      sm.Tick(1);
    }
  }
  else
  {
    Log.Debug("Flip face down");

  }
}

private void listenForEvents()
{
  foreach (Cube cube in cubeSet)
  {
    cube.ButtonEvent += OnButton;
    cube.ShakeStartedEvent += OnShakeStarted;
    cube.ShakeStoppedEvent += OnShakeStopped;
```

```
        cube.FlipEvent += OnFlip;
      }
    }

    private void checkEventCount()
    {
      eventCount++;
      if (eventCount == acts.Count)
      {
        Log.Debug("next round!");
        sm.QueueTransition("gameToPattern");
        sm.Tick(1);
      }
    }
  }
}

/* CubePainter.cs
 * Written by Sean Voisen and with addition functions
 * by Anjelica Concepcion for Mickey's Memory
 * Used to handle image data to be displayed on cubes
 */

using System;
using Sifteo;
using Sifteo.Util;

namespace AnjelicaApp
{
  public class CubePainter
  {
    private Random mRandomizer = new Random();

    public void PaintAllImage (CubeSet cubes, String imageName)
    {
      foreach (Cube cube in cubes)
      {
        PaintFullImage (cube, imageName);
      }
    }

    public void PaintImage (Cube cube, string imageName, int x, int y, int width, int height)
    {
      cube.Image (imageName, x, y, 0, 0, width, height, 1, 0);
    }

    public void PaintSprite (Cube cube, SpriteData data, int x, int y)
    {
      cube.Image (data.imageName, x, y, data.source.x, data.source.y, data.size.x,
        data.size.y, 1, 0);
    }


    void PaintSpriteCentered (Cube cube, SpriteData data)
    {
      int x = (int)((Cube.SCREEN_WIDTH - data.size.x) * 0.5);
      int y = (int)((Cube.SCREEN_HEIGHT - data.size.y) * 0.5);
```

```
  cube.Image (data.imageName, x, y, data.source.x, data.source.y, data.size.x,
    data.size.y, 1, 0);
}

/* PaintFullImage()
* Paints 128x128 image on cube
*/
public void PaintFullImage (Cube cube, string imageName)
{
  cube.Image (imageName, 0, 0, 0, 0, Cube.SCREEN_WIDTH, Cube.SCREEN_HEIGHT, 1, 0);
}

/* ClearScreen()
* Fill cube with black color
*/
public void ClearScreen (Cube cube)
{
  cube.FillScreen (Color.Black);
}

public void ClearScreen (Cube cube, Color c)
{
  cube.FillScreen (c);
}

public void ClearScreen (CubeSet cubeSet)
{
  foreach (Cube cube in cubeSet)
  {
    ClearScreen (cube);
  }
}

/* Commit()
* Commits data stored to be painted onto cubes and displays them
*/
public void Commit (CubeSet cubeSet)
{
  foreach (Cube cube in cubeSet)
  {
    cube.Paint ();
  }
}

public void Commit (Cube cube)
{
  cube.Paint ();
}

public void PaintAction(Cube cube, String action)
{
  if (action.Equals("shake"))
  {
    DrawShake(cube);
  }
  else if (action.Equals("flip"))
  {
    DrawFlip(cube);
```

```
  }
  else if (action.Equals("click"))
  {
    DrawClick(cube);
  }
  System.Threading.Thread.Sleep(800);
  ClearScreen(cube);
  cube.Paint();
}


public void DrawColoredCircle(Cube cube)
{

  // Increment the frame counter so that we can keep track of how long
  // we've been running.
  for (int mFrameCount = 0; mFrameCount < 65; mFrameCount++)
  {

    // Pick a point on a circle based on the current frame counter. The
    // cube's screen is 128x128 pixels, so the center of our circle is at
    // (64,64), and it has a radius of 56 pixels.
    double theta = (double)mFrameCount / 30.0 * Math.PI;
    int x = 64 + (int)(56.0 * Math.Cos(theta)) - 2;
    int y = 64 + (int)(56.0 * Math.Sin(theta)) - 2;

    // Pick a random color to draw the dot.
    int r = mRandomizer.Next(256);
    int g = mRandomizer.Next(256);
    int b = mRandomizer.Next(256);
    Color color = new Color(r, g, b);

    // Draw the dot.
    cube.FillRect(color, x, y, 4, 4);

    // Remember to call Paint!
    cube.Paint();
  }
}

public void DrawFlip(Cube cube)
{
  int x = 24;
  int y = 24;
  int width = 80;
  int height = 80;
  Color color = new Color(255, 32, 255);
  cube.FillRect(color, x, y, width, height);

  // F
  cube.FillRect(Color.Black, 42, 45, 2, 32);
  cube.FillRect(Color.Black, 44, 45, 8, 2);
  cube.FillRect(Color.Black, 44, 60, 4, 2);

  // L
  cube.FillRect(Color.Black, 54, 45, 2, 32);
  cube.FillRect(Color.Black, 56, 75, 8, 2);
```

```
  // I
  cube.FillRect(Color.Black, 67, 45, 2, 32);

  // P
  cube.FillRect(Color.Black, 74, 45, 2, 32);
  cube.FillRect(Color.Black, 76, 45, 8, 2);
  cube.FillRect(Color.Black, 76, 60, 8, 2);
  cube.FillRect(Color.Black, 82, 45, 2, 15);

  DrawColoredCircle(cube);
}

public void DrawClick(Cube cube)
{
  // ### FillRect ###
  // FillRect draws a rectangle on the cube's screen at a given location
  // in a given size and color. A cube's screen is 128x128 pixels. Here
  // we draw a big square in the center of the screen.
  int x = 24;
  int y = 24;
  int width = 80;
  int height = 80;
  Color color = new Color(255, 145, 0);
  cube.FillRect(color, x, y, width, height);

  // C
  cube.FillRect(Color.Black, 35, 45, 2, 32);
  cube.FillRect(Color.Black, 37, 45, 8, 2);
  cube.FillRect(Color.Black, 37, 75, 8, 2);

  // L
  cube.FillRect(Color.Black, 49, 45, 2, 32);
  cube.FillRect(Color.Black, 51, 75, 8, 2);

  // I
  cube.FillRect(Color.Black, 62, 45, 2, 32);

  // C
  cube.FillRect(Color.Black, 69, 45, 2, 32);
  cube.FillRect(Color.Black, 71, 45, 8, 2);
  cube.FillRect(Color.Black, 71, 75, 8, 2);

  // K
  cube.FillRect(Color.Black, 83, 45, 2, 32);
  cube.FillRect(Color.Black, 85, 60, 6, 2);
  cube.FillRect(Color.Black, 91, 45, 2, 32);
  cube.FillRect(color, 91, 59, 2, 4);

  DrawColoredCircle(cube);
}

public void DrawShake(Cube cube)
{
  int x = 24;
  int y = 24;
  int width = 80;
```

```
      int height = 80;
      Color color = new Color(0, 145, 255);
      cube.FillRect(color, x, y, width, height);

      // S
      cube.FillRect(Color.Black, 28, 45, 2, 16);
      cube.FillRect(Color.Black, 36, 60, 2, 16);
      cube.FillRect(Color.Black, 30, 45, 8, 2);
      cube.FillRect(Color.Black, 28, 60, 8, 2);
      cube.FillRect(Color.Black, 28, 75, 8, 2);

      // H
      cube.FillRect(Color.Black, 42, 45, 2, 32);
      cube.FillRect(Color.Black, 44, 60, 6, 2);
      cube.FillRect(Color.Black, 50, 45, 2, 32);

      // A
      cube.FillRect(Color.Black, 56, 45, 2, 32);
      cube.FillRect(Color.Black, 58, 60, 6, 2);
      cube.FillRect(Color.Black, 58, 45, 6, 2);
      cube.FillRect(Color.Black, 64, 45, 2, 32);

      // K
      cube.FillRect(Color.Black, 71, 45, 2, 32);
      cube.FillRect(Color.Black, 73, 60, 6, 2);
      cube.FillRect(Color.Black, 79, 45, 2, 32);
      cube.FillRect(color, 79, 59, 2, 4);

      // E
      cube.FillRect(Color.Black, 86, 45, 2, 32);
      cube.FillRect(Color.Black, 88, 45, 8, 2);
      cube.FillRect(Color.Black, 88, 60, 4, 2);
      cube.FillRect(Color.Black, 88, 75, 8, 2);



      DrawColoredCircle(cube);
    }

    public void printScore(CubeSet cubes, int score)
    {
      String name = "score" + score.ToString();
      Log.Debug("score name was {0}", name);
      PaintAllImage(cubes, name);
    }

  }
}
```