Chef Sifty

a Sifteo project

by

Dereck Quock

advisor

Dr. John Oliver

Senior Project

Computer Engineering Department

California Polytechnic State University

San Luis Obispo

June 2012

Table of Contents

List of Tables and Figures

Tables

Figures

Acknowledgments

Dr. John Oliver
Dr. Jennifer Jipson
Karina Cordon
Sifteo Inc.

Abstract

In our society today, childhood obesity has become a more prevalent problem, which needs to be fixed. Children constantly eat and snack unhealthily and parents need to understand the importance of teaching children how food affects their body and well-being. Teaching this to children at an early age will help to instill a healthy lifestyle for the child. Chef Sifty is a cooking game developed for Sifteo cubes that aims to show children the importance of nutrition while having fun. Using Sifteo's concept of Intelligent Play, the objective of this project is to show children some healthy food and snack alternatives and follow the procedural directions for each recipe.

I. Introduction

I.i. Problem

According to the Centers for Disease Control and Prevention, more than one-third of adults and almost 17% of youth were obese in 2009–2010 [1]. Childhood obesity continues to be a problem in today's society and parents are not the only ones to blame. Children are not taught the importance of healthy nutrition in an interesting and interactive way. There should be an easy way to instill a healthy lifestyle for our children and we shouldn't have to force-feed them fruits and vegetables, which will turn them off to wanting to eat those types of food. It is hard to get kids to eat healthy foods, especially with the commercials on TV making fast food look delicious. Childhood obesity needs to be solved, but in a way that makes children actually want to eat healthy alternatives.

I.ii. Solution

Instead of taking your child to McDonald's or any other fast food restaurant every day, you can help them cook the meals and snacks they have learned from Chef Sifty. By having children learn by playing, the importance of healthy nutrition can be taught without the child realizing it. Dr. Bruce Perry states that "play enhances every domain of a child's development," so it seems perfect that one of Sifteo's main features is intelligent play [2]. This game is targeted for ages 8-12 and parents actually helping their child to make the food can complement the game. The recipes are designed to be healthier alternatives, but the number of recipes is limited because of the limitations of the Sifteo cubes, which will be explained further in the Background.

II. Background of Sifteo Cubes

Sifteo cubes are 1.5-inch computers with full-color LCD displays. Each cube also has a clickable screen that acts like a button, senses motion, senses another neighboring cube, and connects wirelessly to your computer. Having such a small interactive device makes Sifteo cubes perfect for all ages to play with.

II.i. Intelligent Play

Sifteo games are based on intelligent play and the different aspects of learning through playing and enhancing thinking skills. By combining learning with playing and palpable interactivity, Sifteo cubes enable users to expand and enrich their several different thinking skills.

Table 1. Sifteo Thinking Skills [3]

Thinking Skill	Description
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

The different thinking skills Chef Sifty targets to improve include:

- *Spatial Reasoning* – help children understand the relationships between food ingredients and the cookware used to make the food
- *Cooperation and Collaboration* – children can collaborate with others to finish a recipe and they can also collaborate with their parents to actually make the food

- *Creativity and Design* – although the recipes are static, there is an element of creating something from scratch and building a recipe using different ingredients to make a final product

Also included with the Sifteo cubes is the Creativity Kit, which allows you to create your own games. You can provide your own input to create any type of sorting game, from words to numbers.

As with Chef Sifty, Sifteo games utilize one or more of the thinking skills and are categorized by Sifteo as shown in Figure 1 below.

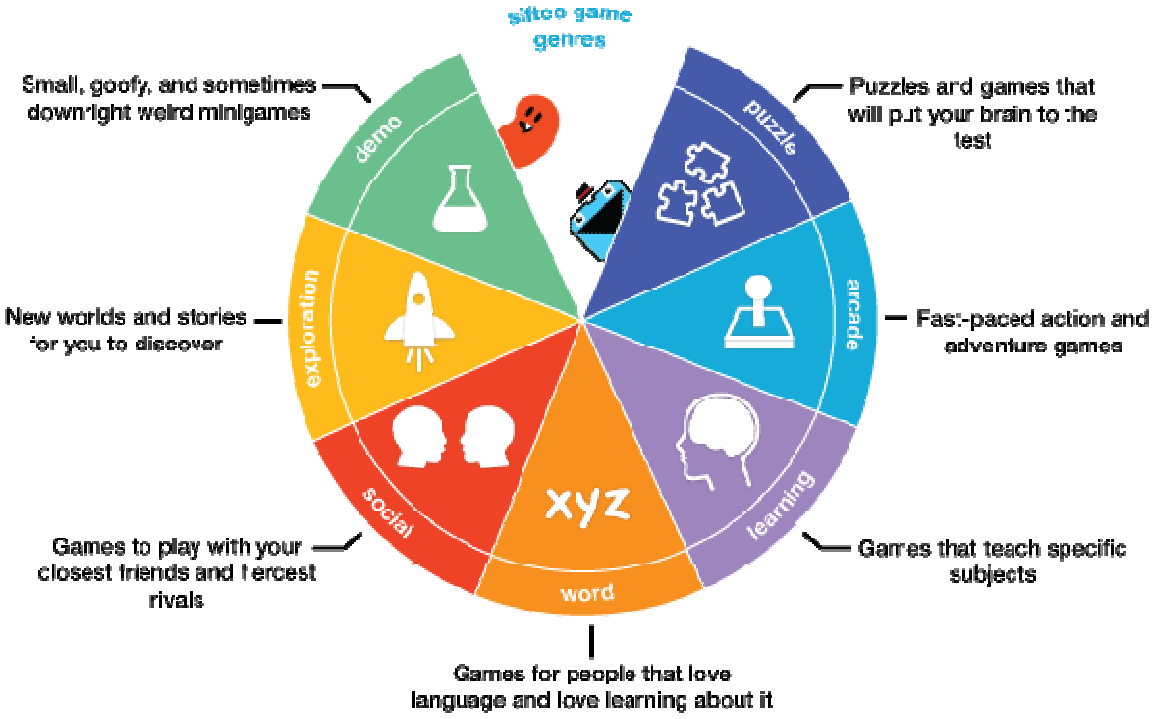

Figure 1. Sifteo Game Genres [3]

The different game genres Chef Sifty falls under includes:

- *Learning* – The focus of this game is to teach children about healthy nutrition and how to cook more healthy foods
- *Social* – There is also a social aspect to the game because children can collaborate with each other to make a recipe, but children can also cook the food they learn with their parents

II.ii. Sifteo Technical Specifications and Features

The Sifteo cubes have many features that make intelligent play interesting and there is a lot of technology in such a small cube. Most games in the Sifteo store leverage only one or a few of the features mentioned below.

The following describes the general features of the Sifteo cubes:

- Cubes can interact and react to its adjacent neighbors
- 3-axis accelerometer for tilting, shaking, and flipping
- Clickable LCD screen
- Can play music on the connected computer
- Can play sound on the connected computer based on different cube events
- Cubes support basic drawing and animation
- Can show images, animations, or other media on the connected computer based on different cube events

The following describes the technical specifications for each Sifteo cube:

- 32-bit ARM CPU
- 128 x 128 color TFT LCD (thin film transistor liquid crystal display)
- 3-axis accelerometer
- 8MB Flash
- Lithium Polymer rechargeable battery (with charging station dock)
- 2.4 GHz wireless radio (with USB dongle)
- Sifteo's proprietary near field communication (NFC) technology

II.iii. Minimum System Requirements

The following describes the computer system requirements you need to be able to play with the Sifteo cubes:

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 (500MB recommended)
- Internet connection (for software download and setup)

II.iv. Related Systems

II.iv.i. Scrabble Flash

This electronic version of Scrabble uses five interactive SmartLink tiles that you put together to form as many words as possible for a score. This is similar to Sifteo's Creativity Kit and Sifteo games such as Mount Brainiac, and Word Play.

II.iv.ii YAHTZEE Flash

This electronic version of YAHTZEE uses five smart dice that use Wonder-Link technology to allow rolling the dice and lining up the numbers you want to keep by lining those cubes up next to each other.

III. Requirements Specifications

III.i. Functional Requirements

Chef Sifty is made to be an interactive learning game for children to learn about the importance of nutrition and living a healthy lifestyle. This game shall consist of three different recipes that the user can make. Each recipe shall provide multiple steps in order to complete a recipe and the nutritional facts for each recipe shall also be displayed.

III.i.i. User Interface Overview

Figure 2. Chef Sifty Title Cubes

When the user starts the game as shown in Figure 2 above, he or she shall see the title screen where they must line up three cubes correctly in order to continue. Then the selection menu shall be displayed with the three different recipes for the user to make. They must connect the selection cube to the recipe cube in order to start that recipe. Also, there shall be music to accompany gameplay.

Each recipe is different in that there are different steps to make the recipe. Each recipe shall display and tell you through sound what each step is. All images shall be created using Sifteo image sprites, which were created using Adobe Photoshop.

The most important aspect of the game is the nutritional facts of the food being prepared, which shall be displayed when the recipe is completed along with why eating the food is healthy for you. Sound shall also be used to reiterate the healthiness of the food being prepared.

III.i.ii System Requirements

The following is a list of the various functionalities of the game:

- The user shall be able to select from three different recipes to make as shown in Figure 3 below
- The user shall be presented with specific directions to complete each recipe, both visually and through sound
- The user shall use the different features of the Sifteo cubes to make each recipe
    - Connecting cubes to add ingredients
    - Flipping a cube to flip the food
    - Shaking the cube to spread ingredient around in a pan
- The recipes shall be simple because the game is targeted toward children

Figure 3. Chef Sifty Menu Cubes

III.ii Non-Functional Requirements

The following explains the non-functional requirements of the system. As shown in Figure 4 and Figure 5 below, a fair amount is needed to run games on the Sifteo cubes.

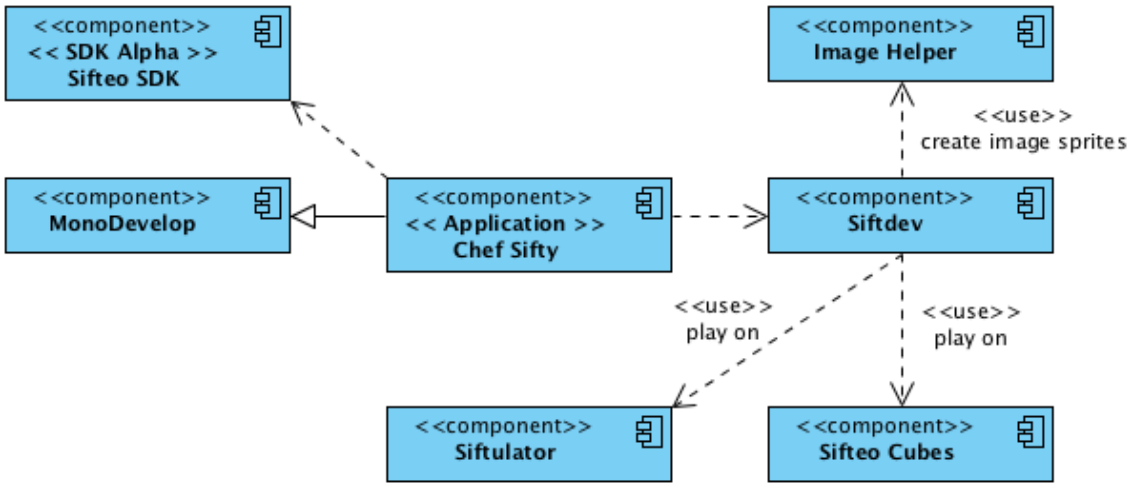

Figure 4. System Relationships and Dependencies

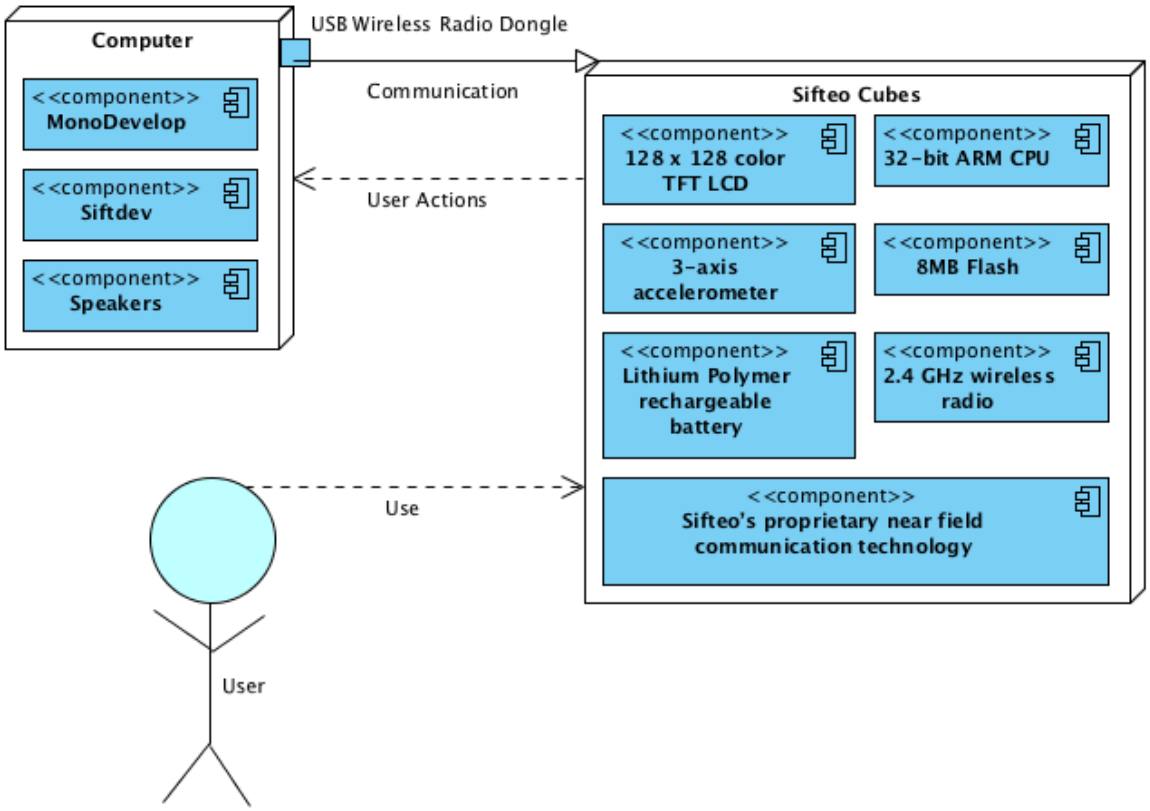

Figure 5. General Relationship between the User and Hardware

Figures 4 and 5 describe the organization of Sifteo's components and subsystems and their relationships and dependencies between one another. Depicted in these figures above, the Sifteo cubes rely on a computer running Siftdev with the loaded application as well as the Sifteo cubes themselves or the Siftulator. Also shown is the Sifteo SDK, which is in Alpha at the moment. This makes the Sifteo future promising because there is a lot of potential with this technology.

These details explain some of the non-functional requirements shown in the figures above:

- Development is done using C# using the MonoDevelop IDE
- In order to use the cubes, you must load the application using the Siftdev client application and make sure the cubes are connected using the actual Sifteo cubes or the cubes through the Siftulator, which is the Sifteo simulator
    - All images or image sprites created for Sifteo must be converted using the Image Helper within Siftdev
    - The application must be reloaded when any images or audio files are added to the project
- The Sifteo simulator, Siftualtor client application, can be used for testing
    - The number of cubes connected must be specified and the application's images directory path must also be specified
    - The application's images directory path must be redefined when any images are added to the project

III.ii.i Health and Nutrition Learning Statndardss

The learning standards described below are compiled from the Health Education Content Standards for California Public Schools [6].

III.iii. Limitations of Sifteo Cubes

Sifteo cubes still are not perfect and the SDK is still in Alpha. There is still much room for improvement and innovation, which makes this new technology interesting. There are many different human-computer interaction applications that can be created and the possibilities are vast. The following shows some limitations of the Sifteo cubes and what can be improved upon:

- You can only use a maximum of six cubes at one time
- Since each set of cubes use the same 2.4 GHz wireless radio dongle to connect the cubes to a computer, using different sets may interfere with each other when they are in the same vicinity

- You must be connected to a computer to be able to use the Sifteo cubes
- The cubes only sense each other when their two sides are touching
- The cubes do not sense stacking
- The accelerometer is not accurate and cannot be used to control another device, image, animation, or other media
- Tilting is also not very accurate and is not as responsive as the simulator
- The LCD is not a retina display and colors are augmented when viewed at an angle
- The cubes do not have that much memory, so you cannot store that many large games
- Games with many images or sprites cause the game to download really slowly
- The maximum range for wireless connection is 20 feet
- There is no USB charger for the cubes and the power adapter is 120V only, so international users need to use a step down voltage converter
- You cannot use Sifteo cubes with an iPad or any other mobile device
- A pack of three cubes cost $150 and a pack of six cubes cost $250, so Sifteo cubes are fairly expensive

IV. Design and Implementation

The overall design of this game consists of an easy to use and friendly interface. The user interface is a light interface with a playful font and colorful images to appeal to the children paying the game. Along with the simple design of the interface, the gameplay is also very simple so that children can have an easier playing experience, rather than frustrating them. This section further describes the overall flow of the game as well as the design for each recipe. The general application flow is shown in Figure 6 below.

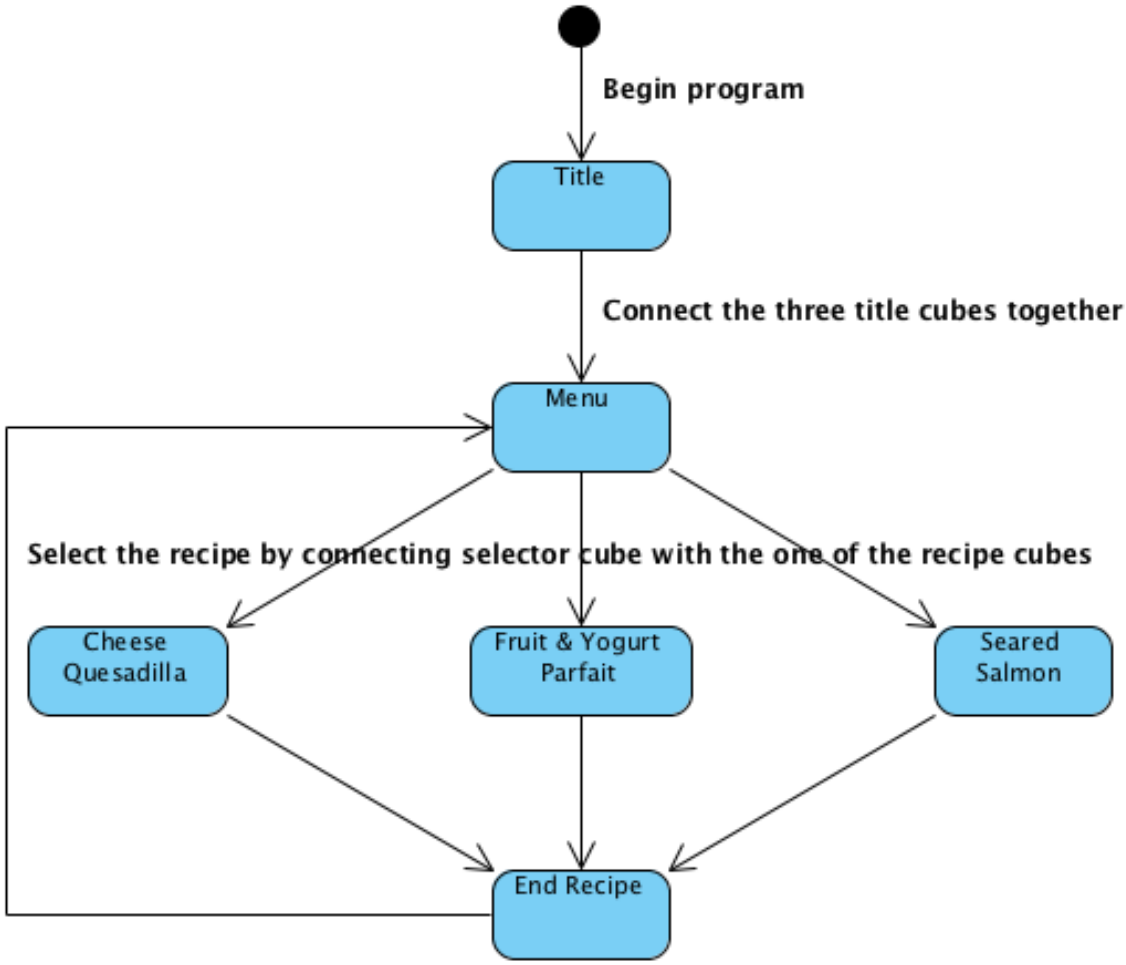

Figure 6. Overall Program Flow State Machine

IV.i Recipe Design

Each recipe was designed to be a step-by-step process to build or create the selected food. Since making food usually requires the adding or combining of ingredients, Chef Sifty is the perfect game to leverage the Sifteo cube neighboring to add ingredients to show the resulting combination on the resulting cube. Each step is displayed and told to the user so that they know what to do next. All three recipes follow a different procedure and the procedural flow is displayed in Figures 7, 8, and 9 below.

IV.i Cheese Quesadilla

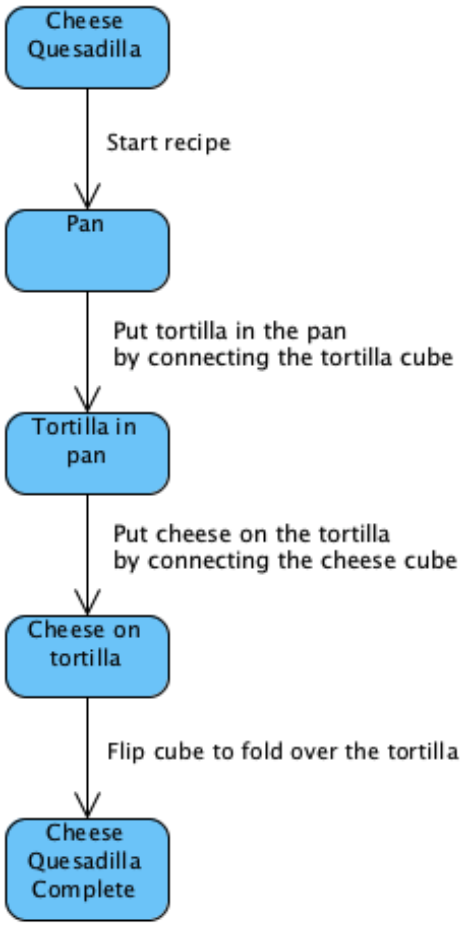

Figure 7. Cheese Quesadilla State Diagram

This recipe shows children a fast and easy meal or snack that can be healthy if you don't eat too much of it. The cheese is a good source of calcium and protein and children love foods they can eat with their hands, rendering the cheese quesadilla a simple and fun food for children to learn how to make.

Steps from Figure 7:

1. Put a tortilla in the pan
2. Put some cheese on the tortilla
3. Fold the tortilla over by flipping the cube
4. You have a nice cheesy quesadilla
5. Show the nutritional facts

IV.ii Fruit and Yogurt Parfait

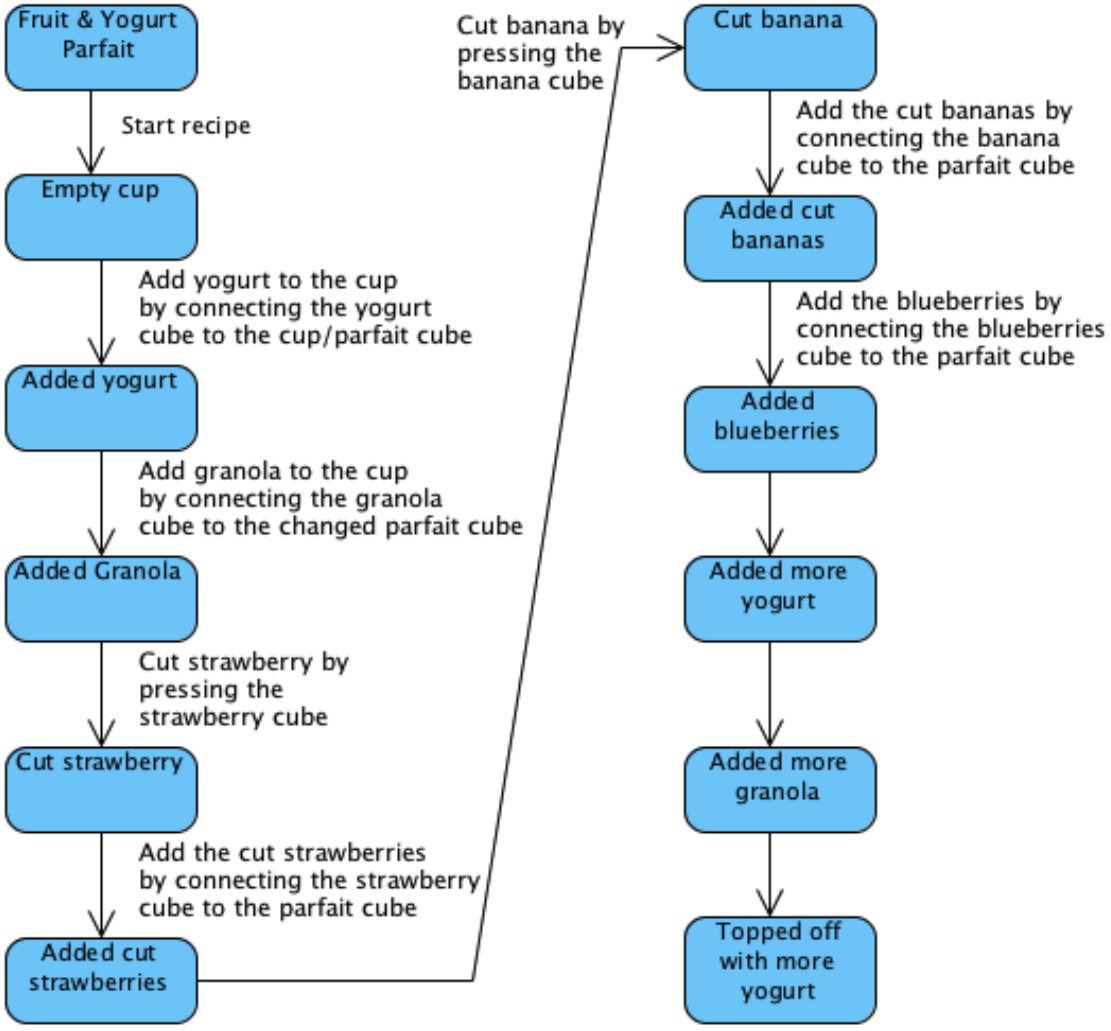

Figure 8. Fruit and Yogurt Parfait State Diagram

The fruit and yogurt parfait with granola is one of the healthiest and beneficial foods you can have as a snack or meal. This parfait provides calcium, protein, fiber, antioxidants, and many other benefits. The fruit also makes the parfait colorful and the layers make it fun to make.

Steps from Figure 8:

1. Add some yogurt to the cup
2. Add granola
3. Cut some strawberries by pressing the cube
4. Add the strawberries
5. Cut the banana
6. Add the bananas
7. Add blueberries
8. Add more yogurt
9. Add more granola
10. Top it off with more yogurt and that's it
11. Show the nutritional facts, health benefits, and the importance of breakfast

IV.iii Seared Salmon

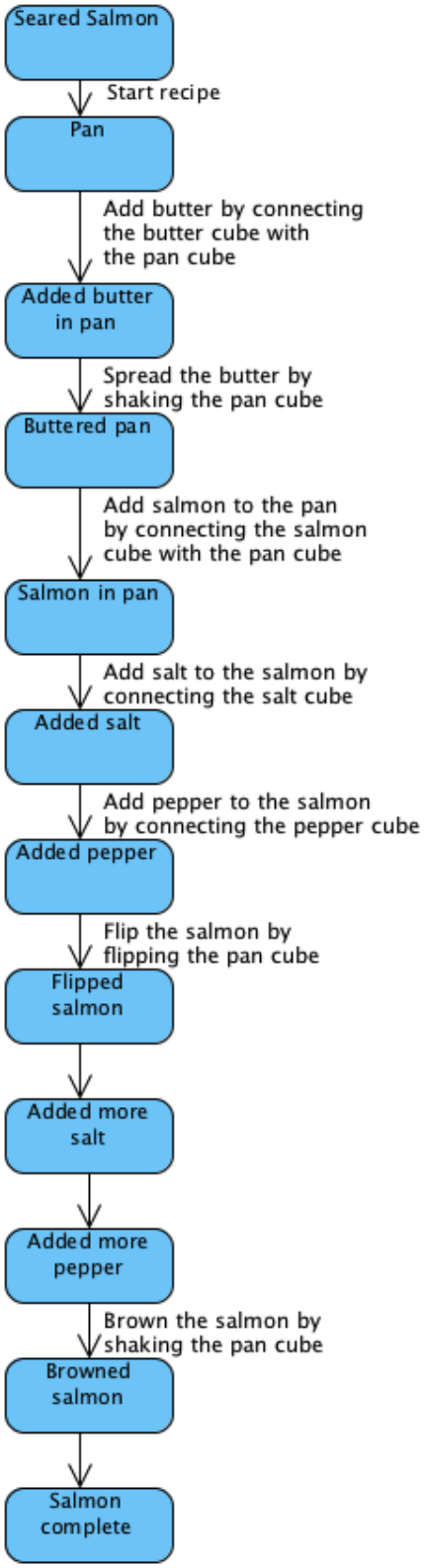

Figure 9. Seared Salmon State Diagram

Salmon is really good for you and is also a healthy alternative to any meat such as steak and it is delicious too. Because salmon is packed with disease fighting acids, vitamins, and minerals, it makes a perfect meal for children to learn about.

Steps from Figure 9:

1. Add a little butter to the pan
2. Spread the butter around the pan by shaking the cube
3. Add the salmon to the pan
4. Add some salt
5. Add some pepper
6. Flip the salmon over by flipping the cube
7. Add more salt
8. Add more pepper
9. Shake the salmon in the pan
10. Show the nutritional facts and health benefits

IV.ii Software Implementation

Learning C# wasn't that difficult, but getting used to the Sifteo SDK and how you should implement a game was fairly challenging. Since Sifteo technology is so new, there isn't a lot of tutorials or sample code to go off of. Also, the SDK documentation wasn't all that helpful. The most helpful resources that had been used were online guides by Sean Voisen, who is a Sr. Experience Developer at Adobe XD. One guide was **Up and running with the Sifteo SDK** [4] and the other guide was **Sifteo development with MVC and Ninject** [5], which helped with software architecture and the design pattern to be used. Although we didn't use Ninject because we couldn't get it working, Model-View-Controller, or MVC, proved to be an important design pattern in which we followed. MVC is really useful because it divides the application into three parts that define the way objects communicate with each other:

1. The **Model** represents the data specific to an application and should not directly access the View.
2. The **View** is what the user sees on the screen and only knows how to draw itself and can respond to user actions.
3. The **Controller** is the middleman where view objects can learn about changes in the model and update the view and model objects can learn about events or user actions from the view and update the model.

The Sifteo SDK includes `Sifteo.Util.StateMachine`, which allows for manageable program flow control using different states. Each state contains a model, a view, and a controller and knows how to transition between the different

states. Each controller also controls when to transition between states. An example from Sean Voisen is shown below [5]:

```
override public void Setup
{
  StateMachine sm = new StateMachine();
  sm.State("Title", titleController);
  sm.State("Menu", menuController);
  sm.State("Game", gameController);

  sm.Transition("Title", "TitleToMenu", "Menu");
  sm.Transition("Menu", "MenuToGame", "Game");
  sm.Transition("Game", "GameToMenu", "Menu");
}
```

Lastly, the Views use a Cube Wrapper that is specific to each state or each controller. Each cube in the Cube Set is initialized as a Cube Wrapper. You also add event listeners to each wrapped cube such as a neighbor add event or a button event. In this way, you can update the view or the wrapped cube based on the current specific state that you are in the game. Sample code for a Cube Wrapper from Sean Voisen is shown below [5]:

```
public class CubeWrapper
{
  public CubeWrapper(Cube cube)
  {
    Cube = cube;
  }

  public Cube Cube
  {
    get{ return cube; }

    set
    {
      cube = value;
      cube.userData = this;
    }
  }

  public void Paint()
  {
    // Painting code goes here
  }

  private Cube cube;
}
```

V. Testing

V.i. Acceptance Testing

Most of the testing was acceptance testing by going through the gameplay and making sure the expected outcomes actually occurred. There was some difference between how the simulator and how the actual Sifteo cubes would respond. The acceptance testing made sure that each state would move to the next state successfully when the correct user action was made.

V.i.i. Adding an Ingredient

For example, to add butter to the pan, you connect the butter cube to the pan cube as shown in Figure 10 below.

Figure 10. Adding butter to the pan

This should result in a change in the resulting pan cube as shown in Figure 11 below.

Figure 11. Pan cube changed to a pan with butter

V.i.ii. Mixing or Spreading an Ingredient

To spread the butter around in the pan, you shake the pan with butter cube as in Figure 11 above. The resulting cube should be a pan with butter spread around as shown in Figure 12.

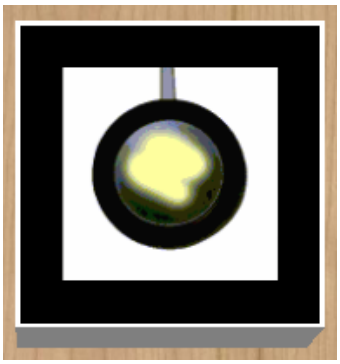

Figure 12. Pan with butter changed to a pan with spread butter

V.i.iii. Flipping

To flip the salmon, you flip the cube as shown in Figure 13 below.

Figure 13. Flipping the salmon

The resulting cube should change to reflect the flipped salmon as seen in Figure 14.

Figure 14. Pan with the flipped salmon

V.ii. Child Development Feedback – CD 413

We had visited the CD 413 class both to get ideas for what applications children would be interested in playing and feedback on our games and how they could be changed or improved.

V.ii.i. Brainstorming

Having actual child development majors help us brainstorm game ideas was really beneficial and helped us decide what we were going to develop. They gave us good ideas on how we should approach our application and how we should make a game that is targeted for children. At first, it was hard for us to come up with an idea because we were not in the mindset for creating a game that children would actually want to play.

When we visited the class, we gave an introductory presentation about Sifteo cubes and what the technology can do. We then had them play with the Sifteo cubes to get a feel for how they work and to start thinking about the different learning game possibilities. The class separated into groups and we conducted a brainstorming session to come up with a few game ideas per group and shared those ideas with everyone at the end. This brainstorming session was very helpful and it was beneficial to have a perspective of Child Development majors. With their help, I was ultimately able to finalize my learning game to be a cooking game to help children learn healthy nutrition.

V.ii.ii. Feedback

Getting feedback was also very useful because the child development students know how children would play and react to our games. We visited the class again after implementing most of our projects. The class was very excited and enthusiastic to see what we had developed. They gave us good feedback and valuable advice, which

was really helpful in order to get the perspective of how children would play our games.

For my game specifically, they told me to get rid of the tilting on the connected cube to add ingredients. This was because the tilting was a bit unresponsive and inaccurate. Also, at first, the first recipe I created was making cereal. The students advised me that most children cereal is really sweet and not that healthy for them. They told me that I should change it to a parfait recipe, which I implemented as a result and turned out to be successful. Lastly, they advised us to use sound to help the children learn through sound or audio in addition to visually and playing with the cubes physically.

VI. Conclusions and Recommendations

This project was interesting and a great learning experience. It was great to learn about this new technology and the idea behind Sifteo cubes. The company has only been around for 3 years and there is still a lot more they can do to innovate and create a great human-computer interaction for users of all ages. The concept of intelligent play and learning by playing is valuable because it allows for another medium where children can learn important subjects and still have fun. I also learned a lot going through the development process. It was interesting to work with the child development majors and we got a lot of ideas and insight into how to develop a learning game for children.

VI.i. Future Work

There is a great deal of growth for this project and also some room for improvement. This project utilizes only some of the Sifteo features, but could involve many more. Also, there could be some improvements to the project allowing for a better game experience. Some future work includes:

- Having more recipes
- Unlocking more recipes as you successfully create other recipes
- Using better animation
- Having recipes employ more features of the Sifteo cubes
    - Tilt ingredients into the result cube
    - Match the ingredient pictures with their names
    - Measure ingredients by tilting the ingredient into a measuring cup
- Use the computer screen to interact with the cubes as well

References

[1] Ogden CL, Carroll MD, Kit BK, Flegal KM. Prevalence of obesity in the United States, 2009–2010. NCHS data brief, no 82. Hyattsville, MD: National Center for Health Statistics. 2012. http://www.cdc.gov/nchs/data/databriefs/db82.htm

[2] Perry, B., "The Importance of Pleasure in Play," *Scholastic Inc.* http://teacher.scholastic.com/professional/bruceperry/pleasure.htm

[3] Sifteo Inc., "Intelligent Play?," 2012. https://www.sifteo.com/play

[4] Voisen, S., "Up and running with the Sifteo SDK," October 4th 2012. http://sean.voisen.org/blog/2011/10/up-and-running-with-sifteo-sdk/

[5] Voisen, S., "Up and running with the Sifteo SDK," October 25th 2012. http://sean.voisen.org/blog/2011/10/sifteo-development-mvc-ninject/

[6] California State Board, "Health Education Content Standards for California Public Schools," March 2008. http://www.cde.ca.gov/be/st/ss/documents/healthstandmar08.pdf

Appendices

A. Senior Project Analysis

Summary of Functional Requirements
Chef Sifty is an interactive learning game for children to learn about the importance of nutrition and living a healthy lifestyle. This game consists of three different recipes that the user can make. Each recipe provides multiple steps in order to complete a recipe and the nutritional facts for each recipe is also displayed. Once the user selects a recipe, they are provided procedural instructions in order to create the food such as adding ingredients by combining cubes using the Sifteo near field communication or flipping the cube to flip the food.

Primary Constraints
There were some issues that made this project challenging and difficult. First of all, I had to learn C# and get familiar with the Sifteo SDK and the development tools needed. Getting up and running took longer than expected because the technology is very new and there aren't many guides or tutorials on how to develop Sifteo games. The SDK is also Alpha, so it isn't as detailed or complete. Also, the development tools were somewhat hard to get used to because Sifteo uses three different developments tools in order to run a game: MonoDevelop for development, Siftdev for running the game, and Siftulator for simulating the game. The hardest task at the beginning was creating the image sprites, which involved a process where you had to convert the image sprite using the Image Helper in Siftdev. Another aspect that made this project difficult was creating my own image sprites in Adobe Photoshop. I spent a good amount of time finding pictures and editing them to create the image sprites for the game.

There were also a few hardware related constraints that altered the project. The accelerometer is not accurate and tilting is not as responsive as the simulator, so to add the ingredients, you just connect cubes instead of tilting onto the resulting cube. In addition, the cubes do not have that much memory, so you cannot store that many images or audio files. This made the game install onto the cubes very slowly.

Economic
Original estimated cost of component parts: $250
Actual final cost of component parts: $250

Bill of Materials

Component	Quantity	Price Per Unit	Cost
Sifteo Cubes	6	$45	$250

Additional equipment costs: none

Original estimated development time: about 60 days
Actual development time: about 70 days

Environmental
The Sifteo cubes do not contain that many components and do not require that much power. You can play Sifteo games for three hours or more without the need to recharge.

Manufacturability
N/A

Sustainability
The only challenge with Sifteo cubes is that you need to charge the cubes and the laptop in order to play Sifteo games. To help improve Sifteo cubes, they could build the cubes with components that have lower power consumption. They could also look into using Bluetooth Low Energy for wireless communication so that you could play Sifteo games with mobile devices and tablets.

Ethical
There aren't any unethical issues involved with Sifteo cubes and you cannot misuse the project.

Health and Safety
Sifteo cubes are not intended for users under the age of 3 because the USB wireless link poses as a choking hazard. Possible seizures may occur depending on the child and the game they are playing. Also, excessive play may cause injury to hands, wrists, arms or eyes. Sifteo cubes can emit radio waves that can affect the operation of nearby electronics, including cardiac pacemakers. The Sifteo cubes contain rechargeable lithium ion battery packs. Leakage of ingredients contained within the battery pack, or the combustion products of the ingredients, can cause personal injury as well as damage to the Sifteo cubes.

Social and Political
All Sifteo games in the Sifteo store do not pay developers, so developers are kind of developing freeware. This might pose a problem because most third party developers won't want to develop applications for free.

Development
I learned C# and the Sifteo SDK as well the development tools needed to create a Sifteo game. These tools include MonoDevelop, Siftdev, and the Siftulator. Also, I had to learn how to use the State Machine interface from the Sifteo SDK in order to develop and implement the control-flow for my game in a modular fashion. I also got better at using Adobe Photoshop to create and edit images.

B. Source Code

B.a. ChefSifty

```
using Sifteo;
using Sifteo.Util;
using System;
using System.Collections;
using System.Collections.Generic;

namespace ChefSifty
{
	public class ChefSifty : BaseApp
	{
		public List<CubeWrapper> mWrappers = new List<CubeWrapper> (0); //
list of wrapped cubes
		private bool mNeedCheck;
		private Sound mMusic;
		private int lastIndex;
		public StateMachine sm;

		// State MachineControllers
		private TitleController titleController;
		private MenuController menuController;
		private CerealController cerealController;
		private QuesadillaController quesadillaController;
		private ParfaitController parfaitController;
		private SalmonController salmonController;

		// States
		public static readonly string TitleState = "Title";
		public static readonly string MenuState = "Menu";
		public static readonly string CerealState = "Cereal";
		public static readonly string QuesadillaState = "Quesadilla";
		public static readonly string ParfaitState = "Parfait";
		public static readonly string SalmonState = "Salmon";

		// Transition IDs
		public static readonly string tTitleToMenu = "TitleToMenu";
		public static readonly string tMenuToCereal = "MenuToCereal";
		public static readonly string tMenuToQuesadilla =
"MenuToQuesadilla";
		public static readonly string tMenuToParfait = "MenuToParfait";
		public static readonly string tMenuToSalmon = "MenuToSalmon";
		public static readonly string tCerealToMenu = "CerealToMenu";
		public static readonly string tQuesadillaToMenu =
"QuesadillaToMenu";
		public static readonly string tParfaitToMenu = "ParfaitToMenu";
		public static readonly string tSalmonToMenu = "SalmonToMenu";

		override public int FrameRate {
			get { return 20; }
		}

		// called during intitialization, before the game has started to
run
		override public void Setup ()
		{
			Log.Debug ("Setup()");
			mNeedCheck = true;
```

```
                    // play game music at a lower volume: 0.15f
                    mMusic = Sounds.CreateSound ("music2");
                    mMusic.Play (0.15f, 1);

                    // Init the State Machine
                    sm = new StateMachine ();

                    // Init the Controllers
                    titleController = new TitleController ();
                    menuController = new MenuController (this, CubeSet);
                    cerealController = new CerealController (this, CubeSet);
                    quesadillaController = new QuesadillaController (this,
CubeSet);
                    parfaitController = new ParfaitController (this, CubeSet);
                    salmonController = new SalmonController (this, CubeSet);

                    // assign the respective state to each controller
                    sm.State (TitleState, titleController);
                    sm.State (MenuState, menuController);
                    sm.State (CerealState, cerealController);
                    sm.State (QuesadillaState, quesadillaController);
                    sm.State (ParfaitState, parfaitController);
                    sm.State (SalmonState, salmonController);

                    // set transitions
                    sm.Transition (TitleState, tTitleToMenu, MenuState);
                    sm.Transition (MenuState, tMenuToCereal, CerealState);
                    sm.Transition (MenuState, tMenuToQuesadilla,
QuesadillaState);
                    sm.Transition (MenuState, tMenuToParfait, ParfaitState);
                    sm.Transition (MenuState, tMenuToSalmon, SalmonState);
                    sm.Transition (CerealState, tCerealToMenu, MenuState);
                    sm.Transition (QuesadillaState, tQuesadillaToMenu,
MenuState);
                    sm.Transition (ParfaitState, tParfaitToMenu, MenuState);
                    sm.Transition (SalmonState, tSalmonToMenu, MenuState);

                    // set state to the Title state with the tTitleToMenu
transition
                    sm.SetState (TitleState, tTitleToMenu);

                    // Loop through all the cubes and set them up.
                    lastIndex = 1;
                    int spriteYPos = 0; // the iamge sprite position

                    // initialize each cube in the CubeSet by
                    // wrapping it and painting the image on the cube
                    // image/screen dimensions are 128x128
                    foreach (Cube cube in CubeSet) {
                           CubeWrapper wrapper = new CubeWrapper (this, cube,
lastIndex);
                           lastIndex++;
                           mWrappers.Add (wrapper);

                           // Draw sprites
                           cube.FillScreen (Color.White);

                           if (lastIndex > 4) {
                                  cube.Image (
                                  "chefSiftySprites",
                                  0,
                                  0,
```

```
                                0,
                                spriteYPos,
                                Cube.SCREEN_WIDTH,
                                Cube.SCREEN_HEIGHT,
                                0,
                                0
                                );
                                spriteYPos += Cube.SCREEN_HEIGHT;
                        }
                        cube.Paint ();
                }

                // add event listeners to the CubeSet
                // event callbacks functions are specified below
                CubeSet.NewCubeEvent += OnNewCube;
                CubeSet.LostCubeEvent += OnLostCube;
                CubeSet.NeighborAddEvent += OnNeighborAdd;
                CubeSet.NeighborRemoveEvent += OnNeighborRemove;
        }

        // development mode only
        // start ChefSifty as an executable and run it, waiting for
Siftrunner to connect
//      static void Main (string[] args)
//      {
//              new ChefSifty ().Run ();
//      }

        // ### New Cube ###
        // When a new cube connects while the game is running, we need to
create a
        // wrapper for it so that it is included in gameplay.
        //
        // If a new cube is added while the game is paused, this event
will be
        // handled after the player unpauses, but before the unpause event
is
        // handled.
        private void OnNewCube (Cube c)
        {
                Log.Debug ("New Cube {0}", c.UniqueId);
                CubeWrapper wrapper = (CubeWrapper)c.userData;
                if (wrapper == null) {
                        wrapper = new CubeWrapper (this, c, lastIndex);
                        lastIndex++;
                        mWrappers.Add (wrapper);
                        Log.Debug ("{0}", mWrappers);
                }
                mNeedCheck = true;
        }

        // ### Lost Cube ###
        // When a cube falls offline while the game is running, we need to
delete
        // its wrapper.
        //
        // If Siftrunner forced the to pause due to a cube going offline,
this
        // event will be handled before the pause event is handled.
        private void OnLostCube (Cube c)
        {
                Log.Debug ("Lost Cube {0}", c.UniqueId);
                CubeWrapper wrapper = (CubeWrapper)c.userData;
```

```
                        if (wrapper != null) {
                                c.userData = null;
                                mWrappers.Remove (wrapper);
                        }
                        mNeedCheck = true;
                }

                // On Neighbor Add Event
                private void OnNeighborAdd (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
                {
                        mNeedCheck = true;

                        // choose recipe
                        if (sm.CurrentState == menuController) {
                                // the MenuCube wrapper instance initialized with the
two cubes connected
                                MenuCube wrapper1 = (MenuCube)cube1.userData;
                                MenuCube wrapper2 = (MenuCube)cube2.userData;
                                Log.Debug ("connected " + wrapper1.mIndex + " and " +
wrapper2.mIndex);

                                // chech which recipe is selected
                                if (wrapper1.mIndex == 1) {
                                        if (wrapper2.mIndex == 4) {
                                                Log.Debug ("Quesadilla");
                                                Sound s = Sounds.CreateSound ("tasty");
                                                s.Play (1);
                                                stateTransition (tMenuToQuesadilla);
                                                sm.Tick (1);
                                        }  else if (wrapper2.mIndex == 5) {
                                                Log.Debug ("Parfait");
                                                Sound s = Sounds.CreateSound
("parfait");
                                                s.Play (1);
                                                stateTransition (tMenuToParfait);
                                                sm.Tick (1);
                                        }  else if (wrapper2.mIndex == 6) {
                                                Log.Debug ("Salmon");
                                                Sound s = Sounds.CreateSound
("salmon");
                                                s.Play (1);
                                                stateTransition (tMenuToSalmon);
                                                sm.Tick (1);
                                        }
                                }  else if (wrapper2.mIndex == 1) {
                                        if (wrapper1.mIndex == 4) {
                                                Log.Debug ("Quesadilla");
                                                Sound s = Sounds.CreateSound ("tasty");
                                                s.Play (1);
                                                stateTransition (tMenuToQuesadilla);
                                                sm.Tick (1);
                                        }  else if (wrapper1.mIndex == 5) {
                                                Log.Debug ("Parfait");
                                                Sound s = Sounds.CreateSound
("parfait");
                                                s.Play (1);
                                                stateTransition (tMenuToParfait);
                                                sm.Tick (1);
                                        }  else if (wrapper1.mIndex == 6) {
                                                Log.Debug ("Salmon");
                                                Sound s = Sounds.CreateSound
("salmon");
```

```
                                        s.Play (1);
                                        stateTransition (tMenuToSalmon);
                                        sm.Tick (1);
                                }
                        }
                }
        }

        // On Neighbor Remove Event
        private void OnNeighborRemove (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
        {
                mNeedCheck = true;
        }

        // called every frame and also calls the other controller Ticks
        public override void Tick ()
        {
                // Here we see if anyone raised the flag for a neighbor
check
                if (mNeedCheck) {
                        mNeedCheck = false;
                        bool changed = CheckNeighbors ();
                        if (changed)
                                sm.CurrentState.OnPaint (true);
                }
        }

        // check connected neighbors
        private bool CheckNeighbors ()
        {
                bool connected = false;
                int totalCubes = CubeSet.Count;
                // connect only 3 cubes
                int numCubesToConnect = (sm.CurrentState ==
titleController) ? 3 : totalCubes;

                // check if title cubes are lined up correctly
                if (sm.CurrentState == titleController) {
                        // ### CubeHelper.FindColumn ###
                        // FindColumn returns the first column found in the
given cube set. It
                        // can be used to check whether your cubes are all
lined up.
                        //
                        // A column is a series of cubes neighbored **bottom
to top**.  Cubes can
                        // only form a column if they are all oriented the
same way.
                        Cube[] column = CubeHelper.FindColumn (CubeSet);
                        // If we have a full column, check to see if it is
sorted by index.
                        if (column.Length == numCubesToConnect) {
                                connected = true;
                                int lastId = 7;
                                foreach (Cube cube in column) {
                                        CubeWrapper wrapper =
(CubeWrapper)cube.userData;
                                        Log.Debug ("id={0}, last={1}",
wrapper.mIndex, lastId);
                                        if (wrapper.mIndex > lastId)
                                                connected = false;
                                        lastId = wrapper.mIndex;
```

```
                                }
                        }
                        Log.Debug ("connected: {0}", connected);
                }

                // Here we go through each wrapper and update its state
depending on the
                // results of our search.
                foreach (CubeWrapper wrapper in mWrappers) {
                        wrapper.CheckNeighbors (connected);
                }

                // NOW CONNECTED -> go to MENU
                if (connected && sm.CurrentState == titleController) {
                        // play sound and transition
                        Sound s = Sounds.CreateSound ("cookintime");
                        s.Play (1);
                        stateTransition (tTitleToMenu);
                        sm.Tick (1);
                }

                return connected;
        }

        // Play Sound
        public void PlaySound (string sound, int loop)
        {
                Sound s = Sounds.CreateSound (sound);
                s.Play (1, loop);
        }

        // Manages state transisitons
        public void stateTransition (string transition)
        {
                Log.Debug ("Start Transition = " + transition + " from " +
sm.CurrentState);

                // based on current state, transition appropriately

                // go to Menu from Title state
                if (sm.CurrentState == titleController) {
                        sm.QueueTransition (tTitleToMenu);
                }

                // go to chosen Recipe state from the Menu
                if (sm.CurrentState == menuController && transition ==
tMenuToQuesadilla) {
                        sm.QueueTransition (tMenuToQuesadilla);
                }
                if (sm.CurrentState == menuController && transition ==
tMenuToParfait) {
                        sm.QueueTransition (tMenuToParfait);
                }
                if (sm.CurrentState == menuController && transition ==
tMenuToSalmon) {
                        sm.QueueTransition (tMenuToSalmon);
                }

                // return to Menu from the Recipe
                if (sm.CurrentState == quesadillaController && transition
== tQuesadillaToMenu) {
                        sm.QueueTransition (tQuesadillaToMenu);
                        Sound s = Sounds.CreateSound ("delicious");
```

```
                        s.Play (1);

                        // force repaint
                        menuController.OnPaint (true);
                    }
                    if (sm.CurrentState == parfaitController && transition ==
tParfaitToMenu) {
                        sm.QueueTransition (tParfaitToMenu);
                        Sound s = Sounds.CreateSound ("nicejob");
                        s.Play (1);

                        // force repaint
                        menuController.OnPaint (true);
                    }
                    if (sm.CurrentState == salmonController && transition ==
tSalmonToMenu) {
                        sm.QueueTransition (tSalmonToMenu);
                        Sound s = Sounds.CreateSound ("good");
                        s.Play (1);

                        // force repaint
                        menuController.OnPaint (true);
                    }
                }
        }

        // ## CubeWrapper ##

        /// <summary>
        /// Cube wrapper.
        /// </summary>

        public class CubeWrapper
        {
                public ChefSifty mApp;
                public Cube mCube;
                public int mIndex;
                private int mRotation;
                public IStateController    mCubeStateController;
                public StateMachine mCubeStateMachine;

                // This flag tells the wrapper to redraw the current image on the
cube. (See Tick, below).
                public bool mCubeSelected = false; //Was selected by the user on
this tick
                public bool mIsSelected = false; //Is noted as the selected cube
                public bool mNeedDraw = true;

                public CubeWrapper (ChefSifty app, Cube cube, int seq)
                {
                        mApp = app;
                        mCube = cube;
                        mCube.userData = this;
                        mRotation = 0;
                        // add a tilt event for each cube
                        mCube.TiltEvent += OnTilt;
                        mIndex = seq;

                        // add a button press event for each cube
                        mCube.ButtonEvent += HandleCubeButtonEvent;

                        //mCubeStateController
                }
```

```
		//Handle Button Event
		void HandleCubeButtonEvent (Cube c, bool pressed)
		{
			//Check if the button was pressed
			if (pressed) {
				Log.Debug ("{0} Button pressed", c);
			}
		}

		// On Tilt Event
		private void OnTilt (Cube cube, int tiltX, int tiltY, int tiltZ)
		{
			int oldRotation = mRotation;
			// If the cube is tilted to a standing position, set the
sprite's
			// rotation so that its head is pointing towards that side.
			if (tiltZ == 1) {
				if (tiltX == 0) { // LEFT
					mRotation = 1;
				} else if (tiltX == 2) { // RIGHT
					mRotation = 2;
				} else if (tiltY == 0) { // DOWN
					mRotation = 3;
				} else if (tiltY == 2) { // UP
					mRotation = 4;
				}
			} else {
				mRotation = 0;
			}

			// If the rotation has changed, raise the flag to force a
repaint.
			if (mRotation != oldRotation) {
				//Log.Debug ("ROTATED " + mRotation);
				mNeedDraw = true;
			}
		}

		public void CheckNeighbors (bool rowFound)
		{
			if (mCube != null) {

				// ### CubeHelper.FindConnected ###
				// CubeHelper.FindConnected returns an array of all
cubes that are
				// neighbors of the given cube, or neighbors of those
neighbors, etc.
				// The result includes the given cube, so there
should always be at
				// least one element in the array.
				//
				// Here we check to see if the cube is connected to
any other cubes,
				// and if it is, we draw the orange background.
//				Cube[] connected = CubeHelper.FindConnected (mCube);
//				if (connected.Length > 1) {
//				}
//
//				if (rowFound) {
//					mSpriteIndex = 2;
//					mRectColor = new Color (182, 218, 85);
//				}
```

```
                        mNeedDraw = true;
                }
        }

        // called every frame by the main Tick()
        public void Tick ()
        {
                // If anyone has raised the mNeedDraw flag, redraw the
image on the cube.
                if (mNeedDraw) {
                        Log.Debug ("mNeedDraw {0}", this.mCube.UniqueId);
                        mNeedDraw = false;
                        //Paint ();
                }
        }

        public void Paint ()
        {
                if (mCube != null) {

                        mCube.Paint ();
                }
        }
    }
}
```

B.b. MenuController and MenuCube

```
using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
    public class MenuController : IStateController
    {
        String classname = "MenuController";
        public List<MenuCube> mWrappers = new List<MenuCube> (0);
        private int lastIndex;
        public CubeSet cubes;

        //Init
        public MenuController (ChefSifty app, CubeSet cubeSet)
        {
            Log.Debug (classname + " Init");
            cubes = cubeSet;
        }

        //Setup
        public void OnSetup (string trainsitionId)
        {
            Log.Debug (classname + " OnSetup");

            // Loop through all the cubes and set them up.
```

```
                        lastIndex = 1;
                        foreach (Cube cube in cubes) {
                                MenuCube wrapper = new MenuCube (this, cube,
lastIndex);
                                lastIndex += 1;
                                mWrappers.Add (wrapper);
                        }
                }

                public void OnTick (float dt)
                {
                        Log.Debug (classname + " OnTick");
                }

                // paint each cube with the menu image sprites
                public void OnPaint (bool canvasDirty)
                {
                        Log.Debug ("menu OnPaint({0})", canvasDirty);

                        //Check the cube set
                        if (cubes != null) {
                                //Make sure the canvas needs to be redrawn
                                if (canvasDirty) {
                                        int i = 0, mSpriteIndex = 0;

                                        //Cycle through all the cubes
                                        foreach (Cube cube in cubes) {
                                                //Paint the cube
                                                if (cube != null) {
                                                        cube.FillScreen (Color.White);
                                                        if (i == 0 || i > 2) {
                                                                // Draw sprites
                                                                cube.Image (
                                                                        "menuSprites",
                                                                        0,
                                                                        0,
                                                                        0,
                                                                        mSpriteIndex * 128,
                                                                        Cube.SCREEN_WIDTH,
                                                                        Cube.SCREEN_HEIGHT,
                                                                        0,
                                                                        0
                                                                );
                                                                mSpriteIndex++;
                                                        }
                                                        cube.Paint ();
                                                        i++;
                                                }  else {
                                                        //Handle this exception
                                                }
                                        }

                                }  else {
                                        //Skip paint
                                        //return
                                }
                        }  else {
                                //Handle this exception
                        }
                }

                // On Dispose called when transitioning and cleans up
                public void OnDispose ()
```

```
		{
			Log.Debug (classname + " OnDispose");
		}
	}
}


using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
	// this class is pretty much useless
	// just a cube wrapper class for the
	// menu, which is used in the main class
	public class MenuCube
	{
		public MenuController mApp;
		public Cube mCube;
		public int mIndex;
		private int mRotation;
		public IStateController    mCubeStateController;
		public StateMachine mCubeStateMachine;


		// This flag tells the wrapper to redraw the current image on the
cube. (See Tick, below).
		public bool mCubeSelected = false;
		public bool mNeedDraw = true;

		public MenuCube (MenuController app, Cube cube, int seq)
		{
			Log.Debug ("init menu cube");
			mApp = app;
			mCube = cube;
			mCube.userData = this;
			mRotation = 0;
			mCube.TiltEvent += OnTilt;
			mIndex = seq;

			mCube.NeighborAddEvent += OnNeighborAdd;
			mCube.NeighborRemoveEvent += OnNeighborRemove;
			mCube.ButtonEvent += HandleCubeButtonEvent;

			//mCubeStateController
		}

		private void OnNeighborAdd (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
		{
			Log.Debug ("menu cube add");
			mNeedDraw = true;
		}

		private void OnNeighborRemove (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
		{
			mNeedDraw = true;
		}

		private void HandleCubeButtonEvent (Cube c, bool pressed)
```

```
		{
			//Check if the button was pressed
			if (pressed) {
				Log.Debug ("pressed {0}", c.UniqueId);

				//If this cube was already selected, turn it off
				if (mCubeSelected) {
					mCubeSelected = false;
				}  else {
					mCubeSelected = true;

//					mCubeStateMachine.QueueTransition
(ChefSifty.tTitleToMenu);
//					mCubeStateMachine.Tick (1);
					//Check if a neighboor cube is selected
					//If they are, tell them to not be.
				}
				//Refresh the screen by setting this flag
				mNeedDraw = true;
			}
		}

		private void OnTilt (Cube cube, int tiltX, int tiltY, int tiltZ)
		{
			Log.Debug ("menu cube tilt");

			int oldRotation = mRotation;
			// If the cube is tilted to a standing position, set the
sprite's
			// rotation so that its head is pointing towards that side.
			if (tiltZ == 1) {
				if (tiltY == 2) {
					mRotation = 0;
				}  else if (tiltY == 0) {
					mRotation = 2;
				}  else if (tiltX == 0) {
					mRotation = 1;
				}  else if (tiltX == 2) {
					mRotation = 3;
				}
			}  else {
				mRotation = 0;
			}

			// If the rotation has changed, raise the flag to force a
repaint.

			if (mRotation != oldRotation) {
				mNeedDraw = true;
			}
		}

		// This method changes the background color depending on the game
state.
		public void CheckNeighbors (bool rowFound)
		{
			Log.Debug ("menu cube checking neighbors");
		}

		// This method is called every frame by the Tick in SorterApp.
(see above.)
		public void Tick ()
		{
```

```
                        Log.Debug("menucube tick");

                        // If anyone has raised the mNeedDraw flag, redraw the
image on the cube.
                        if (mNeedDraw) {
                                Log.Debug ("mNeedDraw {0}", this.mCube.UniqueId);
                                mNeedDraw = false;
                                Paint ();
                        }
                }

                public void Paint ()
                {
                        Log.Debug ("Painting {0}", mCube.UniqueId);
                }
        }
}
```

B.c. QuesadillaController and QuesadillaCube

```
using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
        // every controller has to implement the IStateController interface
        public class QuesadillaController : IStateController
        {
                String classname = "QuesadillaController";
                public List<QuesadillaCube> mWrappers = new List<QuesadillaCube>
(0);
                private int lastIndex;
                public CubeSet cubes;

                ChefSifty mApp;

                //Init
                public QuesadillaController (ChefSifty app, CubeSet cubeSet)
                {
                        Log.Debug (classname + " Init");
                        mApp = app;
                        cubes = cubeSet;
                }

                //Setup
                public void OnSetup (string trainsitionId)
                {
                        Log.Debug (classname + " OnSetup");

                        // Loop through all the cubes and set them up.
                        lastIndex = 1;
                        foreach (Cube cube in cubes) {
                                QuesadillaCube wrapper = new QuesadillaCube (this,
cube, lastIndex);
                                mWrappers.Add (wrapper);

                                cube.FillScreen (Color.White);

                                // call individual cube's Paint()
```

```
                        cube.Paint ();
                        lastIndex++;
                    }
                }

                public void OnTick (float dt)
                {
                    Log.Debug (classname + " OnTick");

                    // call Tick() on each ParfairCube
                    foreach (QuesadillaCube wrapper in mWrappers) {
                        wrapper.Tick ();
                    }
                }

                // call ChefSifty's stateTransition to change state
                public void stateTransition ()
                {
                    Log.Debug ("transitioning " + ChefSifty.tQuesadillaToMenu +
" from " + classname);
                    mApp.stateTransition (ChefSifty.tQuesadillaToMenu);
                    mApp.sm.Tick (1);
                }

                // call ChefSifty's PlaySound to play a sound
                public void playSound (string sound, int loop)
                {
                    mApp.PlaySound (sound, loop);
                }

                public void OnPaint (bool canvasDirty)
                {
                    Log.Debug ("Quesadilla OnPaint({0})", canvasDirty);
                    // paints cube from each ParfaitCube wrapper on each Tick
                }

                // On Dispose do cleanup
                public void OnDispose ()
                {
                    Log.Debug (classname + " OnDispose");

                    // dispose each cube wrapper in the CubeSet
                    foreach (Cube cube in cubes) {
                        QuesadillaCube wrapper =
(QuesadillaCube)cube.userData;
                        Log.Debug ("Disposing Quesadilla Cube {0}",
wrapper.mIndex);
                        wrapper.dispose ();
                        if (wrapper != null) {
                            cube.userData = null;
                            mWrappers.Remove (wrapper);
                        }
                    }
                }
            }
}


using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;
```

```
namespace ChefSifty
{
	public class QuesadillaCube
	{
		public QuesadillaController mApp;
		public Cube mCube;
		public int mIndex;
		private int mSpriteIndex;
		public IStateController    mCubeStateController;
		public StateMachine mCubeStateMachine;


		// This flag tells the wrapper to redraw the current image on the
cube. (See Tick, below).
		public bool mCubeSelected = false;
		public bool mNeedDraw = true;

		public QuesadillaCube (QuesadillaController app, Cube cube, int
seq)
		{
			Log.Debug ("init Quesadilla cube");
			mApp = app;
			mCube = cube;
			mCube.userData = this;
			mSpriteIndex = 0; // sprite index
			mIndex = seq;

			// add event listeners to the cube
			// event callbacks functions are specified below
			mCube.NeighborAddEvent += OnNeighborAdd;
			mCube.ButtonEvent += HandleCubeButtonEvent;
			mCube.ShakeStartedEvent += OnShakeStarted;
			mCube.FlipEvent += OnFlip;
		}

		// On Neighbor Add Event
		private void OnNeighborAdd (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
		{
			Log.Debug ("Quesadilla cube add");
			//mNeedDraw = true;

			// check added ingredient on neighbor add
			// if recipe not at the last step
			if (mSpriteIndex < 9)
				CheckNeighbors ();
		}

		// Handle Cube Button Press Event for returning to the menu
		private void HandleCubeButtonEvent (Cube c, bool pressed)
		{
			QuesadillaCube wrapper = (QuesadillaCube)c.userData;

			//Check if the button was pressed
			if (pressed) {
				Log.Debug ("pressed {0}", wrapper.mIndex);

				// change state back to MENU
				if (mSpriteIndex == 12)
					mApp.stateTransition ();
			}
		}
```

```
		// On Shake Event that just transitions back to the menu
		private void OnShakeStarted (Cube cube)
		{
			// change state back to MENU
			if (mSpriteIndex == 12)
				mApp.stateTransition ();
		}

		// On Flip Event for flipping the quesadilla
		private void OnFlip (Cube cube, bool newOrientationIsUp)
		{
			if (newOrientationIsUp) {
				Log.Debug ("{0} Flip face up", mIndex);

				// redraw
				if (mSpriteIndex == 9 && mIndex == 5) {
					mApp.playSound ("done", 0);
					mNeedDraw = true;
					Tick ();
				}
			}  else {
				Log.Debug ("{0} Flip face down", mIndex);
				//mNeedDraw = true;
			}
		}

		// check neighbors to see if the right ingredient was added
		public void CheckNeighbors ()
		{
			Log.Debug ("Quesadilla cube checking neighbors");

			int okCubes = 0;

			if (mCube != null) {
				// ### CubeHelper.FindConnected ###
				// CubeHelper.FindConnected returns an array of all
cubes that are
				// neighbors of the given cube, or neighbors of those
neighbors, etc.
				// The result includes the given cube, so there
should always be at
				// least one element in the array.
				//
				// Here we check to see if the cube is connected to
any other cubes,
				// and if it is, we draw the orange background.
				Cube[] connected = CubeHelper.FindConnected (mCube);
				foreach (Cube cube in connected) {
					QuesadillaCube wrapper =
(QuesadillaCube)cube.userData;
					Log.Debug ("cube index = " + wrapper.mIndex);
					// return if an invalid cube is connected
					if (okCubes < 2 && wrapper.mIndex % 2 == 0)
						return;
					if (wrapper.mIndex == 1 || wrapper.mIndex ==
5)
						okCubes++;
				}

				// the right ingredient was added, so redraw
				if (okCubes == 2) {
					mNeedDraw = true;
					Tick ();
```

```
                        }
                }
        }

        public void Tick ()
        {
                Log.Debug ("Quesadilla cube tick");

                // If anyone has raised the mNeedDraw flag, redraw the
image on the cube.
                if (mNeedDraw) {
                        Log.Debug ("mNeedDraw {0}", this.mCube.UniqueId);
                        mNeedDraw = false;
                        // paint the right image sprites
                        Paint ();
                }
        }

        // Paint each cube in the set
        public void Paint ()
        {
                Log.Debug ("Painting sprite {0} Quesadilla cube {1}",
mSpriteIndex, mIndex);
                int i = 0;

                if (mCube != null && mSpriteIndex < 12) {
                        foreach (Cube cube in mApp.cubes) {
                                cube.FillScreen (Color.White);

                                // Draw sprites on even cubes
                                if (i % 2 == 0) {
                                        cube.Image (
                                                "quesadillaSprites",
                                                0,
                                                0,
                                                0,
                                                mSpriteIndex * 128,
                                                Cube.SCREEN_WIDTH,
                                                Cube.SCREEN_HEIGHT,
                                                0,
                                                0
                                        );
                                        mSpriteIndex++;
                                }
                                cube.Paint ();
                                i++;
                        }

                        if (mSpriteIndex == 9)
                                mApp.playSound ("flip", 0);
                }
        }

        // removes the events from the cubes
        public void dispose ()
        {
                mCube.NeighborAddEvent -= OnNeighborAdd;
                mCube.ButtonEvent -= HandleCubeButtonEvent;
                mCube.ShakeStartedEvent -= OnShakeStarted;
                mCube.FlipEvent -= OnFlip;
        }
    }
}
```

B.d. ParfaitController and ParfaitCube

```
using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
	// every controller has to implement the IStateController interface
	public class ParfaitController : IStateController
	{
		String classname = "ParfaitController";
		public List<ParfaitCube> mWrappers = new List<ParfaitCube> (0);
		private int lastIndex;
		public CubeSet cubes;

		ChefSifty mApp;

		//Init
		public ParfaitController (ChefSifty app, CubeSet cubeSet)
		{
			Log.Debug (classname + " Init");
			mApp = app;
			cubes = cubeSet;
		}

		//Setup
		public void OnSetup (string trainsitionId)
		{
			Log.Debug (classname + " OnSetup");

			// Loop through all the cubes and set them up.
			lastIndex = 1;
			foreach (Cube cube in cubes) {
				ParfaitCube wrapper = new ParfaitCube (this, cube,
lastIndex);
				mWrappers.Add (wrapper);

				cube.FillScreen (Color.White);

				// call individual cube's Paint()
				cube.Paint ();
				lastIndex++;
			}

			mApp.PlaySound ("shake", 0);
		}

		public void OnTick (float dt)
		{
			Log.Debug (classname + " OnTick");

			// call Tick() on each ParfairCube
			foreach (ParfaitCube wrapper in mWrappers) {
				wrapper.Tick ();
			}
		}

		// call ChefSifty's stateTransition to change state
```

```
		public void stateTransition ()
		{
			Log.Debug ("transitioning " + ChefSifty.tParfaitToMenu + "
from " + classname);
			mApp.stateTransition (ChefSifty.tParfaitToMenu);
			mApp.sm.Tick (1);
		}

		// call ChefSifty's PlaySound to play a sound
		public void playSound (string sound, int loop)
		{
			mApp.PlaySound (sound, loop);
		}

		public void OnPaint (bool canvasDirty)
		{
			Log.Debug ("Parfait OnPaint({0})", canvasDirty);
			// paints cube from each ParfaitCube wrapper on each Tick
		}

		// On Dispose do cleanup
		public void OnDispose ()
		{
			Log.Debug (classname + " OnDispose");

			// dispose each cube wrapper in the CubeSet
			foreach (Cube cube in cubes) {
				ParfaitCube wrapper = (ParfaitCube)cube.userData;
				Log.Debug ("Disposing Parfait Cube {0}",
wrapper.mIndex);
				wrapper.dispose ();
				if (wrapper != null) {
					cube.userData = null;
					mWrappers.Remove (wrapper);
				}
			}
		}
	}
}


using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
	public class ParfaitCube
	{
		public ParfaitController mApp;
		public Cube mCube;
		public int mIndex;
		private static int mParfaitIndex;
		private static int mStrawberryIndex;
		private static int mBananaIndex;
		private int mSpriteIndex;
		private bool isMaking = true;
		public IStateController    mCubeStateController;
		public StateMachine mCubeStateMachine;


		// This flag tells the wrapper to redraw the current image on the
```

```
cube. (See Tick, below).
		public bool mCubeSelected = false;
		public bool mNeedDraw = true;

		public ParfaitCube (ParfaitController app, Cube cube, int seq)
		{
			Log.Debug ("init Parfait cube");
			mApp = app;
			mCube = cube;
			mCube.userData = this;

			// sprite indexes
			mParfaitIndex = 0;
			mStrawberryIndex = 0;
			mBananaIndex = 0;
			mSpriteIndex = 0;

			// cube index
			mIndex = seq;

			// add event listeners to the ParfaitCube
			// event callbacks functions are specified below
			mCube.NeighborAddEvent += OnNeighborAdd;
			mCube.ButtonEvent += HandleCubeButtonEvent;
			mCube.ShakeStartedEvent += OnShakeStarted;
		}

		// On Neighbor Add Event
		private void OnNeighborAdd (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
		{
			Log.Debug ("Parfait cube add");

			// check added ingredient on neighbor add
			// if recipe not at the last step
			if (mParfaitIndex < 9)
				CheckNeighbors ();
		}

		// Handle Cube Button Press Event for chop action
		private void HandleCubeButtonEvent (Cube c, bool pressed)
		{
			// initialize wrapper with the pressed cube to get the
index
			ParfaitCube wrapper = (ParfaitCube)c.userData;

			//Check if the button was pressed
			if (pressed) {
				Log.Debug ("pressed parfait cube {0}",
wrapper.mIndex);

				// cut strawberry
				if (wrapper.mIndex == 5 && mStrawberryIndex < 3) {
					mStrawberryIndex++;
					mApp.playSound ("chop", 0); // play chop sound
					// Draw sprites
					c.Image (
						"strawberrySprites",
						0,
						0,
						0,
						mStrawberryIndex * 128,
						Cube.SCREEN_WIDTH,
```

```
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
                c.Paint ();
        }

        // cut banana
        if (wrapper.mIndex == 6 && mBananaIndex < 3) {
                mBananaIndex++;
                mApp.playSound ("chop", 0); // play chop sound
                // Draw sprites
                c.Image (
                        "bananaSprites",
                        0,
                        0,
                        0,
                        mBananaIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
                c.Paint ();
        }

        Log.Debug ("index = " + mParfaitIndex);

        // end of recipe
        if (mParfaitIndex == 8) {
                mParfaitIndex++;
                Paint ();
        }
        // change state back to MENU
        else if (mParfaitIndex == 9)
                mApp.stateTransition ();
    }
}

// On Shake -> toggle displaying the recipe instructions
private void OnShakeStarted (Cube cube)
{
    Log.Debug ("SHAKING!");

    if (isMaking) {
        isMaking = false;

        // set index based on how far you are in the recipe
        int i = (mParfaitIndex < 4) ? 0 : 6;

        // paint each cube to show the recipe steps
        foreach (Cube c in mApp.cubes) {
            c.FillScreen (Color.White);
            c.Image (
                "recipeSprites",
                0,
                0,
                0,
                i * 128,
                Cube.SCREEN_WIDTH,
                Cube.SCREEN_HEIGHT,
                0,
                0
```

```
                                );
                                c.Paint ();
                                i++;
                        }
                } else {
                        // paint the parfait recipe cubes
                        isMaking = true;
                        Paint();
                }
        }

        // check neighbors to see if the right ingredient was added
        public void CheckNeighbors ()
        {
                Log.Debug ("Parfait cube checking neighbors");

                int okCubes = 0;

                // get the row or column of the two connected cubes
                Cube[] cubesToCheck;
                Cube[] row = CubeHelper.FindRow (mApp.cubes);
                Cube[] column = CubeHelper.FindColumn (mApp.cubes);
                if (row.Length == 2) {
                        Log.Debug ("ROW FOUND");
                        cubesToCheck = row;
                } else if (column.Length == 2) {
                        Log.Debug ("COLUMN FOUND");
                        cubesToCheck = column;
                } else {
                        return;
                }

                // based on the parfait index or where you are in the
                // recipe, check to see if the right ingredeient was added
                switch (mParfaitIndex) {
                case 0:
                        foreach (Cube cube in cubesToCheck) {
                                ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                                if (wrapper.mIndex == 1 || wrapper.mIndex ==
2)
                                        okCubes++;
                        }
                        break;

                case 1:
                        foreach (Cube cube in cubesToCheck) {
                                ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                                if (wrapper.mIndex == 1 || wrapper.mIndex ==
3)
                                        okCubes++;
                        }
                        break;

                case 2:
                        if (mStrawberryIndex == 3) {
                                foreach (Cube cube in cubesToCheck) {
                                        ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                                        if (wrapper.mIndex == 1 ||
wrapper.mIndex == 5)
                                                okCubes++;
```

```
                        }
                    }  else {
                        // tell user they need to cut
                    }
                    break;

                case 3:
                    if (mBananaIndex == 3) {
                        foreach (Cube cube in cubesToCheck) {
                            ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                            if (wrapper.mIndex == 1 ||
wrapper.mIndex == 6)
                                okCubes++;
                        }
                    }  else {
                        // tell user they need to cut
                    }
                    break;

                case 4:
                    foreach (Cube cube in cubesToCheck) {
                        ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
4)
                            okCubes++;
                    }
                    break;

                case 5:
                    foreach (Cube cube in cubesToCheck) {
                        ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
2)
                            okCubes++;
                    }
                    break;

                case 6:
                    foreach (Cube cube in cubesToCheck) {
                        ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
3)
                            okCubes++;
                    }
                    break;

                case 7:
                    foreach (Cube cube in cubesToCheck) {
                        ParfaitCube wrapper =
(ParfaitCube)cube.userData;
                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
2)
                            okCubes++;
                    }
                    break;

                default:
                    break;
                }
```

```
                        // the right ingredient was added, so increment the recipe
step
                        if (okCubes == 2) {
                                mParfaitIndex++;
                                mNeedDraw = true;
                                // redraw
                                Tick ();
                        }
                }

                public void Tick ()
                {
                        Log.Debug ("Parfait cube tick");

                        // If anyone has raised the mNeedDraw flag, redraw the
image on the cube.
                        if (mNeedDraw) {
                                Log.Debug ("mNeedDraw {0}", this.mCube.UniqueId);
                                mNeedDraw = false;
                                // paint the right image sprites
                                Paint ();
                        }
                }

                // Paint each cube in the set
                public void Paint ()
                {
                        Log.Debug ("Painting sprite {0} Parfait cube {1}",
mParfaitIndex, mIndex);
                        int i = 0;

                        if (mCube != null && mParfaitIndex < 10) {
                                foreach (Cube cube in mApp.cubes) {
                                        cube.FillScreen (Color.White);

                                        // paint each cube in the set based in their
current indexes
                                        if (i == 0) {
                                                // Draw sprites
                                                cube.Image (
                                                        "parfaitSprites",
                                                        0,
                                                        0,
                                                        0,
                                                        mParfaitIndex * 128,
                                                        Cube.SCREEN_WIDTH,
                                                        Cube.SCREEN_HEIGHT,
                                                        0,
                                                        0
                                                );
                                                //mParfaitIndex++;
                                        }  else if (i == 1) {
                                                // change sprite if you're on the last
step
                                                mSpriteIndex = (mParfaitIndex == 9) ? 3
: 0;
                                                // Draw sprites
                                                cube.Image (
                                                        "niceSprites",
                                                        0,
                                                        0,
                                                        0,
```

```
                        mSpriteIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
        }  else if (i == 2) {
                // change sprite if you're on the last
step
                mSpriteIndex = (mParfaitIndex == 9) ? 4
: 1;
                // Draw sprites
                cube.Image (
                        "niceSprites",
                        0,
                        0,
                        0,
                        mSpriteIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
        }  else if (i == 3) {
                // change sprite if you're on the last
step
                mSpriteIndex = (mParfaitIndex == 9) ? 5
: 2;
                // Draw sprites
                cube.Image (
                        "niceSprites",
                        0,
                        0,
                        0,
                        mSpriteIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
        }  else if (i == 4) {
                // Draw sprites
                // change sprite if you're on the last
step
                if (mParfaitIndex == 9) {
                        cube.Image (
                                "niceSprites",
                                0,
                                0,
                                0,
                                768,
                                Cube.SCREEN_WIDTH,
                                Cube.SCREEN_HEIGHT,
                                0,
                                0
                        );
                }  else {
                        cube.Image (
                                "strawberrySprites",
                                0,
                                0,
                                0,
                                mStrawberryIndex * 128,
```

```
                                                Cube.SCREEN_WIDTH,
                                                Cube.SCREEN_HEIGHT,
                                                0,
                                                0
                                        );
                                }
                        }
                        // paint if not on the last step
                        else if (i == 5 && mParfaitIndex != 9) {
                                // Draw sprites
                                cube.Image (
                                        "bananaSprites",
                                        0,
                                        0,
                                        0,
                                        mBananaIndex * 128,
                                        Cube.SCREEN_WIDTH,
                                        Cube.SCREEN_HEIGHT,
                                        0,
                                        0
                                );
                        }
                        cube.Paint ();
                        i++;
                }
                Log.Debug ("@ sprite " + mParfaitIndex + " after
Paint()");
                }
        }

        // removes the events from the cubes
        public void dispose ()
        {
                mCube.NeighborAddEvent -= OnNeighborAdd;
                mCube.ButtonEvent -= HandleCubeButtonEvent;
                mCube.ShakeStartedEvent -= OnShakeStarted;
        }
    }
}
```

B.e. SalmonController and SalmonCube

```
using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
    // every controller has to implement the IStateController interface
    public class SalmonController : IStateController
    {
        String classname = "SalmonController";
        public List<SalmonCube> mWrappers = new List<SalmonCube> (0);
        private int lastIndex;
        public CubeSet cubes;

        ChefSifty mApp;

        //Init
        public SalmonController (ChefSifty app, CubeSet cubeSet)
```

```
		{
			Log.Debug (classname + " Init");
			mApp = app;
			cubes = cubeSet;
		}

		//Setup
		public void OnSetup (string trainsitionId)
		{
			Log.Debug (classname + " OnSetup");

			// Loop through all the cubes and set them up.
			lastIndex = 1;
			foreach (Cube cube in cubes) {
				SalmonCube wrapper = new SalmonCube (this, cube,
lastIndex);
				mWrappers.Add (wrapper);

				cube.FillScreen (Color.White);

				// call individual cube's Paint()
				cube.Paint ();
				lastIndex++;
			}
		}

		public void OnTick (float dt)
		{
			Log.Debug (classname + " OnTick");

			// call Tick() on each ParfairCube
			foreach (SalmonCube wrapper in mWrappers) {
				wrapper.Tick ();
			}
		}

		// call ChefSifty's stateTransition to change state
		public void stateTransition ()
		{
			Log.Debug ("transitioning " + ChefSifty.tSalmonToMenu + "
from " + classname);
			mApp.stateTransition (ChefSifty.tSalmonToMenu);
			mApp.sm.Tick (1);
		}

		// call ChefSifty's PlaySound to play a sound
		public void playSound (string sound, int loop)
		{
			mApp.PlaySound (sound, loop);
		}

		public void OnPaint (bool canvasDirty)
		{
			Log.Debug ("Salmon OnPaint({0})", canvasDirty);
			// paints cube from each SalmonCube wrapper on each Tick
		}

		// On Dispose do cleanup
		public void OnDispose ()
		{
			Log.Debug (classname + " OnDispose");

			// dispose each cube wrapper in the CubeSet
```

```
                    foreach (Cube cube in cubes) {
                           SalmonCube wrapper = (SalmonCube)cube.userData;
                           Log.Debug ("Disposing Salmon Cube {0}",
wrapper.mIndex);
                           wrapper.dispose ();
                           if (wrapper != null) {
                                  cube.userData = null;
                                  mWrappers.Remove (wrapper);
                           }
                    }
             }
       }
}


using System;
using Sifteo.Util;
using Sifteo;
using System.Collections.Generic;

namespace ChefSifty
{
       public class SalmonCube
       {
              public SalmonController mApp;
              public Cube mCube;
              public int mIndex;
              private static int mSalmonIndex;
              public IStateController    mCubeStateController;
              public StateMachine mCubeStateMachine;


              // This flag tells the wrapper to redraw the current image on the
cube. (See Tick, below).
              public bool mCubeSelected = false;
              public bool mNeedDraw = true;

              public SalmonCube (SalmonController app, Cube cube, int seq)
              {
                     Log.Debug ("init Salmon cube");
                     mApp = app;
                     mCube = cube;
                     mCube.userData = this;

                     // sprite indexes
                     mSalmonIndex = 0;

                     // cube index
                     mIndex = seq;

                     // add event listeners to the SalmonCube
                     // event callbacks functions are specified below
                     mCube.NeighborAddEvent += OnNeighborAdd;
                     mCube.ButtonEvent += HandleCubeButtonEvent;
                     mCube.ShakeStartedEvent += OnShakeStarted;
                     mCube.FlipEvent += OnFlip;
              }

              // On Neighbor Add Event
              private void OnNeighborAdd (Cube cube1, Cube.Side side1, Cube
cube2, Cube.Side side2)
              {
                     Log.Debug ("Salmon cube add");
```

```
                        // check added ingredient on neighbor add
                        // if recipe not at the last step
                        if (mSalmonIndex < 9)
                                CheckNeighbors ();
                }

                // Handle Cube Button Press Event for chop action
                private void HandleCubeButtonEvent (Cube c, bool pressed)
                {
                        // initialize wrapper with the pressed cube to get the
index
                        SalmonCube wrapper = (SalmonCube)c.userData;

                        //Check if the button was pressed
                        if (pressed) {
                                Log.Debug ("pressed Salmon cube {0}",
wrapper.mIndex);
                                Log.Debug ("index = " + mSalmonIndex);

                                // end of recipe
                                if (mSalmonIndex == 9) {
                                        mSalmonIndex++;
                                        Paint ();
                                }
                                // change state back to MENU
                                else if (mSalmonIndex == 10)
                                        mApp.stateTransition ();
                        }
                }

                // On Shake -> toggle displaying the recipe instructions
                private void OnShakeStarted (Cube cube)
                {
                        Log.Debug ("SHAKING!");

                        // redraw
                        if (mSalmonIndex == 1 || mSalmonIndex == 8) {
                                mSalmonIndex++;
                                mNeedDraw = true;
                                Tick ();
                        }
                }

                // On Flip Event for flipping the salmon
                private void OnFlip (Cube cube, bool newOrientationIsUp)
                {
                        if (newOrientationIsUp) {
                                Log.Debug ("{0} Flip face up", mIndex);

                                // redraw
                                if (mSalmonIndex == 5 && mIndex == 1) {
                                        mSalmonIndex++;
                                        mNeedDraw = true;
                                        Tick ();
                                }
                        }  else {
                                Log.Debug ("{0} Flip face down", mIndex);
                                //mNeedDraw = true;
                        }
                }

                // check neighbors to see if the right ingredient was added
```

```
	public void CheckNeighbors ()
	{
		Log.Debug ("Salmon cube checking neighbors");

		int okCubes = 0;

		// get the row or column of the two connected cubes
		Cube[] cubesToCheck;
		Cube[] row = CubeHelper.FindRow (mApp.cubes);
		Cube[] column = CubeHelper.FindColumn (mApp.cubes);
		if (row.Length == 2) {
			Log.Debug ("ROW FOUND");
			cubesToCheck = row;
		}  else if (column.Length == 2) {
			Log.Debug ("COLUMN FOUND");
			cubesToCheck = column;
		}  else {
			return;
		}

		// based on the Salmon index or where you are in the
		// recipe, check to see if the right ingredeient was added
		switch (mSalmonIndex) {
		case 0:
			foreach (Cube cube in cubesToCheck) {
				SalmonCube wrapper =
(SalmonCube)cube.userData;
				if (wrapper.mIndex == 1 || wrapper.mIndex ==
3)
					okCubes++;
			}
			break;

		case 2:
			foreach (Cube cube in cubesToCheck) {
				SalmonCube wrapper =
(SalmonCube)cube.userData;
				if (wrapper.mIndex == 1 || wrapper.mIndex ==
4)
					okCubes++;
			}
			break;

		case 3:
			foreach (Cube cube in cubesToCheck) {
				SalmonCube wrapper =
(SalmonCube)cube.userData;
				if (wrapper.mIndex == 1 || wrapper.mIndex ==
5)
					okCubes++;
			}
			break;

		case 4:
			foreach (Cube cube in cubesToCheck) {
				SalmonCube wrapper =
(SalmonCube)cube.userData;
				if (wrapper.mIndex == 1 || wrapper.mIndex ==
6)
					okCubes++;
			}
			break;
```

```
                        case 6:
                                foreach (Cube cube in cubesToCheck) {
                                        SalmonCube wrapper =
(SalmonCube)cube.userData;
                                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
5)
                                                okCubes++;
                                }
                                break;

                        case 7:
                                foreach (Cube cube in cubesToCheck) {
                                        SalmonCube wrapper =
(SalmonCube)cube.userData;
                                        if (wrapper.mIndex == 1 || wrapper.mIndex ==
6)
                                                okCubes++;
                                }
                                break;

                        default:
                                break;
                        }

                        // the right ingredient was added, so increment the recipe
step
                        if (okCubes == 2) {
                                mSalmonIndex++;
                                mNeedDraw = true;
                                // redraw
                                Tick ();
                        }
                }

                public void Tick ()
                {
                        Log.Debug ("Salmon cube tick");

                        // If anyone has raised the mNeedDraw flag, redraw the
image on the cube.
                        if (mNeedDraw) {
                                Log.Debug ("mNeedDraw {0}", this.mCube.UniqueId);
                                mNeedDraw = false;
                                // paint the right image sprites
                                Paint ();
                        }
                }

                // Paint each cube in the set
                public void Paint ()
                {
                        Log.Debug ("Painting sprite {0} Salmon cube {1}",
mSalmonIndex, mIndex);
                        int i = 0;

                        if (mCube != null) {
                                foreach (Cube cube in mApp.cubes) {
                                        cube.FillScreen (Color.White);

                                        // paint each cube in the set based in their
current indexes
                                        if (i == 0) {
                                                // Draw sprites
```

```
                cube.Image (
                        "salmonSprites",
                        0,
                        0,
                        0,
                        mSalmonIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
                //mSalmonIndex++;
        }  else if (i == 1) {
                // Draw sprites
                if (mSalmonIndex == 10) {
                        cube.Image (
                                "salmonSprites",
                                0,
                                0,
                                0,
                                1408,
                                Cube.SCREEN_WIDTH,
                                Cube.SCREEN_HEIGHT,
                                0,
                                0
                        );
                }  else {
                        cube.Image (
                                "sRecipeSprites",
                                0,
                                0,
                                0,
                                mSalmonIndex * 128,
                                Cube.SCREEN_WIDTH,
                                Cube.SCREEN_HEIGHT,
                                0,
                                0
                        );
                }
        }  else if (i == 2) {
                // change sprite if you're on the last
step

                int spriteIndex = (mSalmonIndex == 9) ?
4 : 0;

                // Draw sprites
                cube.Image (
                        "sIngredientsSprites",
                        0,
                        0,
                        0,
                        spriteIndex * 128,
                        Cube.SCREEN_WIDTH,
                        Cube.SCREEN_HEIGHT,
                        0,
                        0
                );
        }  else if (i == 3) {
                // Draw sprites
                cube.Image (
                        "sIngredientsSprites",
                        0,
                        0,
                        0,
```

```
                                        128,
                                        Cube.SCREEN_WIDTH,
                                        Cube.SCREEN_HEIGHT,
                                        0,
                                        0
                                );
                        }  else if (i == 4) {
                                cube.Image (
                                        "sIngredientsSprites",
                                        0,
                                        0,
                                        0,
                                        256,
                                        Cube.SCREEN_WIDTH,
                                        Cube.SCREEN_HEIGHT,
                                        0,
                                        0
                                );
                        }  else if (i == 5) {
                                cube.Image (
                                        "sIngredientsSprites",
                                        0,
                                        0,
                                        0,
                                        384,
                                        Cube.SCREEN_WIDTH,
                                        Cube.SCREEN_HEIGHT,
                                        0,
                                        0
                                );
                        }
                        cube.Paint ();
                        i++;
                }
                Log.Debug ("@ sprite " + mSalmonIndex + " after
Paint()");

                if (mSalmonIndex == 9 || mSalmonIndex == 10)
                        mApp.playSound ("done", 0);
        }
    }

    // removes the events from the cubes
    public void dispose ()
    {
        mCube.NeighborAddEvent -= OnNeighborAdd;
        mCube.ButtonEvent -= HandleCubeButtonEvent;
        mCube.ShakeStartedEvent -= OnShakeStarted;
    }
  }
}
```