Masters Project

LeapMotion for Kids

Author:
Christopher Desch

Supervisor:
Dr. Jerry Fails

Submitted in partial fulfilment of the requirements
for the degree of Masters of Computer Science

in the

Department of Computer Science
Montclair State University

May 2013

MONTCLAIR STATE UNIVERSITY

Abstract

Department of Computer Science

Masters of Computer Science

LeapMotion for Kids

by Christopher DESCH

The LeapMotion Technology presents a new way of interacting with a computer system alternative to the customary keyboard and mouse interface. The goal of the project was to discover whether or not an alternative, "hands off" interface has the possibility of being as successful and reliable as the keyboard and mouse interface. The purpose of this paper is to describe the development process of building an application for this new interface in order to highlight the capabilities of the new interface. The application was designed by KidsTeam through cooperative inquiry techniques for the LeapMotion.

Acknowledgements

I would like to express my sincerest gratitude to my supervisor Dr. Jerry Fails for his constructive comments and insights throughout the process of creating this master project. His enthusiasm for his craft is both inspiring and contagious. I would like to thank Montclair State University for the opportunity to study as part of their learning community. I would like to thank the faculty and staff of the University for... Furthermore, I would like to thank the members of KidsTeam for volunteering their time and energy to the work necessary to complete this project. I thank you for seeing a world of possibilities in the smallest of ideas. Finally, I would like to thank my family and friends for their unending love and support.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
HUD	**H**eads **U**p **D**isplay
SDK	**S**oftware **D**eveloper **K**it
RAD	**R**apid **A**pplication **D**evelopment
USB	**U**niversal **S**erial **B**us

Chapter 1

Overview

1.1 Introduction

For decades, the keyboard and mouse have been the standard interface mechanisms of input for computer systems. Their legacy is apparent in the applications that have been designed with the purpose of maximizing their efficiency and effectiveness. However, as technology advances, new mechanisms have been created in the hopes of making interactions with computer systems less rigid. While the keyboard and mouse interface are less physically demanding on the body in order to carry out commands, they do not allow for an expressive range of motion. The keyboard and mouse are only expressive in two dimensions as they are limited to up, down or right, left movements of the mouse across a computer screen. The keyboard is even less expressive than the mouse as the user must use key strokes to carry out commands, thus making the mouse slightly more fluid in motion than the keyboard but not by much. With the advent of tablets and smart devices, the envelope had been pushed in terms of what could be considered "effective" means of input for a computer system. These devices allow for a user to carry out a command with a simple stroke of their finger directly on the screen of their computer system, thus making the command more expressive than the traditional keyboard and mouse interface. The devices with touchscreen capabilities have opened the door for opportunities to create a fresh interface. This new mechanism would have the ability to

FIGURE 1.1: Interacting with the LeapMotion. [1]

replace the use of a keyboard and mouse while offering an even greater, more expressive range of motion to the user in a more "hands-off" capacity, the term "hands-off" referring to the fact that the keyboard and mouse interface as well as the touchscreen interface both require the user to literally be touching a device of some sort in order to carry out a command. However, certain interface technologies were created with the intention of never having to place one's hands on a device.

With these ideas in mind, the project set out with the simple goal of developing an application for children by children using an interface with these specifications. The interface chosen was LeapMotion.

1.2 LeapMotion

The LeapMotion is a small device slightly larger then a USB drive which sits in front of a monitor and captures motion in 3 cubic feet of space using a pair of cameras. The small cameras triangulate the positions of hands, fingers and tools in their relative space between the LeapMotion and the monitor relaying accurate position and velocity data in real time. The data can then be used to control application by driving the user interface of the system. [3]

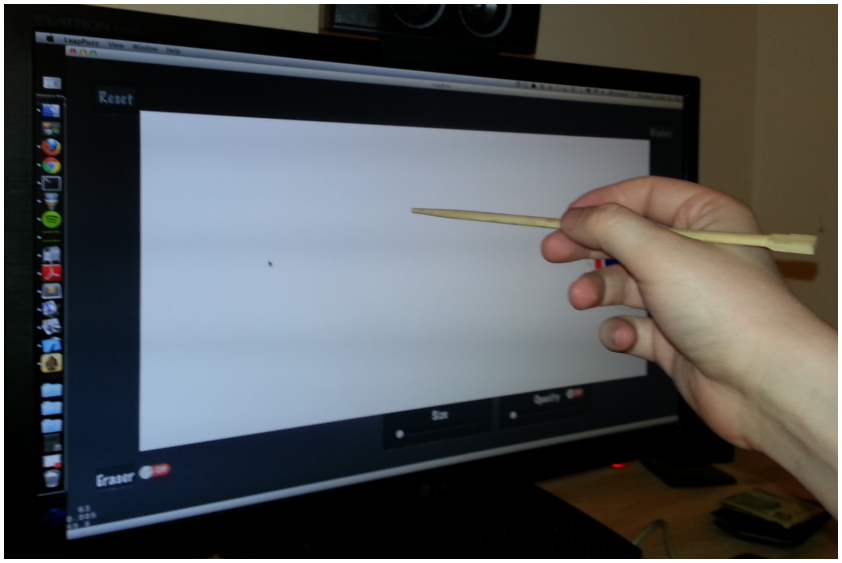


FIGURE 1.2: Using a chopstick as a pointer

This type of interaction with the computer system is different from the traditional keyboard and mouse because it does not require any physical contact with objects connected to the system and has the ability to sense a much wider range of input. The LeapMotion is also expressive in three dimensions as opposed to two.

In this paper and throughout the project we refer to a finger or tool interacting with LeapMotion as a "pointer[1]." The tool can be any object is "stick-like" such as a pen, pencil or chopstick. Throughout this project a chopstick was mostly used as seen in Figure 1.2 as the preferable tool.

1.3 Use Case Exceptions

Although we wanted to see if the keyboard and mouse could be replaced by the LeapMotion there are some functions that we understand would almost be certainly not possible. Tasks that require the user to type large amounts of information, such as data entry or word processing, are two such examples of this exception. The focus, therefore, would be on some of the other functions the keyboard may serve such as switching applications which can be performed by the keyboard shortcut ALT + TAB or with a mouse. The

[1]The documentation refers to this as a "pointable"

LeapMotion might be able to replace this functionality with a wave of a hand indicating to switch the applications automatically in carousel.

Chapter 2

Development Methodology

This project differed from traditional software development projects in that it worked closely with children participating in the design process of the application. The project worked closely with a group called KidsTeam. The children affiliated with this particular group participated in the application's design process. Because of this collaboration, the project's development process was different from the traditional projects in the way the phases were performed and required a very flexible model of development.

2.1 KidsTeam

The KidsTeam is a group of eight children ages 6 to 11, male and female, overseen by Dr. Jerry Fails at Montclair State University. The group met twice a week for one hour sessions over the course of a semester. During that time, the KidsTeam worked on various projects which were facilitated by Dr. Fails along with several undergraduate and graduate students. The objective was to use the children's natural capacity for divergent thinking in order to study and identify new ways of collaborative learning and development as a means of designing a fun educational game to help elementary and middle school aged students learn and practice basic math skills. In order to achieve an optimal outcome, the professor and students created a community of respect and rapport. The purpose was to create a safe space where the children felt free to explore

FIGURE 2.1: Reviewing the sticky notes using a spatial graph on a white board for easy comparison and analysis.

and share their ideas. Therefore, it was imperative that the atmosphere remain relaxed. This was enforced by rules that framed particular behaviors such as encouraging the children to speak when they have a thought, idea, or question rather than raising one's hand. The only strict requirement of the children during the sessions was that they must respect everyone, including the adults, in KidsTeam.

KidsTeam creates a dialog when designing and developing an application between the developers and the users by building off each others ideas. This intergenerational design team collaborates by exchanging ideas and giving opportunities to enhance every aspect of the application.

2.2 Collaboration Techniques

Several techniques were implemented in order to aid the development process. These techniques were chosen for their means to foster a community of cooperative inquiry where the children felt encouraged to contribute design ideas freely. [4][5]

2.2.1 Sticky Notes

Each child was given a pad of Post-It notes[1].During activities which required comment, as the children played with the cubes, the children would write comments on the PostIt notes[2]. These comments were categorized into like, dislike, or design idea and then stuck to the white board. A facilitator would then organize the notes based on the category and the comment content to resemble a spatial graph where similar comments are grouped

[1]Small square of paper with adhesive on reverse side.
[2]One comment per sticky note.

closer together, while outlying comments are spread farther apart. Comments relating to a specific function or component are arranged into the same row while the category of the comment will determine the column. At the end of the session the observer debriefs with the children about the session by reviewing the comments arranged on the white board and having the children comment about the session overall. This time allows for the observer and the children to summarize the session, gives the children extra time to comment and provide more specific feedback, and permits the observer to further clarify a child's reasoning for making certain comments. The result is a frequency analysis as seen in Figure 2.1 which feedback can be turned into specifications for the next design cycle. [6][7]

2.2.2 Bags of Stuff

Each child is given a Bag of Stuff that contains a variety of arts and crafts supplies such as Popsicle sticks, felt, construction paper, markers, etc. The children are then given a design concept by the observer and asked to use these materials to construct that concept. As the children build, they must explain in their own words how their concept works while the observer takes notes. [6][7]

2.2.3 Storyboarding

The children are split into groups. The groups were chosen at random and did not remain intact from session to session. Each group then receives one large piece of construction paper. They will use the paper, sticky notes, and markers to draw their particular game concept sequence of events. Any actions or changes in a scene must be drawn in the order that they occur. The children must use arrows to delineate the progression of events. In order to make sure their ideas are conveyed clearly, the children are asked to be as descriptive as possible while facilitators take notes as the children work. [6][7]

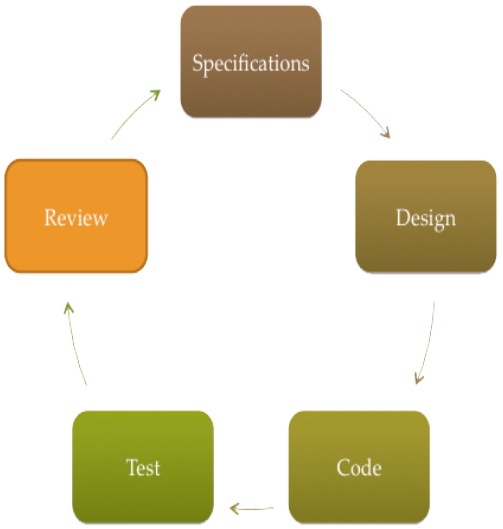


FIGURE 2.2: Rapid Application Development cycle

2.3 Development Model

Working with KidsTeam required a lot of flexibility in the development process and is why Rapid Application Development (RAD) model[3] for software development was chosen as the best fit model for this project. The RAD model uses rapid prototyping[4] and continuous iterative cycles which allowed for testing with KidsTeam on a weekly basis. Each session with the KidsTeam generated new requirements and fixes to be implemented within tight time constraints prior to the next session.

The model centered around five phases Specification, Design, Code, Test and Review as seen in Figure 2.2 constituting a full cycle. Changes in the specification were then implemented and reflected in the application prototypes ready for the next session. Each session with KidsTeam marked the end of current cycle and start of the next in the development process. [8][9]

The children were most heavily involved in the specifications, design, test and review phases in each of the nine sessions. Some parts of the design and testing were done independently of the children but mostly elaborated either on their designs and ideas or were performed to ensure the application was stable enough for testing with the children.

[3]Iterative or incremental development process resembling an evolutionary pattern.
[4]Rapid Prototyping will produce a quick mockup for testing in each cycle.

2.3.1 Specification

The specification is the set of requirements for the application's functional components and use cases. This includes what the application must be able to do and defines the parameters the application must operate within.

2.3.2 Design

The design takes the specification and frames a way to implement them in the next phase of coding. Designing the application design includes the interface design and application architecture into components that will fulfill the specification requirements.

2.3.3 Code

Coding is the process of taking the design and implementing it into functional units of code that run the application.

2.3.4 Test

Includes writing test cases for the functional code and debugging issues within the code such that the application meetings the requirements in the specification. This process was done by developer.

KidsTeam performed fullstack testing. ¡expand upon¿

2.3.5 Review

After testing was performed by KidsTeam, the children provided direct feedback via one of the collaboration techniques. ¡expand upon¿

2.4 Progress Updates

Throughout the course of the project updates were posted on a wiki and on youtube. Each wiki update summarized the session with the children including goals, observations and new requirements taken from the session along with pictures of the session. Videos posted on youtube mostly described major changes within the application itself and intermediate project updates.

Source code was kept in a public github repository[5] along with the project documentation. Any additional project documents were kept here.

[5]Due to the number of prototypes, the repositories were later consolidated into one repository.

Chapter 3

Design and Development

The initial sessions centered around introduction of the LeapMotion exploring how it could be used with existing interfaces and designing new interfaces. In the later sessions the application and interface begin to take form and requirements will begin to solidify.

The initial sessions with the KidsTeam centered mostly around the introduction of the LeapMotion. The purpose was to help the children become comfortable with the technology through both brief explanation of how it works and independent exploration. It was important that the children spent time interacting with the technology as early on in the sessions as possible. The more comfortable the children were with using the LeapMotion, the more realistic their creations could be for the interface.

Later sessions focused on reviewing and improving the application. The application will not be able to make too many drastic design or architectural changes in these later phases.

3.1 Session 1: Introduction and Brainstorm

This first session with the children started with brainstorming ways to use a computer system or control an application without a keyboard and mouse and only using their

hands. Using the Bags of Stuff 2.2.2 exercise the children explored designing applications that could only be controlled using gestures.

The children developed several application and game ideas including Pong, Fruit Ninja, Temple Run, Virtual Pet, Internet Explorer, Paint and Maps. The applications were accompanied by many controlling gestures such as a chopstick with button, waving, grabbing, pointing, dragging, pull and release[1], scratching and striking.

Afterward the children were allowed to play with a LeapMotion visualizer of the LeapMotion at the end of the session. The visualizer is 3d rendering of what the LeapMotion detects in realtime.

3.2 Session 2: Testing Breakout and Designing a Paint Application

The children tested a simple BreakOut game which allowed them to interact with the LeapMotion with an application and get a feeling of how it works. With that experience the children were then tasked with designing a painting application that can draw, choose colors and brushes only using gestures to control the interface of their application using the storyboarding 2.2.3 technique.

Along with the interface the children came up with several gestures in controlling a variety of user interface controls but most of the gesture controls relied on techniques similar to the way a mouse would function in picking up, dragging and dropping an icons on the screen. Other designs required an action to be performed while pointing at the targeted user interface control indicating a beginning action or ending action[2] instead of waving of a hand, drawing a figure eight in the air or a slashing motion.

[1] Similar to the bird launching interface in Angry Birds
[2] begin and end actions

3.3 Gesture Design Challenges

The immediate challenge that faced gestures requiring beginning and ending triggers is how to interpret when each trigger takes place to perform the action. In the mouse and touch screen interfaces the typical design paradigm consists of a beginning action, intermediate action and ending action respectively representing the state in which the input is currently in.

TABLE 3.1: Apple Mac OS X and iOS Application Programming Interface for standard methods of input [2]

Action	OS X	iOS
Begin	MouseDown	TouchesBegan
Moved	MouseDragged	TouchesMoved
End	MouseUp	TouchesEnded
Alternate	–	TouchesCancelled

With the LeapMotion there was no standard way of detecting when each action is to be performed. With this challenge the children came up with the concept of using two fingers to indicate when an action would be performed. Using the thumb as the control mechanism to indicate the action state to be performed and the index finger to indicate the targeted interface control to act upon, the interface control could triggered based on its functionality. The gesture consisted of pulling the thumb flush to the hand while pointing at the user interface control to begin the action and releasing the thumb to indicate the end of the action.With this simple system the a gesture could perform a BeginAction, MovedAction, and EndAction on user interface controls.

Use case include some of the following control operations

- Icon Movement pulling the thumb flush to the hand would trigger the BeginAction "pickup" the icon and start dragging it on the screen. The icon could then be placed anywhere on the screen until releasing icon and "dropping" at that position by moving the thumb to a relaxed position no longer flush with the hand performing the EndAction.

(a) Open Thumb (b) Closed Thumb

FIGURE 3.1: Hand trigging actions

- Selecting a color. The BeginAction would present a popover that would remain open while the user pointed at the desired color until the EndAction. The EndAction would select the color that was pointed at when triggered.
- Radio Buttons. A quick BeginAction and EndAction trigger while pointing at the radio button would change the state of the selected item.

3.3.1 Implementation Challenges

In concept this idea seemed to be an ideal and natural way of interacting with the system but faced several challenges. When the thumb became flush with the hand the LeapMotion could no longer detect it as a finger. This unique challenge was difficult because by the thumb could not be accounted for in all cases. An option might be to use the thumbs absence as a trigger although this is prone to many false positive[3] when the user's hands where on the boarder line of the LeapMotions visibility range or has lost track of the thumb due to occlusion[4]. Reversing the actions and performing the opposite gesture did not feel natural. Holding the thumb closely to the rest of the hand is not relaxed state and also did not mimic the act of grasping something or triggering an action as people are more commonly used to.

[3]False positive indicates a given condition is present when it is not.
[4]Occlusion occurs when the a surface is hidden by blocking the line of sight.

3.3.2 Unity Testing

Touching two or more fingers together rendered them not recognizable by the LeapMotion and thus would not allow pinch gesture to be used. Alternatives might be calculating when two fingers are close to each other although the LeapMotion cannot differentiate the fingers into which is the index, middle, ring, pinky, and thumb fingers. The application would have to assume any two fingers were performing the gesture and would not allow for the natural separation between fingers unless all fingers were kept folded tight into the hand until needed.

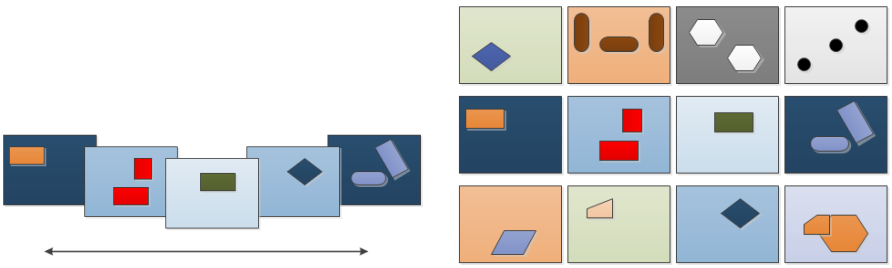
To observe this behavior a separate project was setup using the Unity 3D Engine to track the different pointers and their interactions in 3D space.[10]

It was clear after some lengthy prototyping and testing that some of the gestures developed by the children would not be possible and also raised some preliminary speculations that the LeapMotion may only be supplemental in its role except when in application specific situations. Different methods of interacting with these types of controls would need to be developed or would rely on the keyboard and mouse for their functionality.

3.4 Session 3: Interface Controls

Common user interface controls were printed out and the children were tasked with brainstorming how they might use LeapMotion to control the widgets or design their own widget that can reproduce the functionality. The user interface controls that we focused on included performing the following tasks were: buttons, drop downs, color palettes, check boxes, radio, buttons, sliders, toggles, steppers, wheels, trees and tabs.

The children did not come up with many new ways of interacting with the user interface controls or developing their own. The main method of interaction focused on pointing to the interface control triggering it using a hand signal with their fingers. The wheel controls could be controlled with flicking motions and dialogs could be dismissed with a wave indicating that the user is finished with the form. Selecting from a list could be done with a list in a carousel view style 3.2(a) where each of the items could be panned



(a) Carousel view style (b) Collection view style

FIGURE 3.2: Carousel view style maybe zoomed out using a pinch and pull gesture to a collection view.

through by waving or zoomed out into a collection view style 3.2(b) of items using pinch and pull gestures.

Observation of this session reinforced the suspicion that the LeapMotion may not be able to take a role as a dedicated device but as a supplemental device to the mouse an keyboard. Interface controls would also require more space between each of other and to be a of a larger size allowing for a margin of error. Feasible options may include iOS style of modal dialogs which take primary focus until the user in completed with the control.

3.5 Session 4: Testing a drawing application

Demonstration and hands on testing of a prototype drawing application developed based on the specifications the children had designed in the previous. The children provided feedback using the sticky notes exercise 2.2.1 review technique.

The one requirement that stood out the most early on was the necessity for a Heads Up Display (HUD) which would track where the a pointer is pointing on the screen and also display and interaction with the pointer. Additional features might include motion streaks or gesture animations to provide visual conformation of the action being performed or help the user track where the pointer has been.

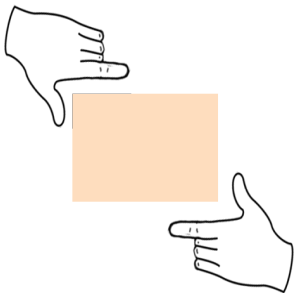


FIGURE 3.3: Indicating the size of a box by specifying two opposing corners with 'L' shape between the index and thumb fingers.

3.6 Session 5: Designing Gestures

The children were given some sample gestures and asked to design a way of using the gestures or generate their own for using Microsoft Paint application using the storyboarding 2.2.3 technique. The sample gestures were turning, tapping, swiping and hand wave. Gestures developed by the children were fairly consistent to previous session in pointing to a user interface control and selecting it to perform the action. In the group discussion an idea to use two fingers on two hands to draw the bounds of the box was generated. The finger tips could either represent each of the four corners of the box or two opposing corners by making an 'L' shape as seen in Figure 3.3.

This session reinforced the idea that the LeapMotion cannot be a dedicated device but a supplemental device.

3.7 Session 6: Testing

The children were tasked with testing the painting prototype application dubbed LeapPaint using the sticky notes exercise 2.2.1 review technique. This early prototype focused on adding a HUD to display the cursor and did not have all of the features in the initial requirements. Testing was focused on the interactions with the cursor. The While not testing they designed their own brushes, shapes and tools to be included in the next iteration of the tool. The overall consensus from this session was to implement better accuracy of the tool which required redesigning how the coordinate systems were translated from LeapMotion space into the application.

3.8 Session 7: Testing

The children were tasked with testing the painting application with the fixes based on the last session and providing feedback using the sticky notes 2.2.1 review technique. The new features focused on testing in the application were changing colors, erasing and an experimental method of triggering the drawing action.

The two different drawing actions were tested with one using the space bar to indicate when to begin drawing. The other used the Z axis of the LeapMotion to begin drawing when breaking a certain plane of depth toward the screen. Children could switch between the modes by pressing the '1' key for space bar mode and the '2' for depth mode and compare each of them.

After testing the children noted that it was difficult to determine where the plane was that would start and stop the drawing actions using the depth mode. The space bar mode seemed to be the preferable option of the two.

To help the children determine whether they were drawing or not a ring indicator would later be added to show when drawing and not drawing around the cursor by changing from green to red. Additionally the depth mode's flat plane along the Z and X axis would have to calculated as a concave shape for better performance since painting near the edges of the drawing required moving the arm slightly further toward the screen then required in the center.

3.9 Session 8: Testing

The children were tasked with testing the painting application for the last time using the sticky notes 2.2.1 review technique. Testing for new features which included a depth opacity control. The opacity of the brush would become greater the closer the pointer was to the monitor simulating how a paintbrush or marker makes a darker line when pressed harder to paper.

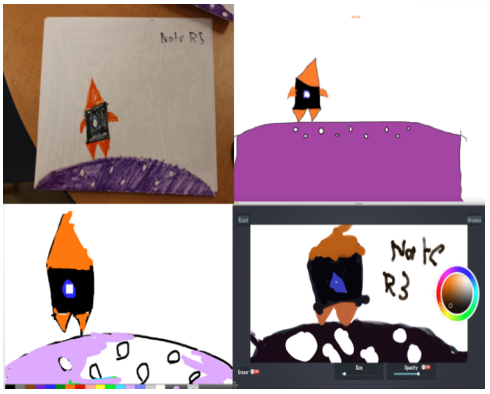


FIGURE 3.4: Drawings done by hand (top left), in Microsoft Paint (top right), on the iPad (bottom left) and with the LeapMotion (bottom right)

The children preferred the method of using depth to control the opacity over performing the manual adjustments with the mouse.

3.10 Session 9: Usability Comparison

The last session compared the LeapPaint application with a three different methods of drawing with different interfaces. The children were given the task of drawing the same picture of their choice on each system. Each child was given 10 minutes of time to attempt to reproduce a drawing by hand drawing using markers, Microsoft Paint using a mouse and keyboard, SimpleDraw on the iPad's touch interface and with LeapPaint using the LeapMotion.

The resulting drawings were mixed in comparison but showed that the drawing ability in the application may be related to the amount of time the child has had using the application.

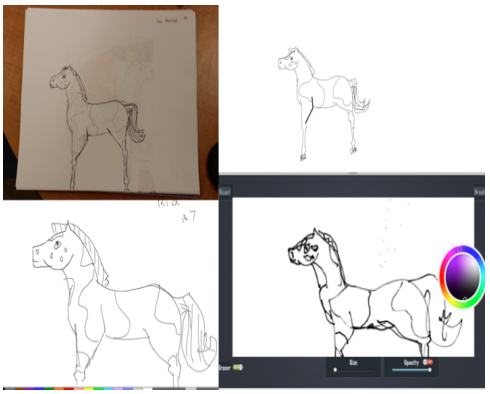


FIGURE 3.5: Drawings done by hand (top left), in Microsoft Paint (top right), on the iPad (bottom left) and with the LeapMotion (bottom right)

Chapter 4

Application Architecture

The changing requirements caused architecure to change several times over the course of the project as components were first prototyped then tested.

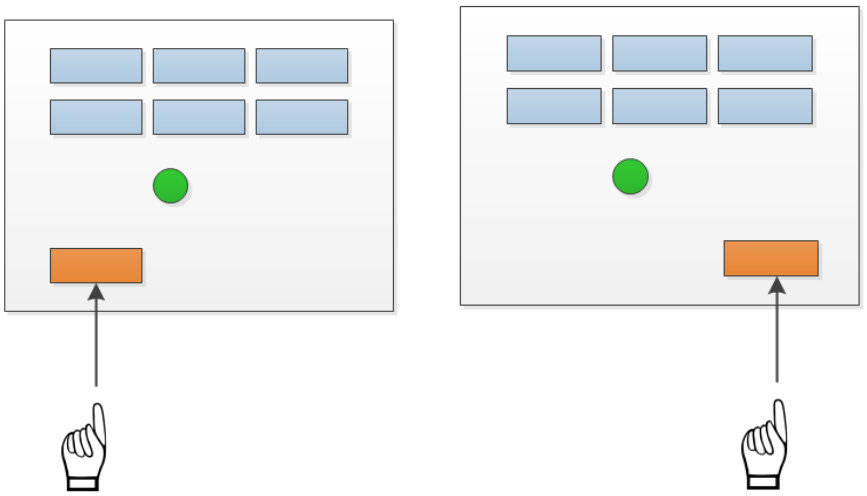
4.1 Prototyping

The initial prototypes included HelloWorld [1] application and simple game of BreakOut. HelloWorld and Breakout were both prototyped with the Cocos2d game engine because of the game engines ability to allow quick control over game objects. [11]

4.1.1 HelloWorld

The HelloWorld application tracked blocks across the screen using input data from the LeapMotion animating the interface. This application served as starting point for working with the LeapMotion SDK and also as a way of testing input received from the LeapMotion and resolving it to the coordinate space within the application. This code became a boiler plate interface to working with LeapMotion SDK later on in the project.

[1] HelloWorld is commonly a testbed to ensure the build environment and external libraries build correctly



(a) Left position (b) Right position

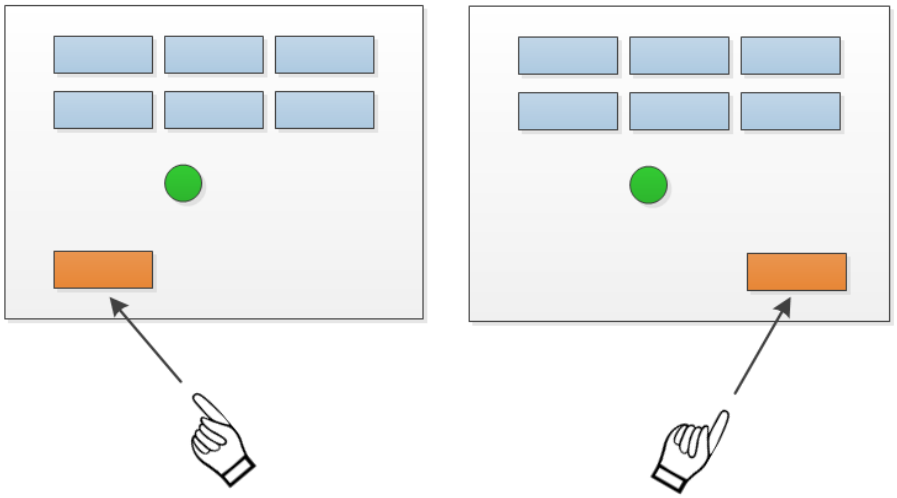
FIGURE 4.1: Left to right paddle control in breakout using the hand position relative to the screen

4.1.2 BreakOut

Furthering the HelloWorld application and testing the LeapMotion capabilities a simple game of BreakOut[2] was adapted for using the LeapMotion as the control mechanism for the paddle. This application was used in Session 2 3.2 with the children to show sample interactions of a complete system.

This prototype also showed through observation with the children how different methods of calculating the coordinates on the screen might be accomplished. The first method takes the position of the pointer in its coordinate space and translates it the coordinate space in the application as shown in Figure 4.1 despite the orientation of the pointer. The second method uses a vector pointing from the tip of the pointer and finds where that vector intersects with the screen as seen in Figure 4.2. Between the two methods the second method of finding the intersection on the screen appeared to be most natural.

[2]The breakout game was pre-built sample code[12]



(a) Left Position (b) Right position

FIGURE 4.2: Left to right paddle control in Breakout using the pointer intersection with the screen

4.1.3 Unity

One of the main gestures in the requirements used fingers to indicate beginning and ending actions as part of the gesture. Attempting to model these gestures required a 3D environment that could show multiple pointer interactions with game objects which was not possible with Cocos2d. The LeapMotion SDK provided a API with Unity 3D Game Engine which could perform the modeling required to begin recognizing the gestures. It was through this prototype that it was found that the LeapMotion would not be able to detect fingers touching each other[3]. [10]

4.1.4 Quartz 2D

The Cocos2d engine is not designed particularly for drawing and rendering textures to images. Apples Quartz 2D and Cocoa libraries are well suited for this task providing a large array of built in functionalities. These libraries were used to create the prototype drawing application using some of the boilerplate code from the earlier Helloworld 4.1.1

[3]A later released visualizer in the LeapMotion SDK would show the same resulut.

and Breakout 4.1.2 prototypes. The boilerplate code formed into a standard interface and coordinate system for working with the LeapMotion. [2]

This prototype worked well in testing in Session 4 3.5 and was generally well recieved by childeren although they had one major requirement of adding a cursor in addition to some minor features. The cursor would show where the pointer was on the screen at any given time so they could position it prior to painting. The minor features included selecting brushes, erasing, changing the brush size, changing the brush type and opacity.

4.1.4.1 Challenges

The cursor was very difficult to implement using the Quartz 2D and Cocoa because the libraries do not natively support layering and drawing simultaneously due to the way the rendering context functions as a single context. Creating independent views and transparent windows did not render correctly when attempting to simultaneously update each view. An alternative might be to put each functionality into separate applications running on different process threads although this was not possible because the LeapMotion can only be accessible from one process at a time. To create a cursor in HUD layer the application needed to access the LeapMotion, HUD and Drawing objects simultaneously. This was the defining factor in returning to the Cocos2d Game Library because it support independent layering of views. [2]

4.1.5 LeapPaint

The prototype dubbed "LeapPaint" by the children consisted of a combination components from the earlier Helloworld 4.1.1 and Breakout 4.1.2 prototypes managing input on a standard interface and output into different visible for drawing and the HUD. The layers were connected via a GameManager [4] passing the delegate actions between layers and interfaces.

[4] GameManager takes the role of the ApplicationDelegate

(a) Mock Up (b) Screenshot

FIGURE 4.3: Mock up compared to a screen shot of finished application with a drawing.

This prototype recieved the best reviews by the children in testing Session 6 3.7 despite lacking some of the features of available in the Quartz 2D 4.1.4. This architecture could be used going forward in adding features.

4.2 User Interface Layout

The interface layout of controls was based upon a combination of designs made by the children as seen in Figure 4.3 with the canvas to paint on centered and the user interface controls tucked to the sides of the canvas.

4.2.1 pointer Tracking

Initially a tracking system was design to use the relative coordinates of a pointer in the LeapMotion coordinate space and translate them to coordinates within the application. Later with the an SDK update the LeapMotion could provide coordinates on the screen where a vector from the pointers tip would intersect. This required some screen calibration to be performed such that the LeapMotion would be able to track points on the screen.

4.2.2 Application Layers

The application was broken into component layers to manage and modularize different functionalities as seen in Figure 4.4. This allowed for some components to become

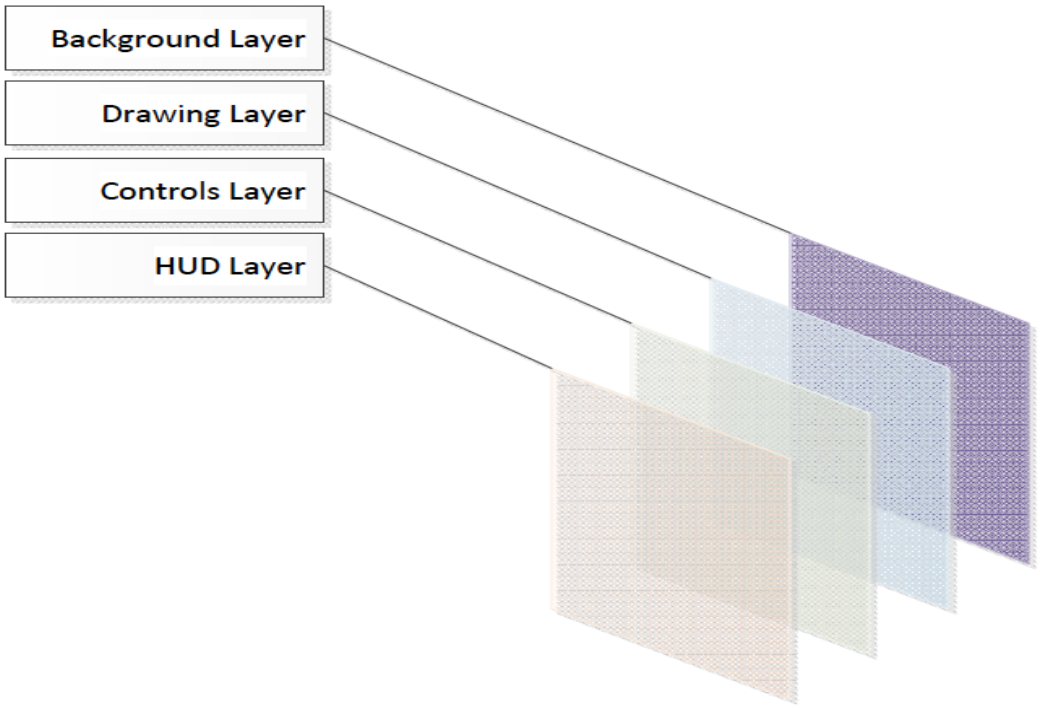


FIGURE 4.4: The ordered layers of visibility in the application.

reusable for other applications by providing a common application programming interface (API).

- Background Layer manages the background images and for the application. This practice is fairly standard and
- Drawing Layer is where the image will be edited and render.
- Controls Layer provides the user interface controls that for different aspects of the application.
- HUD Layer displays feedback information for the position of the pointer in relation to the screen and any different actions that might be taking place.

4.3 External Libraries

- LeapMotion SDK is the provided API for using with the LeapMotion.[3]

- Cocos2d is a game engine framework that allows simple animation and control of sprites and layers. [11]
- CCControlExtensions are user interface control elements built for the Cocos2d Engine. [13]
- Graphics Recognition Toolkit was used initially for doing gesture recognition with customized pipeline until a LeapMotion SDK update provided the same functionality. [14]

4.4 Testing

The project uses unit testing on all the class functions to verify their input and outputs correspond to their intended function. Using the simple methodology of making the test fail and then making the test pass with a variety of inputs and outputs helped modularize class functionality into smaller and more reusable sections. The testing framework OCUnit tests at class and bundle levels to produce full code test coverage. [2]

Parts of the project unit testing could not cover were mostly involved at the interface level where the LeapMotion SDK provided data. Environmental factors effect the LeapMotions performance when in a different lighting conditions. The different types and positions of lighting caused erratic effects that often had to be compensated for when noticed.

4.5 Documentation

Code documentation is done with in-line comments and then automatically generated with Doxygen. This was an essential tool due to the ever changing state of the project during each cycle that the project be able to automatically reflect changes in the design.

4.6 Experimental Features

Several experimental interface designs were added for the children to test with since there were often more than one design approach to a features in the application. The approached attempted to leverage the 3D space available to the LeapMotion that is not available to the keyboard and mouse.

4.6.1 Drawing Modes

Two different drawing modes were tested with the children in Session 7 3.8. The first was a mode where the pointer would begin drawing when crossing a boundary on the Z axis toward the screen. The cursor could then be moved about the screen without interacting with the drawing by pulling the pointer back and then pushing forward when ready to begin drawing. A ring around the cursor icon would indicate weather or not the cursor would begin drawing based on the depth of pointer. The ring would change from red when not in not drawing state to green when beginning to draw. Further development might include a yellow color ring indicator when approaching the threshold to transition states.

The children did not like using the depth mode option of drawing compared to the spacebar mode bar of drawing as they had trouble becoming accustom to the threshold in which the application would begin and stop drawing. They did like that they could draw and change colors at the same time by using their free hand with the mouse to create continuous rainbow effects with the brush.

The second drawing mode was to begin drawing when pressing the space bar on the keyboard and disregarding the depth of the pointer. This proved to be the favorite method of input for the children to indicate their actions.

4.6.2 Depth Opacity

Another option explored was using the Z axis to control the opacity of the brush. The intent was to provide a pressure sensitive brush which would mimic drawing implements

in the real world where pressing harder on a paint brush or marker will draw a darker or thicker line.

The children did like this feature although and preferred it to manually adjusting the opacity with the mouse. This was Dependant on what they were drawing where it was preferable for background colors but not drawing lines. The feature could be improved with adding some brush stroke effects that vary based on the speed of the pointer when drawing.

Chapter 5

Summary

5.1 General Observations

The children would first sketch an outline of their drawing and then attempt to color in their outlines. This proved difficult in the case of the LeapMotion. Drawing the lines proved fairly easy for the children while shading in sections appeared more difficult. I found the opposite to be true of my own experience as the lines were harder to draw than shading in the areas.

5.2 Similar Applications

Comparison to industry competitors Corel's Painter Freestyle which will have many of the same features. In terms of interface design they chose a similar layout of control mechanisms. We haven't be able to to see this all quiet yet since it has not been released to perform a full comparison although from the initial details given on their website seems to lead that their application could be similar to ours in many respects. [**?**]

5.2.1 Google Earth

Examples of dedication are shown in some example applications where the LeapMotion has a specific purpose. Google Earth uses it for navigation only allowing the user to pan, rotate and elevated the camera in relation to the earth. Interpreting the hand motions as the path of airplane as the control mechanism with yaw, pitch, roll, bank and elevation.

The way of controlling the Google Earth is similar to the idea of controlling the Templerun game brainstormed in from Session 1 3.1. This connection was not apparent when first considering the idea for controlling the game.

5.3 Conclusions

The LeapMotion is a great device for capturing the motions and positions of the hand in real time but is tough to consider as replacement for the keyboard and mouse. It is foreseeable that the main application for the LeapMotion will not be as a general use device but for specific applications which enable expressive movement. Games, drawing and music applications are good examples of where the LeapMotion can focus on specialized actions. Integration into general purpose applications will be difficult due to existing designs and interface paradigms.

5.4 Licenses

Appendix A

Documentation

Contents

1 Main Page

`Project Home & Wiki`

#Requirements Specification

Interface

- HUD Requirement to render a cursor where the pointable is intersecting with the screen. The cursor should show the color that will be painting on the screen
- Ring and round cursor to indicate drawing or not drawing.

Features

- Change Colors
- Change Brushes
- Eraser
- Change size of brush
- Reset drawing
- Change Opacity of brushes

#Unit Tests

#Libraries & Sub Modules

- Cocos2d 2.0
- CCControlExtension
- #Build Settings
- Valid Architecture i386 x86_64
- Other Linker Flags -lz -ObjC
- C Language Dialect GNU99 -std=gnu99
- C ++ Language Dialect GNU++11 -std=gnu++11
- C ++ Standard Library libc++ (LLVM C++ standard lib)
- run script after build:

  echo TARGET_BUILD_DIR=${TARGET_BUILD_DIR} echo TARGET_NAME=${TARGET_NAME} cd ${TARGET_BUILD_DIR}/${TARGET_NAME}.app/Contents/MacOS ls -la install_name_tool -change /libLeap.dylib /../Resources/libLeap.dylib ${TARGET_NAME}

#Documentation

Documentation is done using Doxygen

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

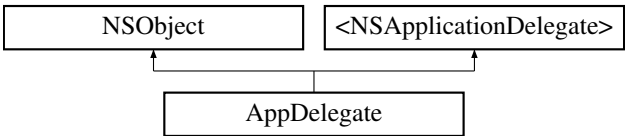
3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 Class Documentation

4.1 AppDelegate Class Reference

Inheritance diagram for AppDelegate:



Properties

- IBOutlet NSWindow ∗ **window**

4.1.1 Detailed Description

Definition at line 11 of file AppDelegate.h.

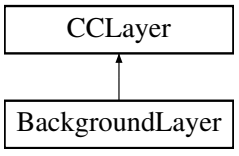
The documentation for this class was generated from the following file:

- LeapPaint/AppDelegate.h

4.2 BackgroundLayer Class Reference

```
#import <BackgroundLayer.h>
```

Inheritance diagram for BackgroundLayer:



4.2.1 Detailed Description

Background Layer Displays a background image for the scene

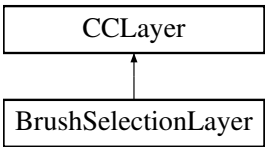
Definition at line 16 of file BackgroundLayer.h.

The documentation for this class was generated from the following file:

- LeapPaint/BackgroundLayer.h

4.3 BrushSelectionLayer Class Reference

Inheritance diagram for BrushSelectionLayer:



Protected Attributes

- NSMutableDictionary ∗ **imageNamesDictionary**

Properties

- id< BrushSelectionLayerDelegate > **delegate**
- bool **layerHidden**

4.3.1 Detailed Description

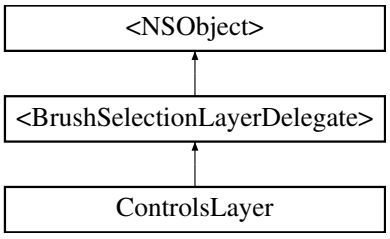
Definition at line 18 of file BrushSelectionLayer.h.

The documentation for this class was generated from the following file:

- LeapPaint/BrushSelectionLayer.h

4.4 <BrushSelectionLayerDelegate> Protocol Reference

Inheritance diagram for <BrushSelectionLayerDelegate>:



Instance Methods

- (void) - **hidePanel**
- (void) - **brushSelected:**

4.4.1 Detailed Description

Definition at line 12 of file BrushSelectionLayer.h.

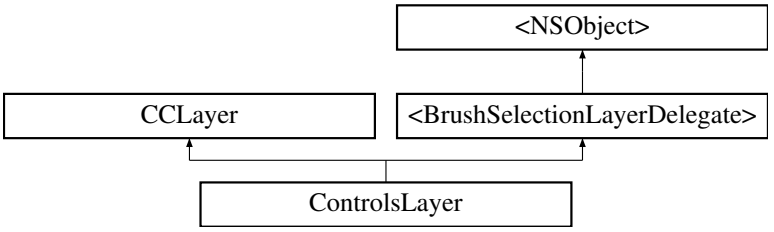
The documentation for this protocol was generated from the following file:

- LeapPaint/BrushSelectionLayer.h

4.5 ControlsLayer Class Reference

`#import <ControlsLayer.h>`

Inheritance diagram for ControlsLayer:



Instance Methods

- (void) - valueChanged:
- (void) - opacitySliderChanged:
- (void) - expandPanel
- (void) - collapsePanel
- (CCControlSwitch ∗) - makeControlSwitch
- (void) - switchValueChanged:
- (void) - updateOpacitySlider:

Protected Attributes

- CCLabelTTF ∗ colorLabel
- GameSettings ∗ gameSettings

Properties

- CCControlSlider ∗ slider
- CCControlSlider ∗ opacitySlider
- CCControlSwitch ∗ opacitySwitchControl
- CCLabelTTF ∗ opacitydisplayValueLabel
- id< ControlsLayerDelegate > delegate
- BrushSelectionLayer ∗ brushSelection
- CCLabelTTF ∗ displayValueLabel
- CCControlSwitch ∗ switchControl

4.5.1 Detailed Description

Controls Layer User inferface controls for operating buttons, switches, sliders

Definition at line 34 of file ControlsLayer.h.

4.5.2 Method Documentation

4.5.2.1 - (void) collapsePanel

Collapses Brushes Panel

Definition at line 445 of file ControlsLayer.mm.

```
00445                       {
00446
00447 }
```

4.5.2.2 - (void) expandPanel

Expands brushes panel

Definition at line 440 of file ControlsLayer.mm.

```
00440                     {
00441
00442
00443 }
```

4.5.2.3 - (CCControlSwitch *) makeControlSwitch

Creates and returns a new CCControlSwitch.

Definition at line 492 of file ControlsLayer.mm.

```
00493 {
00494     return [CCControlSwitch switchWithMaskSprite:[CCSprite spriteWithFile:@"switch-mask.png"]
00495                                           onSprite:[CCSprite spriteWithFile:@"switch-on.png"]
00496                                          offSprite:[CCSprite spriteWithFile:@"switch-off.png"]
00497                                       thumbSprite:[CCSprite spriteWithFile:@"switch-thumb.png"]
00498                                           onLabel:[CCLabelTTF labelWithString:@"On" fontName:@"Arial-BoldMT"
       fontSize:16]
00499                                          offLabel:[CCLabelTTF labelWithString:@"Off" fontName:@"Arial-BoldMT
      " fontSize:16]];
00500 }
```

4.5.2.4 - (void) opacitySliderChanged: (CCControlSlider *) *sender*

Does something

Parameters

slider	changes

Definition at line 142 of file ControlsLayer.mm.

```
00142                               :(CCControlSlider *)sender
00143 {
00144
00145     // Change value of label.
00146     //NSLog(@"slider value %@", [NSString stringWithFormat:@"Slider value = %.02f", sender.value]);
00147     [self.delegate changeOpacityControl:sender.value];
00148 }
```

4.5.2.5 - (void) switchValueChanged: (CCControlSwitch *) *sender*

Callback for the change value.

Definition at line 503 of file ControlsLayer.mm.

```
00503                           :(CCControlSwitch *)sender
00504 {
00505     if ([sender isOn])
00506     {
00507         displayValueLabel.string    = @"Eraser";
00508 
00509         [self.delegate eraserMode:true];
00510     } else
00511     {
00512         displayValueLabel.string    = @"Eraser";
00513         [self.delegate eraserMode:false];
00514     }
00515 }
```

4.5.2.6 - (void) updateOpacitySlider: (float) *value*

Callback for opacity changing with the slider

Definition at line 150 of file ControlsLayer.mm.

```
00150                            :(float)value{
00151 
00152 
00153     //ensure the value is within its bounds
00154     if(value > self.opacitySlider.maximumValue){
00155         //Max Value
00156         self.opacitySlider.value = self.opacitySlider.maximumValue;
00157     }else if(value < self.opacitySlider.minimumValue){
00158         //Min Value
00159         self.opacitySlider.value = self.opacitySlider.minimumValue;
00160     }else{
00161         self.opacitySlider.value = value;
00162     }
00163 }
```

4.5.2.7 - (void) valueChanged: (CCControlSlider *) *sender*

Does something

Parameters

slider	changes

Definition at line 95 of file ControlsLayer.mm.

```
00095                       :(CCControlSlider *)sender
00096 {
00097     // Change value of label.
00098     //NSLog(@"slider value %@", [NSString stringWithFormat:@"Slider value = %.02f", sender.value]);
00099     [self.delegate changeThicknessControl:sender.value];
00100 }
```

4.5.3 Member Data Documentation

4.5.3.1 - (CCLabelTTF*) colorLabel [protected]

colorLabel displays name of color in hash value

Definition at line 36 of file ControlsLayer.h.

4.5.3.2 - (GameSettings*) gameSettings [protected]

gameSettings global reference to shared settings instance

Definition at line 38 of file ControlsLayer.h.

4.5.4 Property Documentation

4.5.4.1 - (BrushSelectionLayer*) brushSelection [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 45 of file ControlsLayer.h.

4.5.4.2 - (id<ControlsLayerDelegate>) delegate [read], [write], [nonatomic], [weak]

colorLabel displays name of color in hash value

Definition at line 44 of file ControlsLayer.h.

4.5.4.3 - (CCLabelTTF *) displayValueLabel [read], [write], [nonatomic], [strong]

displayValueLabel displays coordinate

colorLabel displays name of color in hash value

Definition at line 37 of file ControlsLayer.h.

Referenced by switchValueChanged:.

4.5.4.4 - (CCLabelTTF*) opacitydisplayValueLabel [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 43 of file ControlsLayer.h.

4.5.4.5 - (CCControlSlider*) opacitySlider [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 41 of file ControlsLayer.h.

Referenced by updateOpacitySlider:.

4.5.4.6 - (CCControlSwitch*) opacitySwitchControl [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 42 of file ControlsLayer.h.

4.5.4.7 - (CCControlSlider*) slider [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 40 of file ControlsLayer.h.

4.5.4.8 - (CCControlSwitch*) switchControl [read], [write], [nonatomic], [strong]

colorLabel displays name of color in hash value

Definition at line 47 of file ControlsLayer.h.

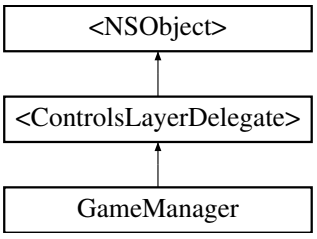
The documentation for this class was generated from the following files:

- LeapPaint/ControlsLayer.h
- LeapPaint/ControlsLayer.mm

4.6 <ControlsLayerDelegate> Protocol Reference

`#import <ControlsLayer.h>`

Inheritance diagram for <ControlsLayerDelegate>:



Instance Methods

- (void) - **changeColorControl:**
- (void) - **changeThicknessControl:**
- (void) - **changeBrushControl:**
- (void) - **changeOpacityControl:**
- (void) - **clearDrawing**
- (void) - **eraserMode:**

4.6.1 Detailed Description

Controls Layer Delegate

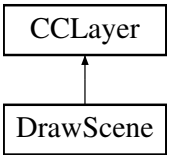
Definition at line 19 of file ControlsLayer.h.

The documentation for this protocol was generated from the following file:

- LeapPaint/ControlsLayer.h

4.7 DrawScene Class Reference

Inheritance diagram for DrawScene:



Protected Attributes

- LeapController * **controller**
- CCTexture2D * **spriteTexture_**
- b2World * **world**
- GLESDebugDraw * **m_debugDraw**
- CCSprite * **targetSprite**

- b2MouseJoint * **_mouseJoint**
- b2World * **_world**
- b2Body * **_groundBody**
- NSMutableDictionary * **trackableList**

4.7.1 Detailed Description

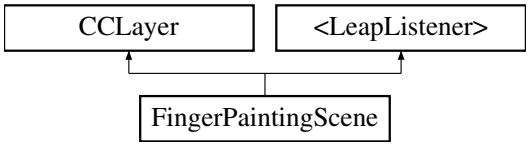
Definition at line 15 of file DrawScene.h.

The documentation for this class was generated from the following file:

- LeapPaint/DrawScene.h

4.8 FingerPaintingScene Class Reference

Inheritance diagram for FingerPaintingScene:



Protected Attributes

- LeapController * **controller**
- CCTexture2D * **spriteTexture_**
- b2World * **world**
- GLESDebugDraw * **m_debugDraw**
- CCSprite * **targetSprite**
- b2MouseJoint * **_mouseJoint**
- b2World * **_world**
- b2Body * **_groundBody**
- CIColor * **brushColor**
- NSMutableDictionary * **trackableList**
- NSMutableDictionary * **brushesList**
- NSTimer * **updateDraw**
- RedDot * **mouseCursor**

4.8.1 Detailed Description

Definition at line 17 of file FingerPaintingScene.h.

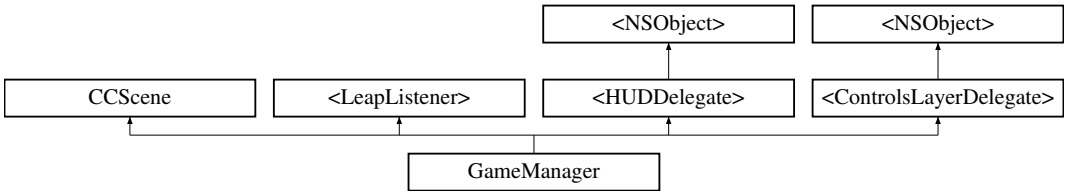
The documentation for this class was generated from the following file:

- LeapPaint/FingerPaintingScene.h

4.9 GameManager Class Reference

`#import <GameManager.h>`

Inheritance diagram for GameManager:



Instance Methods

- (float) - **findPecentageDifference:withMin:withValue:**
- (float) - opacityPercentage:

Protected Attributes

- InputMode inputMode
- LeapPointable * currentPointable
- CGPoint currentPoint
- BOOL **painting**
- GameSettings * gameSettings
- int lastTag
- SimplePoint * lastPoint
- int framesSinceLastFound

Properties

- HUDLayer * hudLayer
- SketchRenderTextureScene * textureScene
- BackgroundLayer * backgroundLayer
- ControlsLayer * controlsLayer
- LeapController * controller
- LeapScreen * leapScreen

4.9.1 Detailed Description

Core Application Management Provides interfaces and controls the various inputs, controls and outputs

Definition at line 27 of file GameManager.h.

4.9.2 Method Documentation

4.9.2.1 - (float) opacityPercentage: (float) *value*

Return the Opacity value based on Z position

Definition at line 328 of file GameManager.mm.

```
00328                           :(float)value{
00329     //NSLog(@"value %0.0f", value);
00330     if (value < kOpMinRange){
00331         return kOpMax;
00332     }else if(value > kOpMaxRange){
00333         return kOpMin;
00334     }else {
00335
00336         float percentage = [self findPecentageDifference:kOpMaxRange withMin:kOpMinRange withValue:value];
00337         //NSLog(@"percentage %0.0f", percentage);
00338
00339         percentage = 100 - percentage;
00340
00341         return percentage;
00342
00343     }
00344
00345 }
```

4.9.3 Member Data Documentation

4.9.3.1 - (CGPoint) currentPoint [protected]

colorLabel displays name of color in hash value

Definition at line 32 of file GameManager.h.

4.9.3.2 - (LeapPointable*) currentPointable [protected]

colorLabel displays name of color in hash value

Definition at line 31 of file GameManager.h.

4.9.3.3 - (int) framesSinceLastFound [protected]

framesSinceLastFound number of frames since last finding a LeapPointable

Definition at line 41 of file GameManager.h.

4.9.3.4 - (GameSettings*) gameSettings [protected]

gameSettings singleton to global seetings

Definition at line 36 of file GameManager.h.

4.9.3.5 - (InputMode) inputMode [protected]

colorLabel displays name of color in hash value

Definition at line 30 of file GameManager.h.

4.9.3.6 - (SimplePoint*) lastPoint [protected]

lastPoint is the last known point on the screen of the LeapPointable

Definition at line 40 of file GameManager.h.

4.9.3.7 - (int) lastTag [protected]

lastTag is the last tag value tracked of a LeapPointable

Definition at line 39 of file GameManager.h.

4.9.4 Property Documentation

4.9.4.1 - (BackgroundLayer*) backgroundLayer [read], [write], [nonatomic], [strong]

backgroundLayer is the layer for setting up the background

Definition at line 48 of file GameManager.h.

4.9.4.2 - (LeapController*) controller [read], [write], [nonatomic], [strong]

controller is the leapController

Definition at line 51 of file GameManager.h.

4.9.4.3 - (ControlsLayer*) controlsLayer [read], [write], [nonatomic], [strong]

controlsLayer is the layer for managing interface controls

Definition at line 49 of file GameManager.h.

4.9.4.4 - (HUDLayer*) hudLayer [read], [write], [nonatomic], [strong]

hudLayer displays the icons for tracking where a leapPointable is pointing

Definition at line 46 of file GameManager.h.

4.9.4.5 - (LeapScreen*) leapScreen [read], [write], [nonatomic], [strong]

leapScreen references the screen being used on the system

Definition at line 52 of file GameManager.h.

4.9.4.6 - (SketchRenderTextureScene*) textureScene [read], [write], [nonatomic], [strong]

textureScene is the drawing layer

Definition at line 47 of file GameManager.h.

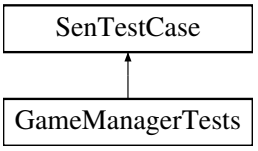
The documentation for this class was generated from the following files:

- LeapPaint/GameManager.h
- LeapPaint/GameManager.mm

4.10 GameManagerTests Class Reference

`#import <GameManagerTests.h>`

Inheritance diagram for GameManagerTests:



Protected Attributes

- GameManager * node

4.10.1 Detailed Description

Tests the SimplePoint object

Definition at line 15 of file GameManagerTests.h.

4.10.2 Member Data Documentation

4.10.2.1 - (GameManager∗) node [protected]

gameManager instance

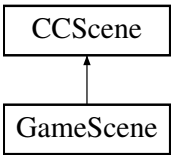
Definition at line 16 of file GameManagerTests.h.

The documentation for this class was generated from the following file:

- LeapPaint/GameManagerTests.h

4.11 GameScene Class Reference

Inheritance diagram for GameScene:



Class Methods

- (CCScene ∗) + **scene**

4.11.1 Detailed Description

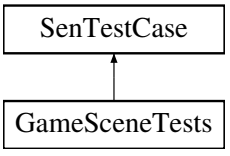
Definition at line 20 of file GameScene.h.

The documentation for this class was generated from the following files:

- LeapPaint/GameScene.h
- LeapPaint/GameScene.mm

4.12 GameSceneTests Class Reference

Inheritance diagram for GameSceneTests:



4.12.1 Detailed Description

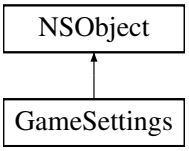
Definition at line 11 of file GameSceneTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/GameSceneTests.h

4.13 GameSettings Class Reference

Inheritance diagram for GameSettings:



Class Methods

- (GameSettings ∗) + sharedInstance

Properties

- BOOL depthOpacityMode
- BOOL eraserMode
- InputMode inputMode

4.13.1 Detailed Description

Definition at line 35 of file GameSettings.h.

4.13.2 Method Documentation

4.13.2.1 + (GameSettings ∗) sharedInstance

Singleton Intiailizes and Returns a shared instance of the class

Definition at line 23 of file GameSettings.mm.

```
00024 {
00025     static GameSettings *sharedInstance;
00026
00027     @synchronized(self)
00028     {
00029         if (!sharedInstance)
00030             sharedInstance = [[GameSettings alloc] init];
00031
00032         return sharedInstance;
00033     }
00034 }
```

4.13.3 Property Documentation

4.13.3.1 - (BOOL) depthOpacityMode [read], [write], [nonatomic], [assign]

depthOpacityMode controls use of z axis control of opacity

Definition at line 40 of file GameSettings.h.

4.13.3.2 - (BOOL) eraserMode [read], [write], [nonatomic], [assign]

eraserMode controls erasing on drawing canvas

Definition at line 41 of file GameSettings.h.

4.13.3.3 - (InputMode) inputMode [read], [write], [nonatomic], [assign]

inputMode controller input mode for leapmotion

Definition at line 42 of file GameSettings.h.

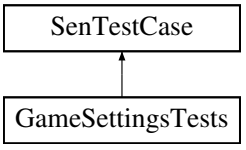
The documentation for this class was generated from the following files:

- LeapPaint/GameSettings.h
- LeapPaint/GameSettings.mm

4.14 GameSettingsTests Class Reference

`#import <GameSettingsTests.h>`

Inheritance diagram for GameSettingsTests:



Protected Attributes

- GameSettings * gameSettings

4.14.1 Detailed Description

Tests the GameSettings object

Definition at line 16 of file GameSettingsTests.h.

4.14.2 Member Data Documentation

4.14.2.1 - (GameSettings*) gameSettings [protected]

gameSettings singleton instance

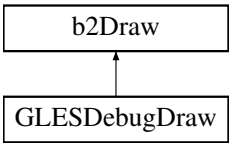
Definition at line 18 of file GameSettingsTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/GameSettingsTests.h

4.15 GLESDebugDraw Class Reference

Inheritance diagram for GLESDebugDraw:



Public Member Functions

- **GLESDebugDraw** (float32 ratio)
- void **DrawPolygon** (const b2Vec2 ∗vertices, int32 vertexCount, const b2Color &color)
- void **DrawSolidPolygon** (const b2Vec2 ∗vertices, int32 vertexCount, const b2Color &color)
- void **DrawCircle** (const b2Vec2 &center, float32 radius, const b2Color &color)
- void **DrawSolidCircle** (const b2Vec2 &center, float32 radius, const b2Vec2 &axis, const b2Color &color)
- void **DrawSegment** (const b2Vec2 &p1, const b2Vec2 &p2, const b2Color &color)
- void **DrawTransform** (const b2Transform &xf)
- void **DrawPoint** (const b2Vec2 &p, float32 size, const b2Color &color)
- void **DrawString** (int x, int y, const char ∗string,...)
- void **DrawAABB** (b2AABB ∗aabb, const b2Color &color)

4.15.1 Detailed Description

Definition at line 43 of file GLES-Render.h.

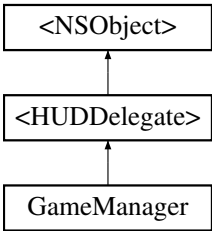
The documentation for this class was generated from the following files:

- LeapPaint/GLES-Render.h
- LeapPaint/GLES-Render.mm

4.16 <HUDDelegate> Protocol Reference

```
#import <HUDLayer.h>
```

Inheritance diagram for <HUDDelegate>:



Instance Methods

- (void) - **changeMode:**
- (void) - **painting:**

4.16.1 Detailed Description

HUD Delegate Protocol User inferface controls for operating buttons, switches, sliders

Definition at line 20 of file HUDLayer.h.

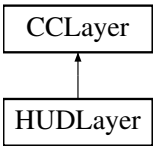
The documentation for this protocol was generated from the following file:

- LeapPaint/HUDLayer.h

4.17 HUDLayer Class Reference

`#import <HUDLayer.h>`

Inheritance diagram for HUDLayer:



Instance Methods

- (void) - **toolMoved:toolID:**
- (void) - **startTrackingTool:toolID:**
- (void) - **moveTrackingTool:toolID:**
- (void) - **endTrackingTool**
- (void) - **changeColor:**
- (void) - **changeBrush:**
- (void) - **changeScale:**
- (void) - **erasingMode:**

Protected Attributes

- NSString * primaryToolID
- LPTool * primaryTool
- InputMode inputMode
- ccColor3B lastColor
- ccColor3B previousColor
- NSString * lastBrush
- float lastScale
- CCSprite * paintingIndicator
- BOOL eraseMode
- GameSettings * gameSettings

Properties

- id< HUDDelegate > delegate
- CCLabelTTF * **xyzcoords**

4.17.1 Detailed Description

HUD Layer Tracks the position of the LeapCursor on the screen

Definition at line 30 of file HUDLayer.h.

4.17.2 Member Data Documentation

4.17.2.1 - (BOOL) eraseMode [protected]

eraseMode determines weather the pointable is painting or erasing

Definition at line 44 of file HUDLayer.h.

4.17.2.2 - (GameSettings*) gameSettings [protected]

gameSettings singleton to global seetings

Definition at line 47 of file HUDLayer.h.

4.17.2.3 - (InputMode) inputMode [protected]

inputMode is the current mode of input

Definition at line 34 of file HUDLayer.h.

4.17.2.4 - (NSString*) lastBrush [protected]

lastBrush is last brush to be selected

Definition at line 38 of file HUDLayer.h.

4.17.2.5 - (ccColor3B) lastColor [protected]

lastColor is the lastColor to be selected

Definition at line 36 of file HUDLayer.h.

4.17.2.6 - (float) lastScale [protected]

lastScale is last scale to be selected

Definition at line 39 of file HUDLayer.h.

4.17.2.7 - (CCSprite*) paintingIndicator [protected]

paintingIndicator shows the state at which the object is currently paintg

Definition at line 43 of file HUDLayer.h.

4.17.2.8 - (ccColor3B) previousColor [protected]

previousColor is the color before the lastcolor to be selected

Definition at line 37 of file HUDLayer.h.

4.17.2.9 - (LPTool*) primaryTool [protected]

primaryTool points to the current pointable object

Definition at line 32 of file HUDLayer.h.

4.17.2.10 - (NSString*) primaryToolID [protected]

primaryToolID stores the id tag to the pointable in reference

Definition at line 31 of file HUDLayer.h.

4.17.3 Property Documentation

4.17.3.1 - (id<HUDDelegate>) delegate [read], [write], [nonatomic], [weak]

colorLabel displays name of color in hash value

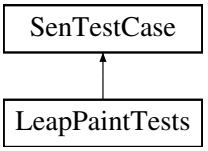
Definition at line 52 of file HUDLayer.h.

The documentation for this class was generated from the following files:

- LeapPaint/HUDLayer.h
- LeapPaint/HUDLayer.mm

4.18 LeapPaintTests Class Reference

Inheritance diagram for LeapPaintTests:



4.18.1 Detailed Description

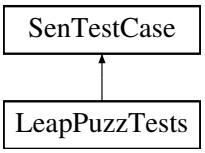
Definition at line 11 of file LeapPaintTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/LeapPaintTests.h

4.19 LeapPuzzTests Class Reference

Inheritance diagram for LeapPuzzTests:



4.19.1 Detailed Description

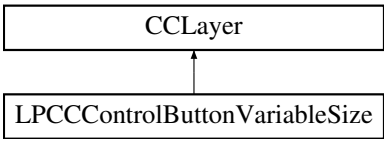
Definition at line 11 of file LeapPuzzTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/LeapPuzzTests.h

4.20 LPCCControlButtonVariableSize Class Reference

Inheritance diagram for LPCCControlButtonVariableSize:



Instance Methods

- (CCControlButton ∗) - standardButtonWithTitle:

4.20.1 Detailed Description

Definition at line 11 of file LPCCControlButtonVariableSize.h.

4.20.2 Method Documentation

4.20.2.1 - (CCControlButton ∗) standardButtonWithTitle: (NSString ∗) *title*

Creates and return a button with a default background and title color. Creates and return a button with a default background and title color.

Definition at line 57 of file LPCCControlButtonVariableSize.m.

```
00057                                           :(NSString *)title
00058 {
00060     CCScale9Sprite *backgroundButton = [CCScale9Sprite spriteWithFile:@"button.png"];
00061     CCScale9Sprite *backgroundHighlightedButton = [CCScale9Sprite spriteWithFile:@"buttonHighlighted.png"];
00062
00063 #ifdef __IPHONE_OS_VERSION_MAX_ALLOWED
00064     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:title fontName:@"HelveticaNeue-Bold" fontSize:30]
      ;
00065 #elif __MAC_OS_X_VERSION_MAX_ALLOWED
00066     CCLabelTTF *titleButton = [CCLabelTTF labelWithString:title fontName:@"Marker Felt" fontSize:30];
00067 #endif
00068     [titleButton setColor:ccc3(159, 168, 176)];
00069
```

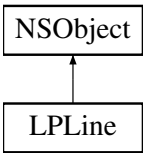
```
00070     CCControlButton *button = [CCControlButton buttonWithLabel:titleButton backgroundSprite:
      backgroundButton];
00071     [button setBackgroundSprite:backgroundHighlightedButton forState:CCControlStateHighlighted];
00072     [button setTitleColor:ccWHITE forState:CCControlStateHighlighted];
00073
00074     return button;
00075 }
```

The documentation for this class was generated from the following files:

- LeapPaint/LPCCControlButtonVariableSize.h
- LeapPaint/LPCCControlButtonVariableSize.m

4.21 LPLine Class Reference

Inheritance diagram for LPLine:



Properties

- NSMutableArray * **points**
- float **width**

4.21.1 Detailed Description

Definition at line 11 of file LPLine.h.

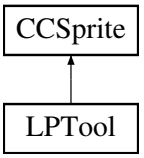
The documentation for this class was generated from the following file:

- LeapPaint/LPLine.h

4.22 LPTool Class Reference

`#import <LPTool.h>`

Inheritance diagram for LPTool:



Properties

- NSString * toolID
- BOOL updated

4.22.1 Detailed Description

Extends CCSprite object with two properties for tracking sprites with pointable objects

Definition at line 16 of file LPTool.h.

4.22.2 Property Documentation

4.22.2.1 - (NSString*) toolID [read], [write], [nonatomic], [strong]

toolID is the ID number assigned by the LeapMotion SDK

Definition at line 18 of file LPTool.h.

4.22.2.2 - (BOOL) updated [read], [write], [nonatomic], [assign]

updated is if the sprite has been updated in that frame.

Definition at line 19 of file LPTool.h.

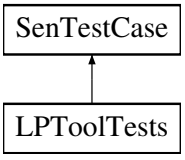
The documentation for this class was generated from the following file:

- LeapPaint/LPTool.h

4.23 LPToolTests Class Reference

`#import <LPToolTests.h>`

Inheritance diagram for LPToolTests:



Protected Attributes

- NSString * testName

4.23.1 Detailed Description

Tests the GameSettings object

Definition at line 13 of file LPToolTests.h.

4.23.2 Member Data Documentation

4.23.2.1 - (NSString*) testName [protected]

name of the test

Definition at line 14 of file LPToolTests.h.

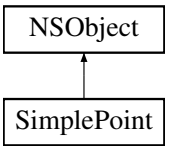
The documentation for this class was generated from the following file:

- LeapPaintTests/LPToolTests.h

4.24 SimplePoint Class Reference

`#import <SimplePoint.h>`

Inheritance diagram for SimplePoint:



Instance Methods

- (id) - initWithPosition:
- (id) - initWithX:withY:
- (id) - initWithPosition:withZ:
- (id) - initWithX:withY:withZ:
- (CGPoint) - point

Properties

- float x
- float y
- float z
- BOOL is3d

4.24.1 Detailed Description

2D or 3D space coordinate for temporarily maniulapting points

Definition at line 18 of file SimplePoint.h.

4.24.2 Method Documentation

4.24.2.1 - (id) initWithPosition: (CGPoint) *p*

Init constructor with existing point to create a 2d Point

Parameters

p	an point (x,y)

Returns

object instance

init 2d point with CGPoint

Definition at line 18 of file SimplePoint.mm.

```
00018                       :(CGPoint)p{
00019     if (self = [super init]) {
00020
00021         self.x = p.x;
00022         self.y = p.y;
00023         self.z = 0.0f;
00024         self.is3d = false;
00025
00026     }
00027     return self;
00028 }
```

4.24.2.2 - (id) initWithPosition: (CGPoint) *p* withZ:(float) *zVal*

Init constructor with existing point to create a 3d Point

Parameters

p	a point (x,y)
zVal	coordinateValue

Returns

object instance

Init 3d point with CGPoint and z Value

Definition at line 46 of file SimplePoint.mm.

```
00046                       :(CGPoint)p withZ:(float)zVal{
00047     if (self = [super init]) {
00048
00049         self.x = p.x;
00050         self.y = p.y;
00051         self.z = zVal;
00052         self.is3d = true;
00053
00054     }
00055     return self;
00056 }
```

4.24.2.3 - (id) initWithX: (float) *xVal* withY:(float) *yVal*

Init constructor with existing point to create a 2d Point

Parameters

xVal	coordinate value
yVal	coordinate value

Returns

object instance

Init 2d Point with 2 separate values

Definition at line 31 of file SimplePoint.mm.

```
00031                 :(float)xVal withY:(float)yVal{
00032
00033     if (self = [super init]) {
00034
00035         self.x = xVal;
00036         self.y = yVal;
00037         self.z = 0.0f;
00038         self.is3d = false;
00039
00040     }
00041     return self;
00042 }
```

4.24.2.4 - (id) initWithX: (float) *xVal* withY:(float) *yVal* withZ:(float) *zVal*

Init constructor with existing point to create a 2d Point

Parameters

xVal	coordinate value
yVal	coordinate value
zval	coordinate value

Returns

object instance

Init 3d Point with 3 separate values

Definition at line 60 of file SimplePoint.mm.

```
00060                 :(float)xVal withY:(float)yVal withZ:(float)zVal{
00061
00062     if (self = [super init]) {
00063
00064         self.x = xVal;
00065         self.y = yVal;
00066         self.z = zVal;
00067         self.is3d = true;
00068
00069     }
00070     return self;
00071 }
```

4.24.2.5 - (CGPoint) point

Returns point based on x and y

Returns

CGPoint

Return the CGPoint type from the object

Definition at line 74 of file SimplePoint.mm.

```
00074                 {
00075     return CGPointMake(self.x, self.y);
00076 }
```

4.24.3 Property Documentation

4.24.3.1 - (BOOL) is3d [read], [write], [nonatomic], [assign]

is3d is 2d or 3d point type

Definition at line 26 of file SimplePoint.h.

4.24.3.2 - (float) x [read], [write], [nonatomic], [assign]

x coordinate

Definition at line 23 of file SimplePoint.h.

Referenced by point.

4.24.3.3 - (float) y [read], [write], [nonatomic], [assign]

y coordinate

Definition at line 24 of file SimplePoint.h.

Referenced by point.

4.24.3.4 - (float) z [read], [write], [nonatomic], [assign]

z coordinate

Definition at line 25 of file SimplePoint.h.

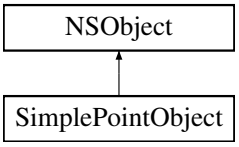
The documentation for this class was generated from the following files:

- LeapPaint/SimplePoint.h
- LeapPaint/SimplePoint.mm

4.25 SimplePointObject Class Reference

`#import <SimplePointObject.h>`

Inheritance diagram for SimplePointObject:



Instance Methods

- (id) - **initWithPosition:**
- (id) - **initWithX:withY:**

Properties

- CGPoint point

4.25.1 Detailed Description

2D space coordinate for temporarily maniulapting points

Definition at line 16 of file SimplePointObject.h.

4.25.2 Property Documentation

4.25.2.1 - (CGPoint) point [read], [write], [nonatomic], [assign]

point is the X and Y coordinates

Definition at line 21 of file SimplePointObject.h.

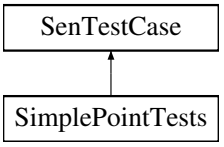
The documentation for this class was generated from the following files:

- LeapPaint/SimplePointObject.h
- LeapPaint/SimplePointObject.m

4.26 SimplePointTests Class Reference

`#import <SimplePointTests.h>`

Inheritance diagram for SimplePointTests:



Protected Attributes

- NSString * testName
- SimplePoint * twoValuePoint
- SimplePoint * threeValuePoint

4.26.1 Detailed Description

Tests the SimplePoint object

Definition at line 16 of file SimplePointTests.h.

4.26.2 Member Data Documentation

4.26.2.1 - (NSString*) testName [protected]

name of the test

Definition at line 18 of file SimplePointTests.h.

4.26.2.2 - (SimplePoint*) threeValuePoint [protected]

three coordinate point (x,y,z)

Definition at line 20 of file SimplePointTests.h.

4.26.2.3 - (SimplePoint*) twoValuePoint [protected]

two coordinate point (x,y)

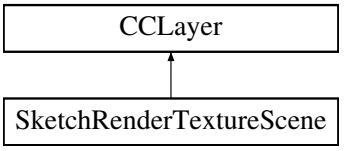
Definition at line 19 of file SimplePointTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/SimplePointTests.h

4.27 SketchRenderTextureScene Class Reference

Inheritance diagram for SketchRenderTextureScene:



Instance Methods

- (void) - **beginDraw:withZ:**
- (void) - **updateDraw:withZ:**
- (void) - **endDraw:**
- (void) - **changeColor:**
- (void) - **changeBrush:**
- (void) - **changeScale:**
- (void) - **changeOpacity:**
- (void) - **erasingMode:**
- (void) - **clearDrawing**

Protected Attributes

- CCSprite * **brush**
- NSMutableArray * **touches**
- ccColor3B **lastColor**
- ccColor3B **previousColor**
- NSString * **lastBrush**
- float **lastScale**
- bool **eraseMode**

Properties

- float **opacity**

4.27.1 Detailed Description

Definition at line 12 of file SketchRenderTextureScene.h.

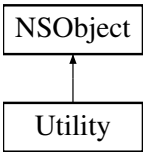
The documentation for this class was generated from the following files:

- LeapPaint/SketchRenderTextureScene.h
- LeapPaint/SketchRenderTextureScene.mm

4.28 Utility Class Reference

`#import <Utility.h>`

Inheritance diagram for Utility:



Class Methods

- (int) + getRandomNumberBetween:to:
- (int) + getRandomUniformNumberUnder:
- (int) + getRandomNumberUnder:

4.28.1 Detailed Description

Utility class provides common usage function throughout the application.

Definition at line 17 of file Utility.h.

4.28.2 Method Documentation

4.28.2.1 + (int) getRandomNumberBetween: (int) *from* to:(int) *to*

Generates a random number between two designated integers

Parameters

from	is the bottom of the range
to	is the top of the range

Returns

a random number between the from and to parameters

returns random number within a range with defined upper and lower bounds

Definition at line 14 of file Utility.m.

```
00014                               :(int)from to:(int)to {
```

```
00015
00016     //Check that one isn't greater than the other
00017     //if so, flip them
00018
00019     return (int)from + arc4random() % (to-from+1);
00020 }
```

4.28.2.2 + (int) getRandomNumberUnder: (int) *to*

Generates a random number between 0 designated integer

Parameters

to	is the top of the range

Returns

a random number between 0 and to parameters

Returns a random number from 0 to an upper bound

Definition at line 23 of file Utility.m.

```
00023                           :(int)to{
00024     return (arc4random() % to);
00025 }
```

4.28.2.3 + (int) getRandomUniformNumberUnder: (int) *to*

Generates a random number between 0 designated integer

Parameters

to	is the top of the range

Returns

a random number between 0 and to parameters

Returns a Uniform Random Number from 0 to an upper bound

Definition at line 29 of file Utility.m.

```
00029                                   :(int)to{
00030     //Check if uniform available
00031     if (arc4random_uniform != NULL)
00032         return  arc4random_uniform (to);
00033     else
00034         return  (arc4random() % to);
00035 }
```

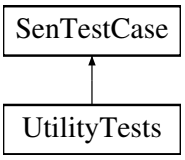
The documentation for this class was generated from the following files:

- LeapPaint/Utility.h
- LeapPaint/Utility.m

4.29 UtilityTests Class Reference

```
#import <UtilityTests.h>
```

Inheritance diagram for UtilityTests:



Protected Attributes

- NSString * testName

4.29.1 Detailed Description

Tests the GameSettings object

Definition at line 15 of file UtilityTests.h.

4.29.2 Member Data Documentation

4.29.2.1 - (NSString*) testName [protected]

name of the test

Definition at line 16 of file UtilityTests.h.

The documentation for this class was generated from the following file:

- LeapPaintTests/UtilityTests.h

Index

Appendix B

Specification

/importspecification.tex

Bibliography

[1] David Pierce. A look inside leap motion, the 3d gesture control that's like kinect on steroids, June 26 2012. URL http://www.theverge.com/2012/6/26/3118592/leap-motion-gesture-controls.

[2] Apple Computer Inc. Mac OS and iOS API, 2013. URL http://developer.apple.com.

[3] Leap Motion Inc. Leapmotion, April 2013. URL http://www.leapMotion.com.

[4] Stacey D. Scott, Regan L. Mandryk, and Kori Inkpen. Understanding children's collaborative interactions in shared environments. *J. Comp. Assisted Learning*, 19(2): 220–228, 2003. URL http://dx.doi.org/10.1046/j.0266-4909.2003.00022.x.

[5] Vanessa Colella, Richard Borovoy, and Mitchel Resnick. Participatory simulations: Using computational objects to learn about dynamic systems. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, 1998.

[6] Allison Druin. Cooperative inquiry: developing new technologies for children with children. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, pages 592–599, New York, NY, USA, 1999. ACM. ISBN 0-201-48559-1. doi: 10.1145/302979.303166. URL http://doi.acm.org/10.1145/302979.303166.

[7] Allison Druin. The role of children in the design of new technology. *Behaviour and Information Technology*, 21:1–25, 2002.

[8] Shari Lawrence Pfleeger and Joanne M. Atlee. *Software Engineering: Theory and Practice (4th Edition)*. Prentice Hall, 2009. ISBN 0136061699. URL

http://www.amazon.com/Software-Engineering-Theory-Practice-Edition/dp/0136061699%3FSubscriptionId%3D0JYN1NVW651KCA56C102%26tag%3Dtechkie-20%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0136061699.

[9] Nayan B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010. ISSN 0163-5948. doi: 10.1145/1764810.1764814. URL http://doi.acm.org/10.1145/1764810.1764814.

[10] Unity Technologies. Unity SDK, 2013. URL http://unity3d.com/.

[11] Community Project. Cocos2d game engine framework. URL http://www.cocos2d-iphone.org/.

[12] Ray Wenderlich. Breakout. URL http://www.raywenderlich.com/28604/how-to-create-a-breakout-game-with-box2d-and-cocos2d-2-x-tutorial-part-1.

[13] Yannick Loriot. Cccontrolextension. URL https://github.com/YannickL/CCControlExtension.

[14] Nick Gillian. Gesture recognition toolkit. URL https://code.google.com/p/gesture-recognition-toolkit/.