Senior Project CIS 4911-U01

Multi-Touch and Mid-Air Framework

Visualizer

Design Document

Team Members:

Richard A. Lopez

April 30, 2015

Mentor:

Francisco Ortega

Professor:

Masoud SadjadiAbstract

This document outlines the design of the Touch and Mid-Air framework for the use of developers who want to create applications that use any type of input devices, focusing mainly on touch screens and mid-air inputs. Also covered on this document is the plan that we are to follow when developing this API, the design of the architecture, systems, subsystems and modules. Also covered is the design of persistent data and the implementation of the communications between subsystems.

We explore the design of the TAM Visualizer system, the architectural patters utilized to decompose the system into subsystems. We describe how the subsystems are related to the requirements, how the hardware and software maps the system and subsystems. Lastly we go over the detailed design, the static and dynamic models, and the detailed code specification.

# Table of Contents

[Table of Contents 3](#_Toc418100522)

[1. Introduction 4](#_Toc418100523)

[1.1. Problem definition 4](#_Toc418100524)

[1.2. Design Methodology 4](#_Toc418100525)

[1.3. Definitions, acronyms, and abbreviations. 5](#_Toc418100526)

[1.4. Overview of Document 5](#_Toc418100527)

[2. System Design 6](#_Toc418100528)

[2.1. Overview 6](#_Toc418100529)

[2.2. Subsystem Decomposition 7](#_Toc418100530)

[2.3. Hardware and Software Mapping 8](#_Toc418100531)

[2.4. Persistent Data Management 9](#_Toc418100532)

[2.5. Security/Privacy 9](#_Toc418100533)

[3. Detailed Design 10](#_Toc418100534)

[3.1. Overview 10](#_Toc418100535)

[3.2. Static model 10](#_Toc418100536)

[3.3. Dynamic model 11](#_Toc418100537)

[3.4. Code Specification 12](#_Toc418100538)

[4. Glossary 13](#_Toc418100539)

[5. Appendix 15](#_Toc418100540)

[5.1. Appendix A - Use case diagram for use cases being implemented 15](#_Toc418100541)

[5.2. Appendix B - Use cases being implemented 16](#_Toc418100542)

[5.3. Appendix C - Documented class interfaces 30](#_Toc418100543)

[5.4. Appendix D - Diary of meeting and tasks. 31](#_Toc418100544)

[6. References 38](#_Toc418100545)

# Introduction

Input devices have advanced at an outstanding rate in the last few decades. The introduction of the mouse was one of the factors that brought the personal computer out of the technical user niche and into the regular consumer. Now touch input is in almost every cellphone and readily available to the general public. Newer input forms have been developed since such as multi touch screen, 3D mouse, midair tracking, and others. Up until now developers have had to code their respective games and applications for the individual input devices which make it a time consuming task and it limits the utilization of more than one input device in use at a time.

The Touch Air Motion Framework is trying to create a more encompassing input device library such that developers can simply choose their input devices and code their apps without having to deal with the individuality of each device.

## Problem definition

Today’s market is flooded with different input devices and it has become a common problem for app and game developers to code for the individual input devices aside that to code for their own application or game. Most of the time the plugins and internal software of the input device is available for the developers to search through, learn and be able to apply the necessary characteristics that the developers needs the input devise to express. In the worst cases the information is not open source or there is not enough information around hindering the use of these input devices.

It has become cumbersome for some software developers to decide which input devices they can use out of the list that they really want to use. A framework that contains most of the input devices in the market will make is so that more developers can code for more out of the box applications, making the computer science field a more user friendly environment. A centralized input device library can make it so that new generations of developers and students can code for the low cost input devices and create the next generation of human computer interactions.

## Design Methodology

The software development methodology used for this semester is Agile Software Development. The work was divided into five Scrums which is an Agile method that focuses on the organizational aspects of the software engineering process. It is currently one of the most popular Agile methods. (Sen. Proj. Class).

Creating a design from user stories was simple since the Agile process provided a small amount of requirements that usually were related to each other. Often the user stories collected during scrums were paired into subsystems due to the cohesion between them. Knowing what the system was supposed to look like provided a good basis for deciding the architectural patterns early on as well as the abstract subsystems. To represent the design multiple UML models were used, package diagrams and system diagrams were constantly re imagined as new functionalities were added and old functionalities were improved. The deployment diagrams helped to have an outside view of the system although in this case the system is but a small portion of a greater system therefore it will differ from the future deployments.

## Definitions, acronyms, and abbreviations.

* API: Application Program Interface
* GUI: Graphical User Interface
* HCI: Human Computer Interaction
* Int: the local data structure that represents an integer value
* Struct: a combination of different types of data all contained under the struct object
* TAM: Touch and Mid-Air Framework
* WIMP: Windows, Icons, Menus, Pointer
* UML Unified Modeling Language

## Overview of Document

This document will go over the complete design of the TAM Visualizer 1.0. In chapter two we look at the system design, the sub system decomposition the ways the system will be deployed and the format of the persistent date generated by the user. In chapter three the system design is explained in detail, the class diagrams and dynamic models for the TAM Visualizer, and the code specification. Chapter for is the glossary of all the technical terms utilized in this document, chapter six contains the appendix with all the supplementary images and tables, and chapter six contains the references used in writing this document.

# System Design

In this chapter we will look at the system decomposition of the TAM Visualizer version 1.0. The system is broken into groups of similar classes, each of this classes has a role to play in the functionalities of the visualizer, the groupings are generated to increate cohesion and to decrease coupling. Further on, the subsystems will be decomposed into individual classes and methods which are the implementations of the necessary functions or the interactions with the user or with other subsystems.

## Overview

Since the TAM framework is designed to be a modular framework, we decided that the TAM Visualizer be a standalone subsystem. For the purposes of this project the visualizer adapted some of the functions that the other subsystems will have in the final version of the TAM framework. The architectural patters implemented for the TAM framework are the Service Oriented Architecture and a modified Model View Controller pattern. In fig. 2.1.1 we can see the design for the complete TAM Framework, the top level or View on our design is the TAM visualizer, the middle layer is the Device, and the bottom layer is the Connection. The bottom layer represents the connection between the input device and the operating system, this connection is the one that we must subscribe to receive any information on any event triggered.

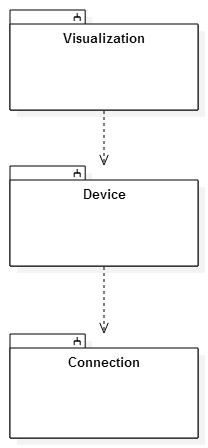


Fig. 2.1.1

## Subsystem Decomposition

The TAM framework consists of tree subsystems, the Visualization subsystem which is the subsystem implemented in this version. The Device subsystem which will deal with the input devices, and the Connection subsystem which will deal with the different operating systems.

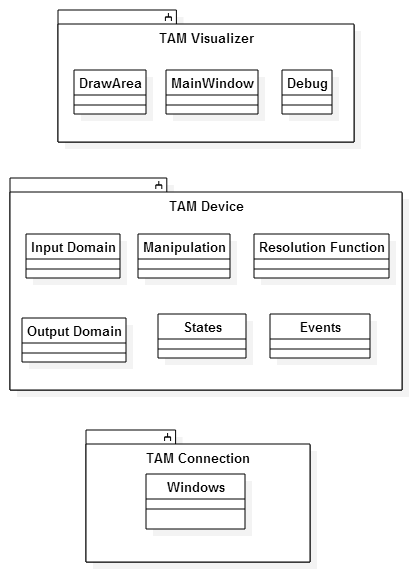


Fig. 2.2.1

As previously mentioned, the subsystem implemented in version 1.0 is the Visualization subsystem, this subsystem deals with the 2D representation of the inputs, outputs, and the eventual verification of the functions mapping input to output. This subsystem implements all of the requirements that were selected to be implemented this semester. Some of the requirements were implemented as part of the Visualization subsystem while they should be part of different subsystems, this was due to the necessity to test the visualization subsystem while the rest of the TAM framework is yet to be implemented. The touch, replay, save, resize, debug, clear screen and open requirements are native to the Visualization subsystem, the rests will be implemented in the other subsystems but for now are part of the visualization subsystem.

## Hardware and Software Mapping

TAM Visualizer is designed to work on any Windows 8 or greater touch screen capable computer. There is no need for a client server deployment since it is a self-contained application. In future versions TAM will have more than one device functionality therefore later on those devices will have to be added to the deployment diagram in fig. 2.3.1.

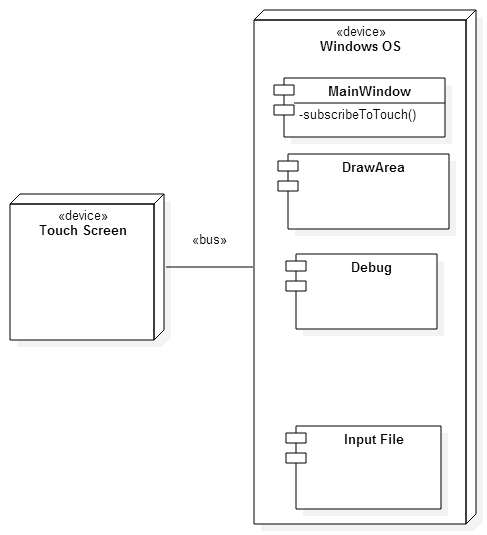


Fig. 2.3.1

As we can see, the device which in our case is the touch screen communicates to the computer (OS) via a bus, be it a serial bus or a parallel one depending on the device itself. The entire visualizer is deployed right onto of the operating system, with the tree classes that make up the visualizer, also included is the input file which is the file structure used to save and play back input data that the user wants to utilize. The Main Window class is the responsible to subscribe to the windows operating system in order to receive the touch inputs.

## Persistent Data Management

Since we are dealing with input devices, there is no need for any complicated data storage. We will only save four pieces of information in the case of touch screens. The X coordinate, the Y coordinate, the ID of the finger to differentiate the number of fingers touching at any time, and finally a timestamp to provide the developer with another way to help the debug process. We used a *struct* to represent the data into tree *int* fields and a *time\_t* field, to facilitate the temporary storage of the data we created a list of these *structs* also called *touchPoints* therefore we have a list of *touchPoints* that represents a touch story.

## Security/Privacy

The TAM visualizer and the TAM framework do not utilize any form of registered user, no private data is kept and the framework is aimed towards developers as well as the open source community, therefore there is no privacy or security implementation.

# Detailed Design

In this chapter we take a look at the lower level implementation of the design of the TAM visualizer. We will first take a look at the general behaviors of the visualizer subsystem, then we will look at the static model of the subsystem which provides a detailed description with the class diagrams. We look at the state machine diagram of the visualizer subsystem along with the algorithms used to design the subsystem. Lastly we look at the classes within the TAM visualizer.

## Overview

The TAM Visualizer subsystem is composed of three classes. These classes contain all of the necessary factions to fulfill the requirements by the product owner. This subsystem subscribes to the Windows operating system to receive event notifications of when a touch even happens. The 2D drawing on the screen happens as the events are received. The functions for saving, opening, replaying, debugging and resizing are all contained in the DrawArea class and the triggers for all those functions are in the MainWindow Class.

## Static model

The structure of the TAM Visualizer subsystem has three classes, the Main window, this being a view object to which the DrawArea our second class is subscribed to, following a Façade pattern, as well as a mediator pattern. The MainWindow controls all of the observed GUI as well as the draw area events once the message from the windows operating system reached the MainWindow. The messaging pattern is applied to the messages between the operating system and the MainWindow, these messages are in the form of Events which is the general class, and the specific event is a Touch Event. The DrawArea is the controller object being the one that actually computes the functions.

Finally the Debug class is the one responsible for creating a debug monitor and establishing all of the observer functions that take any kind messages to the standard output and re routs them to this debug.

As we can see on fig. 3.2.1 the three classes are related as follows; Both DrawArea and Debug are dependent on the MainWindow class which acts as a façade and mediator between the user and the application. DrawArea and MainWindow are directly associated with the debug monitor since any print, debug, or error statement is immediately rerouted to the Debug class.

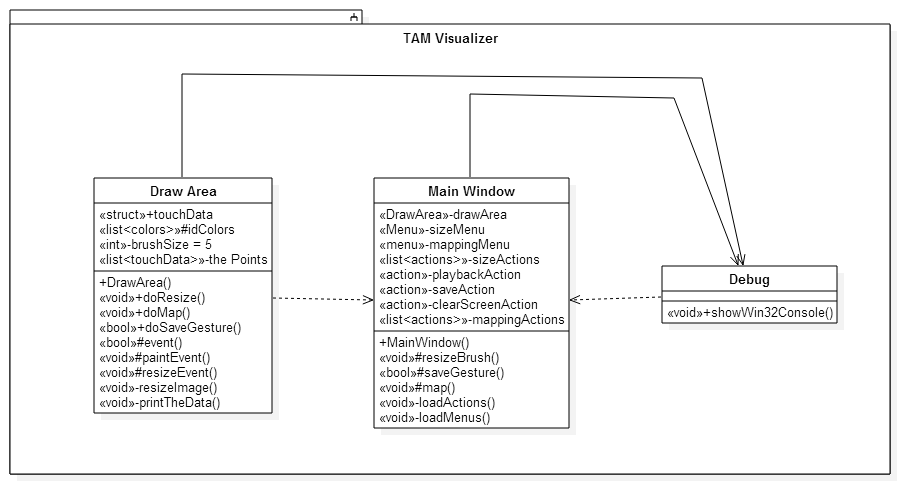


Fig. 3.2.1

## Dynamic model

The main control object in the Visualization subsystem is the event function, this function is triggered when the operating system sends the message to the MainWindow that a touch event has happened; the MainWindow then relays that message to the DrawArea, in here we deal with the message within the tree possible states of a touch event, the touching, moving and lifted events which as their names say they are triggered whenever the user is touching, moving or has lifted a finger or fingers.

As we can see in fig. 3.3.1 the statechart for the event functions starts as the system gets started, the default or no touching state is maintained until a touching signal is received this leads to a touching state. In the same way this touching state is maintained until a moving state is triggered, if a lift even tis triggered instead we return to the default state; if instead we receive a moving event we go to the moving state, here whenever the user stops moving their finger (s) the state goes into a decision, has the user lifted the finger (s) or simply stopped moving, depending on the decision we either go to the default state or to the touching state respectively.

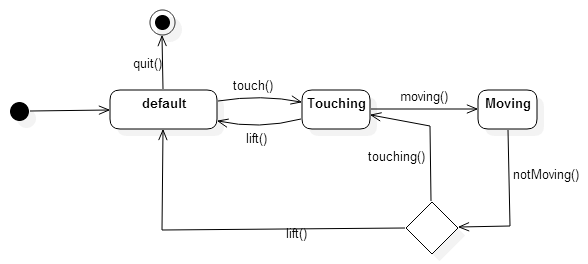


Fig. 3.3.1

## Code Specification

The MainWindow class contains three methods that are public, and two that are private, and eight private attributes. First the Tree public methods resizeBrush(), saveGesture() and map() act as message passers, they are triggered when the respective buttons are clicked, the information regarding the size selected and the map selected are stripped from the message and passed along towards their twin functions in the DrawArea class.

The two private functions in the MainWindow class are loadActions() and loadMenus(), these two functions start the button objects and the lists inside the brushsize and map menus. The attributes of the MainWindow class are a DrawArea object, the menus for size and map, and the actions for size, clearscreen, save, playback, and map, they are all actions objects except size and map which are lists of actions objects. See code in Appendix C

# Glossary

* **Actor:** External entity that interacts with the system
* **Architectural pattern:** Expresses a fundamental structural organization schema for software systems.
* **Class diagram:** describes the system in terms of classes, attributes (data items, instance variables), operations (member functions, methods) and their relationships. Class diagrams are used to represent object models during development conditions for a specified period of time
* **Coupling:** is the strength of dependencies between subsystems.
* **Cohesion:** is the strength of dependencies between within a subsystem.
* **Design pattern:** Provides a schema for refining the subsystems or components of a software system, or relations between them.
* **Frequency:** Rating assigned to the piece of functionality that represents the level of invocations a particular piece of functionality will be subjected to.
* **Functional requirements:** An area of functionality the system must have.
* **Interface:** An abstraction that an entity provides of itself to the outside. Separates the methods of external communication from internal operation, and allows it to be internally modified without affecting the way outside entities interact with it, as well as provide multiple abstractions of itself.
* **Invariants:** is a predicate that is always true for all the instances of a class.
* **Model:** An abstraction of a system aimed at simplifying the reasoning about the system by omitting irrelevant details.
* **Nonfunctional requirements:** a constraint on the system
* **Pattern:** Addresses a recurring design problem that arises in specific design situations, and presents a solution to it
* **Persistent data:** Data that outlives a single execution of the system.
* **Reliability:** Ability of a system or component to perform its required functions under state
* **Risk:** An area of uncertainty that can lead to a deviation in the project plan, including failure of the project.
* **Sequence diagram:** An interaction diagram that emphasizes the time-ordering of messages.
* **Specification:** Describes the requested behavior of the software system.
* **Subsystem decomposition:** The division of the system into subsystems. Each subsystem is described in terms of its services during system design and it’s API during object design.
* **Subsystem:** Is a well-defined software component that provides a number of services to other subsystems, like manages persistent data, the interaction with the user, and the communication with other subsystems over the network
* **UML (Unified Modeling Language)** is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system. (Booch et al. 1999)
* **Usability:** Ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
* **Use case model:** Represents the functionality of the system in terms of a sequence of interactions between an actor and the system.
* **Use case:** General sequence of events that describe all possible actions between actor and the system for a given piece of functionality.

# Appendix

## Appendix A - Use case diagram for use cases being implemented

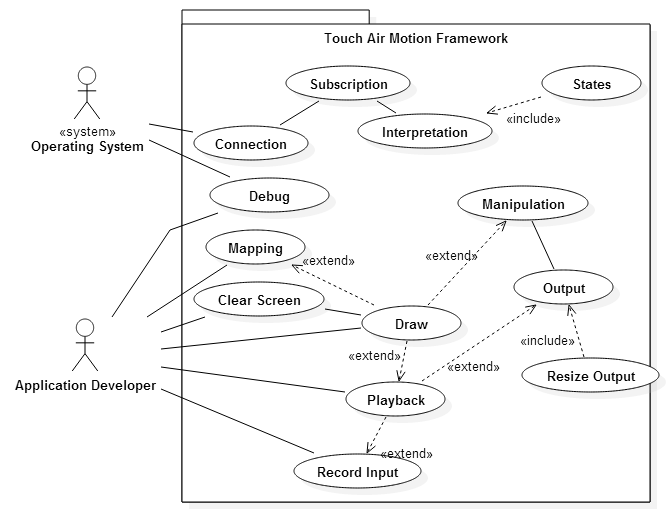


Fig.5.1.1

## Appendix B - Use cases being implemented

**Use Case ID:** TAMUC01 – Subscription

**Use Case Level:** High-Level

**Details:**

* **Actors:** Operating System, Developer
* **Pre-conditions:**
  1. The system is capable of multi touch input (Touch Screen)
* **Description:**
  1. Use case begins when the application is started.
  2. The system creates a listener service that subscribes to the touch events in the draw area.
  3. The use case ends when the system receives the subscription ticket.
* **Post-conditions:**
  1. The system is ready to start receiving touch events in the draw area.
* **Alternative Courses of Action:**

n/a

* **Exceptions:**

The system cannot connect to the touch event service.

* **Related Use Cases:**
  1. TAMUC02
  2. TAMUC03
  3. TAMUC04
  4. TAMUC05
  5. TAMUC07

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1-10’000 touch events per second.

**Criticality:** High. The Framework cannot be tested if the touch input cannot be received.

**Risk:** Medium. The subscription service depends on the Windows operating system.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No Human User
* Reliability: The system shall recognize 10 inputs maximum and up to 10’000 different touch events per second.
* Performance: The input position should be within ±0.5 mm of the touch screen position.
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC02 – Draw Action

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
* **Description:**
  1. Use case begins when the user touches the screen with 1-10 fingers inside the “drawing area”
  2. The system shall obtain the X and Y coordinates and use the mapping function selected (TAMUC04-Mapping)
  3. The system shall display a point the size which has been selected (TAMUC09– Resize) at the position designated by the Mapping function.
  4. The user continues the drawing without lifting the finger(s).
  5. The System continues to trace all of the points while simultaneously saving the points (TAMUC07-Record)
  6. The use case ends whenever the user lifts their finger(s) from the screen.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
* **Alternative Courses of Action:**
  1. The user after lifting finger(s) touches the draw area again and continues drawing.
* **Exceptions:**

The user asks another person for them to draw at the same time as him/her; making it more than 10 inputs (the system will only draw original 10).

* **Related Use Cases:**
  1. TAMUC05
  2. TAMUC07
  3. TAMUC09

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1-10’000 touch events per second.

**Criticality:** High. The Framework cannot be tested if 2D representation is not available.

**Risk:** Low. The implementation can only fail by not drawing an input.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall recognize 10 inputs maximum and up to 10’000 different touch events per second.
* Performance: The displayed points should appear in less than ¼ of a second
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC03 – States

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
* **Description:**
  1. Use case begins when the user touches the screen with 1-10 fingers inside the “drawing area”
  2. The Operating system flags the touch even and sends the touch signal and information to the application.
  3. The application receives the touch event “Touching” and labels it at such.
  4. The user moves their finger(s) across the draw area.
  5. The Operating system flags the touch event and sends the touch signal information to the application.
  6. The application receives the touch event “Moving” and labels it as such.
  7. The user lifts their finger(s) from the draw area.
  8. The Operating system flags the touch event and sends the touch signal information to the application.
  9. The application receives the touch event “Lifted” and labels it at such.
  10. The use case ends after the event “Lifted” is received.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
* **Alternative Courses of Action:**
  1. The user after lifting finger(s) touches the draw area again and continues drawing.
* **Exceptions:**

The user asks another person for them to draw at the same time as him/her; making it more than 10 inputs (the system will only draw original 10).

* **Related Use Cases:**
  1. TAMUC01
  2. TAMUC02
  3. TAMUC04
  4. TAMUC05
  5. TAMUC07

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1-10’000 touch events per second.

**Criticality:** High. The Framework cannot be tested if the touch events cannot be differentiated.

**Risk:** Low. The application does not fail if a touch event is not received.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall recognize 10 inputs maximum and up to 10’000 different touch events per second.
* Performance: The touch events should be sent and received in less than 1/10000 of a second.
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC04 – Mapping

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
  3. TAMUC01 is in step 2 of the description.
* **Description:**
  1. Use case begins when the user selects the mapping menu.
  2. The system displays a selection of all the available mapping functions.
  3. The user selects the desired mapping function.
  4. The system then keeps the selected mapping in memory whenever an input is obtained.
  5. The system shall compute the output coordinates based on the selected mapping function (use 1 to 1 if no function selected).
  6. The System traces all of the points while simultaneously saving the points (TAMUC07-Record)
  7. The use case ends when the last input is mapped to its output, displayed and saved.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
  2. The latest selected mapping function remains active
* **Alternative Courses of Action:**
  1. The user after lifting finger(s) touches the draw area again and continues drawing.
  2. The user selects a different mapping function.
* **Exceptions:**

N/A

* **Related Use Cases:**
  1. TAMUC02

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1-10 changes in the mapping function for every input test.

**Criticality:** Medium. Mapping functions can be used or 1 to 1 as crude testing implements.

**Risk:** Low. The default 1 to 1 mapping creates a safe fall.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall map the output to two ½ a pixel from the received input.
* Performance: The displayed points should appear in less than ¼ of a second
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time. The mapping function shall not exceed the edges of the drawing area.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC05 – Input Recording

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
* **Description:**
  1. Use case begins when the user touches the screen.
  2. The system stores the X and Y coordinate, the ID of the finger, and a timestamp of the touch event.
  3. The user continues to touch the screen and move their fingers around the draw area.
  4. The system records each touch even and adds it at the end of a list.
  5. The use case ends when the user lifts all fingers from the draw screen and selects either replay, save, or closes the application.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
  2. The system continues to wait for the user to end the current test.
* **Alternative Courses of Action:**
  1. The user after lifting finger(s) touches the draw area again and continues drawing.
* **Exceptions:**

The maximum number of touch points is reached.

* **Related Use Cases:**
  1. TAMUC02

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1-10,000 touch events per second.

**Criticality:** High, It is necessary for the input to be saved so that the replay can happen and the visualizer performs its task.

**Risk:** Medium. The size of the input data may slow down the system while displaying the dots or while playing the recorded data back.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall add the input to the list in less than 1/100 of a second.
* Performance: There list should be able to handle 1 Million touch points without noticeable system slowness.
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC07 – Saving Input to a file

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
  3. The user has drawn on the draw area
* **Description:**
  1. Use case begins when the user selects the “Save” option.
  2. The system prompts the user for a filename and a file type. (CSV is default).
  3. The user selects file path and enters the name of the file (UNTITLED is default).
  4. The user selects the save button.
  5. The system creates the file with the name provided and saves each touch point in the order that the touch action happened.
  6. The use case ends when the system writes the last touch point in the file and closes the file.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
  2. The file is ready to be read from.
* **Alternative Courses of Action:**
  1. The user cancels the save.
* **Exceptions:**

There might not be any touch points (empty file).

* **Related Use Cases:**
  1. TAMUC02
  2. TAMUC05

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average 5 saves for every touch test.

**Criticality:** Medium. The ability to save is necessary for replays of older actions, an immediate replay is more important as implemented in TAMUC06 Playback.

**Risk:** Low. If the saving action fails it only cancels the save then the user might save again.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall write the input list to the file for every touch event.
* Performance: There file should be able to handle 1 Million touch points without noticeable system slowness.
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time. The file needs to be saved in a Comma Separated Value file to be used by TAMUC08
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC08 – Playback from a File

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
  3. A file with a list of touch points has been created
* **Description:**
  1. Use case begins when the user selects the “Open” option.
  2. The system prompts the user to select the path of the file and the file to be replayed.
  3. The user navigates to the file location, selects the file and selects the “Open” option
  4. The system opens the file, and reads from it while saving the touch points to the replay list.
  5. The system replays the touch points per TAMUC06 Replay.
  6. The use case ends when the last touch point on the file has been replayed.
* **Post-conditions:**
  1. The screen allows for another draw gesture.
  2. The system allows for replay of the file since it has been open once already.
* **Alternative Courses of Action:**
  1. The user might cancel the open action by not selecting a file and selecting the “Cancel Option”
* **Exceptions:**

N/A

* **Related Use Cases:**
  1. TAMUC02
  2. TAMUC06

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 5 opens per touch test.

**Criticality:** Medium, immediate replay is more necessary than opening files previously saved.

**Risk:** Medium. The size of the input data may slow down the system while displaying the dots or while playing the recorded data back.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall add the input to the list in less than 1/100 of a second.
* Performance: There list should be able to handle 1 Million touch points without noticeable system slowness.
* Supportability: The file type has to be a Comma Separated Value type.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

---------------------------------------------------------------------------------------------------------------------

**Use Case ID:** TAMUC06 – Playback

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
* **Description:**
  1. Use case begins when the developer touches the screen with 1-10 fingers
  2. The system shall store the coordinate data from the touch event.(TAMUC04)
  3. The developer stops touching the screen.
  4. The developer chooses the “Playback” button
  5. Use case ends when the system replays the touch events from the stored data
* **Post-conditions:**
  1. The next time a developer causes a touch event the last touch event to be replayed gets replaced by the newest touch event.
* **Alternative Courses of Action:**
  1. The developer may choose to clear the screen and create a new touch event without replaying the previous touch event
* **Exceptions:**

None.

* **Related Use Cases:**
  1. TAMUC04

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 2-5 playbacks every 10 minutes.

**Criticality:** High. The Framework cannot be tested if a playback is not possible.

**Risk:** Low. The system is independent of the playback until it becomes time to replay, therefore it is not necessary to integrate and can be a module loaded when needed.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The system shall replay the touch events up to ½ mm precision.
* Performance: The replay should load in no more than 1 second
* Supportability: The screens with multi-touch capabilities cannot exceed 10 fingers at the same time.
* Security: n/a

---------------------------------------------------------------------------------------------------------------------

**Use Case ID:** TAMUC09 – Resize

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen is cleared or its started as a new blank canvas
* **Description:**
  1. Use case begins when the user selects the “Size” option.
  2. The system displays a selection of all the available sizes that can be applied to the touch visualization.
  3. The user selects the size wanted.
  4. Use case ends when the system applies the selected size brush to any future touch event.
* **Post-conditions:**
  1. The next time a developer causes a touch event the displayed brush output will be the size selected by the user.
* **Alternative Courses of Action:**
  1. The developer may choose to select a different size after the first selection.
* **Exceptions:**

None.

* **Related Use Cases:**
  1. TAMUC02
  2. TAMUC04

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 5-10 resize events per minute.

**Criticality:** Medium. Resizing is a secondary need to be able to determine the precision of the touch mapping.

**Risk:** Low. The Resizing event changes the size of the drawing brush and no other aspects.

**--------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: The size selected is to be used every time, no failures are acceptable.
* Performance: The change in the size should be available immediately after selecting it.
* Supportability: Brush sizes 5 - 100.
* Security: n/a

**---------------------------------------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC010 – Clear Screen

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
  2. The screen has been drawn to
* **Description:**
  1. Use case begins when the user selects the “Clear” option.
  2. The system creates a new blank canvas the size of the draw area.
  3. Use case ends when the system replaces the old canvas with the new canvas and displays it to the user.
* **Post-conditions:**
  1. The draw area will be clear allowing for new touch events to happen.
* **Alternative Courses of Action:**
  1. The developer may choose to clear the screen more than once consecutively.
* **Exceptions:**

None.

* **Related Use Cases:**
  1. TAMUC02
  2. TAMUC04

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1 – 30 clear events per minute.

**Criticality:** High. For the touch replays and the touch events to be studied there is a need to have a clear canvas.

**Risk:** Low. The new blank canvas can be created in a secondary process to that of the touch actions.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: For every 1 million clear events the screen should clear 999,999 times.
* Performance: The screen shall appear blank in less than 1/1000 of a second
* Supportability: The new canvas should support screens of all sizes.
* Security: n/a

**---------------------------------------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------------------------------------**

**Use Case ID:** TAMUC011 – Debug

**Use Case Level:** High-Level

**Details:**

* **Actors:** Developer
* **Pre-conditions:**
  1. The Application has been launched
* **Description:**
  1. Use case begins when the application first launches.
  2. The system opens a window displaying a debug terminal.
  3. As the actor uses the program any message resulting form that will be displayed in this debug monitor.
  4. Use case ends when the user closes the application.
* **Post-conditions:**
  1. Closing the debug monitor causes the program to close.
* **Alternative Courses of Action:**
  1. The developer may choose to close the program but not the debug monitor.
* **Exceptions:**

None.

* **Related Use Cases:**

n/a

**---------------------------------------------------------------------------------------------------------------------**

**Decision Support**

**Frequency:** On average, 1 – 10,000 messages per minute.

**Criticality:** Medium. The debug monitor is a secondary instance to the main application.

**Risk:** Low. The debug has no connection to the application the messages only go one way from the application to the debug.

**---------------------------------------------------------------------------------------------------------------------**

**Constraints:**

* Usability: No previous Training Time
* Reliability: For every 1 million messages 1 missed message and 999,999 written.
* Performance: The information being displayed shall do so in less than 1/1000 of a second

Supportability: The debug monitor is a Windows application and shall only work in Windows Operating Systems.

## Appendix C - Documented class interfaces

MainWindow

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QtWidgets/QMainWindow>

#include "drawArea.h"

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow();

/\*slots for the communication between MainWindow and DrawArea\*/

protected:

public slots:

void resizeBrush();

bool saveGesture();

bool openGesture();

void map();

private:

DrawArea \*drawArea;

/\*Functions to itialize the Menus and Actions\*/

void loadActions();

void loadMenus();

/\*Menus for the GUI\*/

QMenu \*sizeMenu;

QMenu \*mapMenu;

/\*Actions or Buttons for the GUI\*/

QList<QAction \*> sizeActions;

QList<QAction \*> mapActions;

QAction \*playbackAction;

QAction \*saveAction;

QAction \*openAction;

QAction \*clearScreenAction;

};

#endif // MAINWINDOW\_H

DrawArea

#ifndef DRAWAREA\_H

#define DRAWAREA\_H

#include <cstdlib>

#include <qcolor.h>

#include <qimage.h>

#include <qpoint.h>

#include <qwidget.h>

#include "debug.h"

using namespace std;

class DrawArea : public QWidget

{

Q\_OBJECT

public:

DrawArea(QWidget \*parent = 0);

/\*The structure to hold the data of a touch input\*/

struct touchData

{

long long x;

long long y;

int id;

time\_t timeStamp;

};

/\*Changes the global brushSize attribute\*/

void doResizeBrush(int i);

/\*Changes the global mapping attribute\*/

void doMap(int map);

/\*Starts the save action by opening a save dialog native to the OS\*/

bool doSaveGesture(QString fileName);

/\*Starts the open actionn by opening an open dialog native to the OS\*/

bool doOpenGesture(QString fileName);

/\*Slots used for communication between DrawArea and MainWindow\*/

public slots:

void playback();

void clearScreen();

protected:

/\*The main function that gets triggered when a touch event happens\*/

bool event(QEvent \*event);

/\*A supplementary function that does the 2D painting\*/

void paintEvent(QPaintEvent \*event);

/\*A supplementary function for controlling window size\*/

void resizeEvent(QResizeEvent \*event);

/\* A list of color objects\*/

QList<QColor> idColors;

private:

/\*Global size of the brush\*/

int brushSize = 20;

/\*Global value for mapping function\*/

double mapping = 1;

/\*Global immage object to draw\*/

QImage image;

/\*A supplementary function for controlling window size\*/

void resizeImage(QImage \*image, const QSize &newSize);

/\*Global list of all the touchPoints\*/

QList<touchData> thePoints;

/\*Supplementary function to display the touch points in the debug monitor\*/

void printthedata();

};

#endif

Debug

#ifndef DEBUG\_H

#define DEBUG\_H

#include <QtWidgets/QMainWindow>

#include <qlabel.h>

class Debug : public QMainWindow

{

Q\_OBJECT

public:

Debug();

/\*Function to start the debug monitor, WINDOWS ONLY\*/

void showWin32Console();

protected:

private:

QLabel \*imageLabel = new QLabel;

};

#endif // DEBUG\_H

## Appendix D - Diary of meeting and tasks.

Meeting 1:

Date: 01/19/15

Participants: Anthony M Amador

Francisco Ortega

Richard A Lopez

Topics discussed:

* More in-depth explanation of the system by Francisco
* Detailed explanation on Francisco expectations on Anthony and Richard
* Background information obtained from Anthony and Richard
* Set up next meeting Tuesday January 20th

Meeting 2:

Date: 01/20/15

Participants: Anthony M Amador

Francisco Ortega

Richard A Lopez

Topics discussed:

* Assigned Scrum roles: Richard: Scrum Master

Anthony: Team Member

* Discussed possible use cases
* Set up next meeting Wednesday January 21st

Meeting 3:

Date: 01/21/15

Participants: Anthony M Amador

Francisco Ortega

Richard A Lopez

Topics discussed:

* Took down all the ideas that Francisco wanted for the system
* Molded the ideas into the format of user stories ”who, what, why”
* Collected all the user stories and placed the ones that seemed unfeasible at the moment
* Set up a face to face meeting to learn C++ for Monday January 26th

Meeting 4:

Date: 01/26/15

Participants: Anthony M Amador

Francisco Ortega

Richard A Lopez

Topics discussed:

* Francisco introduced C++
* Francisco introduced some code for the system
* Anthony and Richard set up reading from C++ book
* Set up next meeting for Wednesday January 28th

Meeting 5:

Date: 01/28/15

Participants: Anthony M Amador

Richard A Lopez

Topics discussed:

* Cleaned up user stories
* Decided on points to be given for all stories
* Decided on stories that could not be developed during this semester
* Split up development of documents and diagrams
* Set up net meeting for Saturday January 31st

Meeting 6:

Date: 01/31/15

Participants: Anthony M Amador

Francisco Ortega

Richard A Lopez

Topics discussed:

* Talked about the task that would be created for each story
* Decided on meeting time for sprint planning for Monday February 2nd
* Split up more work for presentation

Meeting 7:

Date: 02/02/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Talked about the task that would be created for each story
* Decided on the task to be implemented for sprint 1
* Talked about learning Qt
* Created the basic structure of the Git repository

Meeting 8:

Date: 02/13/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Reviewed the work done in sprint one
* Went over the user stories to refine them
* Sent a few user stories to the backlog
* Created a new user story replay
* Learned more C++ from Francisco
* Talked about the user stories for next sprint, all user stories will be carried over

Meeting 9:

Date: 02/27/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Reviewed the work done in sprint two
* Accepted most of the user stories for sprint two
* Choose the user stories for sprint three
* Clarified some questions on Qt and the touch interface
* Learned more C++ from Francisco

Meeting 10:

Date: 03/20/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Reviewed the work done in sprint three
* Accepted all user stories from sprint three
* Re-defined the user stories for the semester due to a teammate leaving
* Choose the user stories for sprint four
* Created new user stories to accommodate for previous defects
* Talked about the mapping and recognizer user stories

Meeting 11:

Date: 04/03/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Reviewed the work done in sprint four
* Accepted all user stories except saving to file from sprint four
* Choose the user stories for sprint five that encompasses all the needs for a visualizer in version one point zero.

Meeting 12:

Date: 04/17/15

Participants: Francisco Ortega

Richard A Lopez

Topics discussed:

* Reviewed the work done in sprint five
* Accepted the final user story
* Talked about the documents and the poster
* Went over the complete visualizer
* All done Now to work on the documents!!

# References

“Agile tutorial for the Senior Project Class”. Web. 26 Apr. 2015. <https://moodle.cis.fiu.edu/v2.8/pluginfile.php/45767/mod\_resource/content/2/Agile\_Tutorial.pdf>

Taylor II, Russell. "Overview." *VRPN*. Web. 26 Apr. 2015. <http://www.cs.unc.edu/Research/vrpn/>