DEVELOPER MANUAL

Version 4.0
April 21, 2011

Revision Sign-off

By signing the following, the team member asserts that he has read the entire document and has, to the best of his knowledge, found the information contained herein to be accurate, relevant, and free of typographical error.

Name	Signature	Date
Michael Fore		
Sean Owen		
Anh Pham		
Jeffrey Regan		
Alex Welsh		
Matthew Williams		

Revision History

The following is a history of document revisions.

Version	Changes	Edited
Version 1.0	Initial draft.	**3/30/2011**
Version 4.0	Updated to version 4.0	**4/21/2011**

Contents

1. Introduction

1.1 What is *Healing Touch*

Healing Touch is a computerized system designed to provide therapeutic and rehabilitative exercises through gaming on multi-user, multi-touch devices. In addition, the system provides for the collection, storage, and retrieval of relevant game results in order to report patient outcomes for monitoring and comparative purposes. The system was developed for Texas Health Resources (THR) using the Microsoft Surface and Apple iPad as the multi-touch gaming platforms.

Healing Touch includes: a uniform Framework for both platforms that allow for the augmentation of games, a standalone application for clinician usage, and a database that accommodates the patient and game data.

1.2 How is *Healing Touch* Organized

The *Healing Touch* system consists of three primary components:

1) Multi-touch, multi-user devices (Microsoft Surface and Apple iPad) running the *Healing Touch* application
2) Clinician workstation to run the *Healing Vision* application
3) Database for the storage of clinician profiles, patient profiles, and game results

The *Healing Touch* application running on the Microsoft Surface and Apple iPad consists of a Framework and multiple games. Each game is an independent class from the Framework. When a game is called, it is launched separately on the Framework. Only one instance of a game can be created at a time. Once a game is terminated, it sends game data collected from that session to a central database.

The *Healing Vision* application is designed to run on a clinician workstation. The purpose of *Healing Vision* is to offer the clinician an easy way to interact with the information in the database. The application uses information in the database to dynamically create the pages inside the application.

The database environment consists of a Microsoft SQL 2008 R2 server running on a Microsoft 2008 server. In addition to the database, an application called the iPad Helper application must be running to allow the iPad to successfully communicate with the database.

1.3 Before Getting Started

This document covers the development environment, the Framework on the Surface and iPad, the iPad Helper application that runs on the database machine, and how to develop games for the Framework.

This document assumes that the developer has read other project documentation and is familiar with the *Healing Touch* system's purpose and functions. A developer should also have a good understanding of both the Microsoft Visual Studio and Apple Xcode development environments. Required languages are C# for programming on the Microsoft Surface and the *Healing Vision* application and Objective C for programming on the Apple iPad. Familiarity with databases and writing SQL statements is also essential for communicating with the database.

2. Surface

2.1 Setup the Development Environment

2.1.1 Microsoft Surface

There are a few requirements in order to develop new games for the Framework on the Microsoft Surface:

1. Update the Microsoft Surface to Service Pack 1
2. Install Microsoft XNA Game Studio 3.0

These requirements must be done in the order listed. The correct version of Microsoft XNA Game Studio 3.0 can be located at this address: http://www.microsoft.com/downloads/en/details.aspx?FamilyId=7D70D6ED-1EDD-4852-9883-9A33C0AD8FEE&displaylang=en

2.1.2 Windows PC

Development for the Microsoft Surface can also be done on a normal Windows PC with the following installed:

1. 32 bit operating system (XP or later)
2. Visual Studio 2008
3. Microsoft XNA Game Studio 3.0
4. Microsoft Surface SDK

These requirements must be installed in the order listed. The correct version of Microsoft Surface SDK can be located at this address: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3db8987b-47c8-46ca-aafb-9c3b36f43bcc&displaylang=en

When testing the program, the Surface Emulator must first be running in the background. Once the program is running, you can then proceed to run the program from Visual Studio 2008.

2.2 The Framework on the Surface

This section addresses the structure of the Framework.

In Visual Studio 2008, all of the code for the Framework is in the main folder of the project.

The main class is App1. This is the main component to the Framework and acts as the hub for all of the components in the program; the login screen, the home screen, and the games. All database communications must be routed through this class. A text file is located within the same folder. It contains configuration information about connecting to the database.

The login screen is a simple display that waits for input in the form of either a button press or a Surface tag. When a Surface tag is placed on the Surface, the login screen reads the data from the tag and then passes it on to the App1 class, where a simple SQL query is carried out. If the ID is found within the database, a temporary copy of the patient’s internal ID is stored. If the ‘Free Play’ button is pressed, then there is no SQL communication.

On the home screen, the patient can browse games in 2 different views; Album or Grid view. When a patient launches a game, the home screen notifies the Framework of what game is selected and the Framework then disables the home screen and creates an instance of the game. If the game is being played in patient mode, then the Framework queries the database for the specific game options of the game for the patient and passes them off to the game.

2.3 The Protocol

Here is a list of the current functions that communicate with the SQL database, and what they do:

Function Name	Parameters	Returns	Description
Login	String seriesString, String valueString	Int	Queries the database for the patient that has the location of seriesValue and ID as valueString, returns 0 if there were no errors
SubmitQuery	String table, String values	Int	Submits the data collected from the game. Returns 0 if there were no errors.
GetGameOptions	String gameOptionTable	String[]	Gets the list of game options and returns them in the form of a String array.
CheckAdmin	String seriesString String valueString	Boolean	Checks to see if the ID is an admin tag. Returns true if it is.

2.4 Developing Your Game for the Framework

In order to implement a new game into the Framework, a subfolder must first be created within the project and labeled with the name of the game. The code for the game will then reside within this folder. When incorporating images and audio into a game, a new folder must be created within the Content\Resources directory and labeled with the same name as the code folder.

2.4.1 The Game Class

When creating the game class, select add new item within the game folder and select the Game Component template. Once the file has been created, change the inheritance from **Microsoft.Xna.Framework.GameComponent** to **Microsoft.Xna.Framework.DrawableGameComponent**

This class must have the following methods in order to run correctly:

public *nameOfClass* (Game game) : base(game) {}
public override void Initialize(){base.Initialize();}
protected override void LoadContent(){base.LoadContent();}
public override void Update(GameTimegameTime){base.Update(gameTime);}
public override Draw(GameTimegameTime){base.Draw(gameTime);}

Note: All functions that contain base.*function* must have that command as the last line in that function. Any code following that instruction will not be executed.

A second constructor should be made that accepts a string array that contains all of the game options. To prevent large consumption of memory, only the current game will be loaded into memory. The constructor will not be called until the game has been selected from the home screen. If the Framework is in patient mode, then the second constructor will be used to pass along the game options for the specified patient. Once the game ends, all memory allocated to it will be released.

2.4.2 Game Database

Each game has two tables located within the main database, the options table and the data table. The options table should be queried when the game is created and the game data table will be called every time a patient has completed the game.

2.4.3 Game Instruction Pages

Every game is required to have an instruction page. This page, or pages, give a description of the game and the rules that must be followed to play it. There are several buttons required for these pages:

Name	Location	Description
Back	Bottom left corner	This button allows the patient to return to the home screen.
<<Prev / Next>>	Centered at the bottom of the screen	If there is only one instruction page, then these buttons are not required.
Launch	Bottom right corner	This button allows the patient to launch the game.

2.4.4 Developing Your Game for the Framework

In order to implement the game into the Framework, several sections of the App1 and HomeScreen classes must be modified to allow the game to be recognized:

- Within App1
  - Add an enumeration to the HealingState enum that has the same name as your game
  - Declare the game object at the top of the App1 class
  - Within the Function ChangeHealingState (HealingState state)
    - In the HealingState.HOME case, append within this sub switch statement a new case for your game follows the same behavior as the previous games.
    - In the main switch statement, create a new case for your game that follows the same behavior as the previous games.
- Within HomeScreen
  - Add a private constant with the name of the game to the end of the list of games, with the value 1 more than the current highest number.
  - Include two home screen images of the game within the Content/Resources/Pictures/Home Icons folder
  - Append the image names to the icons array.

3. iPad

3.1 Setup the Development Environment

The following is required to develop on the iPad

- Mac OSX 10.6 or greater
- Apple developer account

The lastest Xcode and iOS SDK 3.2 or greater can be downloaded at **developer.apple.com**

OSX may need to be updated to install Xcode and iOS SDK.

The source code for the iPad can be found on the *Healing Touch* DVD in the Dev/iPad/HealingTouchProject folder. The program can be built and run in the emulator in Xcode.

In order to run the application on an iPad, the developer must have a provisioning profile from Apple that is tied to their developer certificate, the iPad being used, and an application ID for the program. The profile will need to be renewed every 3 months by an Admin account in Apple iOS Developer program.

If this is the first time the iPad is used for development:

- Plug the iPad into the computer being used for development
- Open Xcode/Window/Organizer
- Select the iPad and select “Use for development.”
- Install the provisioning profile in it. Remember to change the app ID of the program to the one in your provisioning profile.

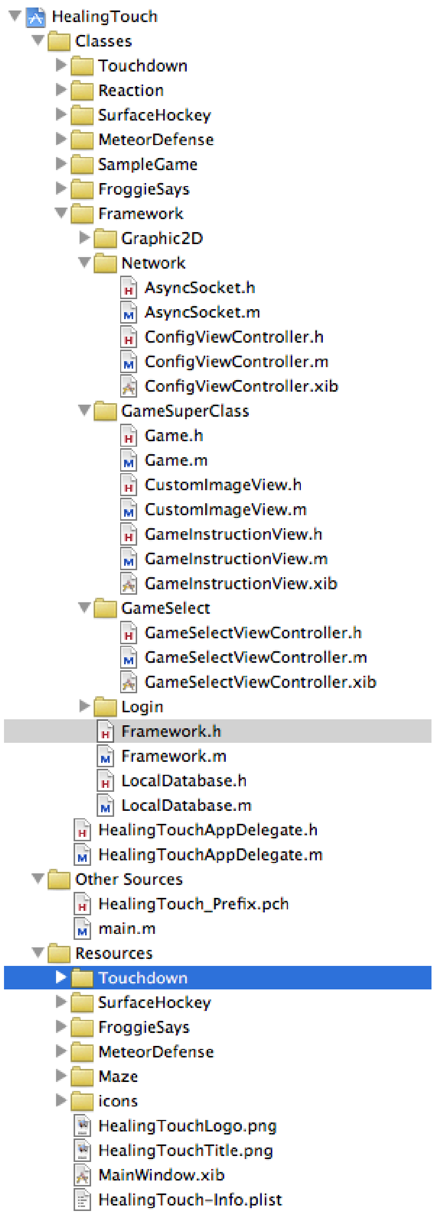
3.2 The Framework on the iPad

In Xcode, all the code for the Framework is in Classes/Framework group.

The main class is Framework. It is a ViewController that coordinates other Framework’s ViewControllers and acts as the info hub for all the components: the patient login view, the game select view, the clinician config view, the local database, and the list of games.

The iPad is intended to be used by one patient for an extended time, and cannot require main database activity. Thus, Framework object only needs to keep 1 patient ID. This ID is stored in the program and is used to save game data and auto-login. To change this ID the clinician needs to login (requires main database connection) and reset it to another patient.

The patient login view simply displays the view and handles events for login and transfer to clinician login.



```
HealingTouch
  Classes
    Touchdown
    Reaction
    SurfaceHockey
    MeteorDefense
    SampleGame
    FroggieSays
    Framework
      Graphic2D
      Network
        AsyncSocket.h
        AsyncSocket.m
        ConfigViewController.h
        ConfigViewController.m
        ConfigViewController.xib
      GameSuperClass
        Game.h
        Game.m
        CustomImageView.h
        CustomImageView.m
        GameInstructionView.h
        GameInstructionView.m
        GameInstructionView.xib
      GameSelect
        GameSelectViewController.h
        GameSelectViewController.m
        GameSelectViewController.xib
      Login
      Framework.h
      Framework.m
      LocalDatabase.h
      LocalDatabase.m
    HealingTouchAppDelegate.h
    HealingTouchAppDelegate.m
  Other Sources
    HealingTouch_Prefix.pch
    main.m
  Resources
    Touchdown
    SurfaceHockey
    FroggieSays
    MeteorDefense
    Maze
    icons
    HealingTouchLogo.png
    HealingTouchTitle.png
    MainWindow.xib
    HealingTouch-Info.plist
```

In the game select view, the patient can browse the games in 2 views. When the patient launches a game, the view notifies the Framework to do that. The patient does not ever need to logout. The logout button is for the clinician to enter admin mode and reset the program for another patient.

The clinician view (ConfigViewController) may be the most complicated component because it handles connectivity to the main database and data synchronization. The Framework uses the CocoaAsyncSocket library to connect to the iPad Helper Application using a TCP connection and a custom protocol discussed Section 3.3. For data synchronization, the clinician can upload the data in the local database to the main database server and reset the local database for another patient. Game options for the new patient will then be downloaded into the local database.

The local database class is the wrapper for sqlite commands to store and retrieve data from an sqlite database. There is also a “Graphic2D” group in the Framework group: Located here are some of the classes used by several games.

3.3 The iPad Helper

Unlike the Microsoft Surface, the iPad cannot communicate directly with the SQL server because there is no library to support it. The iPad also cannot connect to the main database when the WiFi is turned off, the connection is very weak, or if the main database is not connected to the Internet.

The iPad Helper Application is a program written in C#. It runs on the same machine that hosts the SQL Server, or in the same network with the SQL Server so it can query the main database. The Helper Application listens on a specified port so the iPad Framework can connect to it. The Helper acts as a middleman between the iPad Framework and the SQL Server.

The communication uses a custom protocol for several reasons:

- Because the TCP connection is asynchronous on the iPad Framework side, once it sends a query, there is no way it can distinguish the returned result from another query's result. Using a protocol, we can insert custom data for each returned result.
- A protocol gives us more control over the Helper Application. The Framework can ask the Helper Application to signify the end of a series of queries, or custom error reporting (SQL exceptions).

3.4 The Protocol

The protocol is defined as such. Each command over the TCP connection is delimited by "\r\n". For each command sent from the Framework, the Helper will immediately process and return the result, while still listening for incoming commands. The results will also be delimited by "\r\n". In each command, the components will be separated by colons. Data sent over the protocol is limited in one line, no colons. Because the data being sent over is game data and game options, the limit can be easily dealt with. Also, it is not designed for massive data transfer: Total data transferred in a connection can be large; but each command shouldn't be more than a few hundred KB.

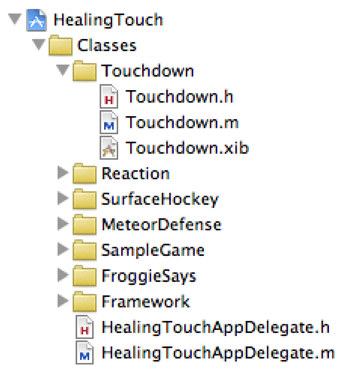
Command	Explanation	Result	Explanation
admin:(Username):(Password)	Check the clinician credential	status:3:(locationID)	locationID of the clinician. If the ID is not there, the login is considered fail.
patientid:(PatientID):(PatientLocationID):	Check the patient credential	status:1:(InternalID)	theInternalID of the patient, used to save game data. If

(isAlbumView)			it's 0:patient not found
sql:(sql)	sql query	status:2	just a confirm reply
schema:(table)	get the schema	(create table sql statement)	create the sqlite table from sql server table
gameoption:(internalID):(game option table)	get the game option for this patient	(2 insert sql statements)	insert game option for the patient id and the default option (internalID=0).
EOR:(message)	end of request (upload/reset) signal	status:0:(message)	reply back the same message

3.5 Developing Your Game for the Framework

All game files should be placed into the same group. All games should be designed in landscape mode.

3.5.1 The Game Class



The game class must extend the Game class. It is a UIViewController class that has a few things: a pointer to the Framework, 2 custom init methods, the game icon and reflection (for the game select view), the description and the game name, and start and end methods.

3.5.2 Override Methods

You should overwrite the initWithFramework: method, but not the initWithNibName:Framework:icon: method. For the super init method, call initWithNibName:Framework:icon: And in the init method, the only things you should set is the description and gameName (only if the game has the GameOption and GameData tables).

```
@implementation Touchdown

-(id)initWithFramework:(Framework *)f{
    self = [super initWithNibName:@"Touchdown" framework:f icon:@"TouchdownLogo.png"];
    if (self) {
        gameName = @"Touchdown";
    }
    return self;
}
```

The shouldAutorotateToInterfaceOrientation: method should not be overwritten. All the games have only landscape mode.

Every time the patient launches a game, the Framework will load the view of the game and call the start method. All variables for the game should be initialized here. The end method is setup with IBAction. It can be either connected with a UI element or called. Variables initialized in the start method should be released in the end method. Remember to call [super end]. Again, check the existing games for example.

3.5.3 Game Database

Each game has 2 tables: {gameName}GameData and {gameName}GameOptions. Set the gameName so the Framework knows what tables to work on when saving and retrieving data for your game.

The {gameName}GameOptions will have a column named “PatientInternalID” that corresponds to the patient.

The {gameName}GameData will have the first 3 columns named “PatientInternalID,” “DatePlayed,” and “HardwareType.” After that, create whatever columns are needed for the game.

Use the Framework pointer to getOptionsFor:self and saveGameData:to:self. The getOptionsFor:self will return at least 1 row (default option) and at most 2 rows (patient option and default option) in descending order of the PatientInternalID. It is guaranteed to have the first row available. The first row is patient option if available and default option if not. When saveGameData, concatenate the data into a string delimited by coma, put in single quote if text type, in the same order of your column in GameData table. Note: the first 3 columns are filled by the Framework. The developer only has to create a string for the game data and give it to the Framework to save.

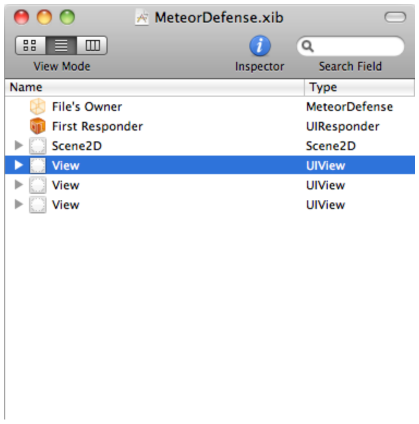
When you get the game’s option and save data, enclosed in the ifdef preprocessor as shown here. That is

```
-(void)start{
    [super start];
    #ifdef NODATABASECONNECTION
    traversalFreq = 2;
    displayType = @"colors";
    [layover changeType:displayType];
    countDownTotal = 3;
    delayTotal = 1;
    #else
    NSArray *result = [framework getOptionsFor:self];
    traversalFreq = [(NSString *)[result objectAtIndex:1] floatValue];

    displayType = [[result objectAtIndex:2] retain];
    [layover changeType:displayType];
    countDownTotal = [(NSString *)[result objectAtIndex:3] intValue];
    delayTotal = [(NSString *)[result objectAtIndex:4] intValue];
    #endif
```

when you want to test the game, but cannot connect to the database for the moment; you can define that in Framework.h; so the game option can be set instead of it looking in the local database. Do that also when you save game, so it just discards the data instead of saving it.

3.5.4 Game Instruction Pages



Name	Type
File's Owner	MeteorDefense
First Responder	UIResponder
Scene2D	Scene2D
View	UIView
View	UIView
View	UIView

Instruction pages are optional. If there is no instruction pages, the game will start when the user launch it from the Game Select screen. Instruction pages should be created as Views in the game's nib file. In the game's viewDidLoad, create new array for gameInstructionViews and add these instruction pages to that. Each time the user scrolls to an instruction page, it calls the game viewInstructionView:(int) method so you can make custom animation for the instruction page. See the game MeteorDefense for sample.

```
// Implement viewDidLoad to do additional setup after loading the view, typi
- (void)viewDidLoad {
    [super viewDidLoad];
    gameInstructionViews = [[NSArray alloc] initWithObjects:page0,page1,page
    textLightup.alpha = 0;
}
```

```
-(void)viewInstructionPage:(int)page{
    switch (page) {
        case 0:
            if(textLightup.alpha == 0){
                [UIView beginAnimations:nil context:NULL];
                [UIView setAnimationDuration:5];
                [textLightup setAlpha:1.0];
                [UIView commitAnimations];
            }
            break;
        default:
            textLightup.alpha = 0;
            break;
    }

}
```

3.5.5 Add Game to the Framework

To make the game appear in the Game Select screen, go to Framework.m, import the game's main header file. Go to the initFramework method and add the game instance with the following line:

[games addObject:[[[{YourGameClass} alloc] initWithFramework:self] autorelease]];

The game is complete.

3.5.6 Miscellaneous

Graphic2D folder: contains things used in various games. The Scene2D is a ScrollView that catches the touch events and can pass that to the game (ViewController). The game can also draw onto the Scene2D.

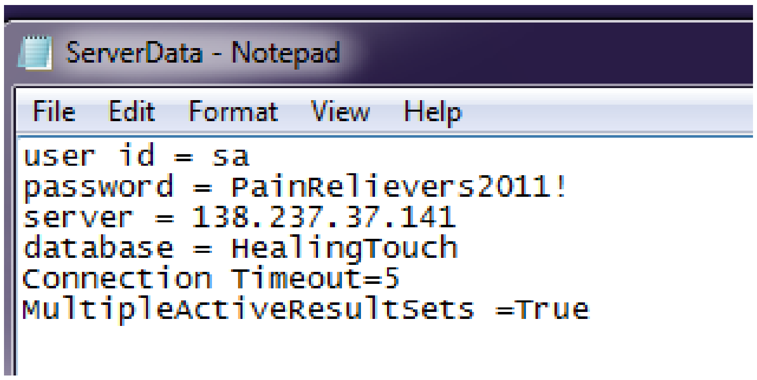
The Entity2D is an Object with an image, acceleration, speed, and position. The developer can keep track of each Entity2D object's properties used for animation.

The developer can add custom own classes here for use across multiple games. Do not reference from one game to another. Each game is supposed to be independent of each other. Removing one game should not affect other games.

4. *Healing Vision* Application

4.1 Setup the Development Environment

To set up the development environment for *Healing Vision*, install Visual Studio 2010 and select the .NET 4.0 Framework. A text file called “ServerData.txt” holds the information needed to connect to the database. Below is a screenshot of the text file used by the 2010-2011 Senior Class. Note that for *Healing Vision* to operate properly, MultipleActiveResultSets must be set to true.



```
user id = sa
password = PainRelievers2011!
server = 138.237.37.141
database = HealingTouch
Connection Timeout=5
MultipleActiveResultSets =True
```

4.2 Loading the Database Schema

To install the database schema, follow these instructions:

1. Open Microsoft SQL Server 2008 R2
2. Go to File > Open > File
3. Navigate to HealingTouchDVD\Installers\Database\HealingTouchDB.sql
4. Run the generated query

Run the following command in a new query, replacing <location> with your clinic location ID and <Password> to the desired root password.

insert into Administrator values (0,'root','root',0,'root'**,<location>,**'root@root.com','<**password**>');

This will set-up the root account.

4.3 Adding a Game

Several operations must be performed in the database for new games to work properly. If this is done properly, nothing has to be done to *Healing Vision*.

1) The game name, its data table and its options table must be added to the "Game" table. This will have to be done by interfacing with the database directly.
2) The data and option tables need to be created with the exact names that appear in the "Game" table. If they do not, *Healing Vision* will be unable to find where to pull game options and game data.
3) The game options for the game must be added into the options table. If this is not done, *Healing Vision* will be unable to set the game options for this game.
4) The game options for the default patient must be set before a game starts. If the options are not set, the Framework will not know what game options to use and will likely fail.

4.4 How Each Page Works

Healing Vision is meant to be extremely flexible and was designed to require zero reprogramming when new patient attributes are added or when new games are added to the Framework. This was accomplished using a great deal of metadata in the database. *Healing Vision* relies heavily on the metadata's completeness and correctness. If for some reason information is unavailable or incorrect, *Healing Vision* will be unable to perform as it was meant to. The discussion below provides a brief overview of how each page in *Healing Vision* works and how metadata plays a vital role in the functionality of the application.

4.4.1 Add Patient Page

The first page a clinician sees and perhaps the most important page is the "Add Patient Page." The "Add Patient Page" generates a form dynamically by pulling all the patient attributes from the "PatientAttributeValues" table and determining if those attributes are to be presented and, if so, how they are to be presented. This is done using several metadata fields. The first field is "IsEntered." If true, this attribute is meant to be entered by the clinician and will be shown on screen within the form. The next metadata field is "HasDefinition." This field is used to determine if the attribute has an enumerated set of values in the "PatientAttributeDefinition" table. If this is true, a combo box will be used to allow the clinician to choose a value. These values are populated from all the definitions in the "PatientAttributeDefinition" table with the attribute ID equal to the attribute in question. If "HasDefinition" is false, then *Healing Vision* checks the "DataType" metadata field. If "DataType" is equal to "date," two combo boxes will be used in the form (one for year and the other for month). The last option for GUI element is a textbox.

For validation, all fields except the "Additional Information" field are required to be filled in or have some valid value. Specifically, text fields are required to have at least one character and combo boxes (date elements included) are required to have a valid item selected.

If all fields were successfully validated, *Healing Vision* will attempt to create a PDF file that contains all the patient information recently entered. This form is to be kept by the clinic to log patients into the *Healing Touch* applications and to tie IDs in the database with real people. To avoid legal complications, *Healing Vision* does not store any data that would be able to distinguish a person. Therefore, the first and last name that appear in the pdf file are not stored in the database.

If the form was validated and the PDF was created, the information is sent to the database. This is done by creating an insert statement for each attribute in the form. This approach makes the transmission of data simple and safe.

4.4.2 Edit Patient Page

The “Edit Patient Page” is similar to the “Add Patient Page.” When a clinician navigates to the “Edit Patient Page,” they will be asked to enter a valid patient ID. It is important to note that this patient must have the same location as the clinician. If they do not have the same location, the clinician will get an error when trying to look up the patient. This feature makes it possible for clinicians to only edit patient attributes for patients that visit their clinic. Upon successfully entering a patient ID, a form is generated exactly the same way as in the “Add Patient Page.” Once the form is generated, the element values for each attribute will be chosen based on the patient’s data. If the patient does not have a value for a certain attribute, the attribute will be left blank. The validation and transmission of data works the same way as the “Add Patient Page.” A pdf is not generated after a patient is edited.

4.4.3 Reports Page

The “Reports Page” is very simple. The items that appear in the stack panel are the report names from the “ReportName” field in the “Reports” table in the database. When a report is selected, the report description text box will populate with the information from the “ReportDescription” field in the “Reports” table. Upon generating a report, a SQL statement will be extracted from the “ReportSQL” field in the “Reports” table and be used to populate a dataset object. This object is passed to the “ReportData” class which creates a window with a datagrid object and uses the dataset as the datagrid’s item source. In other words, the results of the SQL query are used to populate a grid in another window.

4.4.4 Query Page

The “Query Page” is by far the most complicated page in Healing Vision. This page allows clinicians to build, generate, and save queries about patients and games using dynamically generated forms. The page is split into two sections. The first section is the patient section. Here a clinician is able to select “All Patients” or “Select Patients.”

If the clinician selects “All Patients,” a form will appear exactly like the form in the add and edit patient pages, except there will be a combo box with comparative signs between the attribute label and the GUI element used to enter the attribute value. This allows clinicians to select exact values or ranges of values for their queries. If the clinician does not select a comparative operator or enters invalid data into the GUI element, the item will not be included in the query and will not generate an error.

There are two different types of comparative combo boxes. The first one is used for character data. The combo box contains and equals (=) and not equals (<>) operator. The second type of comparative combo box is used for dates and numeric data. This combo box has a less than (<) and greater than (>) operators as well as equals and not equals. Healing Vision determines which combo box to use by referencing the “DataType” field in the “Patients” table and the “AttributeDataType” field in the “GameDefinitionTable.”

If the clinician selects “Select Patients,” the clinician will be able to add patient IDs to a list box. If the clinician tries to enter a patient ID that does not exist or does not share their location, an error prompt will appear.

If the clinician selects “Add Game,” he or she will be able to select from all the games in the “Game” table. Once a game is selected (only one game can be selected at a time), the data from the “GameDefinintion” table will be used to populate the game area. Using the data type field in the “GameDefininition” table Healing Vision is able to determine how to display the data within the form. If the “AttributeDataType” field is equal to varchar, a comma delimited list (also present in the “AttributeDataType” field) will be used to populate a combo box item. If the data type is a date or datetime, three text fields will be used (one for year, month, and day). If neither of these data types appear, Healing Vision defaults to a text field.

Patient Section	**Game Section**	**Query Outcome**
All Patients Selected	No Game Selected	All patients matching the criteria expressed by the patient form are selected.
All Patients Selected	A Game is Selected	An inner join is performed on all the patients that matched the criteria of the patient form and all the games that matched the criteria on the games form.

Selected Patient	No Game Selected	All patient data from the patients that were selected will be selected. This is done by selecting all values from the "Patient" table for each ID and performing a union of all the results.
Selected Patient	A Game is Selected	The game data matching the specified criteria in the games form will be joined will all the selected patients IDs.
No Patients	A Game is Selected	All the games matching the criteria in the games form will be selected.
No Patients	No Game	An error will be generated.

4.4.5 Games Page

The "Games" page is used to set the game options for a specific patient or set the default options for the first time any patient plays a game. First, the clinician must select a game from the combo box in the top right. This combo box is populated with all "GameName" values in the "Game" table. After the clinician selects a game and hits "Select Game," he or she will be able to lookup a specific patient or set the default options. Regardless of the choice, the function works in near the same. A form will be generated using the values from the "GameOptions" table that have the same "GameID" as the selected game. Next, the values are selected from the game options table of this specific game. The game options table name is acquired from the "Game" table. If there aren't any values, the fields will be left blank. Otherwise the fields will have selected values. If the default patient is selected, the game options for the patient with internal ID of zero will be selected. Note that location is not used to find the default patient. Therefore, a location can change the default game options of a different location.

4.4.6 Administration Page

The "Administration" page is used to add clinicians. Unlike most of the other pages in Healing Vision, the form on this page is static. When the "Add Clinician" button is clicked, the data is save to the "Administrator" table. Also, a PDF is generated with the clinician's information. This is similar to the card generated in the "Add Patient" page.

4.4.7 Reprint Cards Page

The “Reprint Cards” page allows clinicians to lookup a patient’s information, and regenerate that patient’s card. When the clinician looks up a patient’s ID, a form is generated dynamically (just as in “Add Patient”) and populated with the patient’s information. Also, first name and last name text fields are added to the form, so the clinicians can generate cards that look exactly the same as the cards generated from the “Add Patient” page. All fields in this form, with the exception of the name fields, are not editable. Also, this page does not send any data back to the database.

4.4.8 View Variables Page

The “View Variables” page consists of a datagrid populated with information from the “GameDefinition” table. The clinician is able to click “New Window” which will pass the dataset containing all the grids information to the “ReportData” class. The “ReportData” class creates a separate window with a datagrid, and populates this datagrid with the information in the dataset. This allows clinicians to view variable data without having to click back and forth between pages.

Appendix A. TCU CSLab2 Network Setup

IP range: 138.237.37.151-156

Gateway: 138.237.37.130

Subnet Mask: 255.255.255.128

DNS: 138.237.49.138, 138.237.49.157

Our IP Layout

138.237.37.151	Surface 1
138.237.37.152	Surface 2
138.237.37.153	iPad 1
138.237.37.154	iPad 2(Dr. Payne's iPad)
138.237.37.155	TTL330003
138.237.37.156	spare
138.237.37.141	Brazos (database)