The Art of Computer Programming – A difficult book to understand

Donald Knuth

Dennis Ritchie

B.Sc.(Hons) in Software Development

April 26, 2016

Final Year Project

Advised by: Dr Alan Turing

Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract The Let Me Show You application is a communication application that allows the user to express events in a story board mode. The user can choose to either load a previous story they have told or use the interface to express themselves by taking advantage of the applications animations and audio feature. To achieve the level of interaction required we used a game engine so as to take advantage of the built in animation and interaction features that it is designed for. The majority of the coding is done in C and is organised in Scripts. The other language that was used is JavaScript and this is used for the file picker interface. As this is been developed for portable devices such as iOS and Android we designed many of the assets features with listeners and trigger methods to give a seamless interaction. From our time spent developing this application we found that when developing a piece of software such as this it is very important to keep the target client in mind. It is also important to not assume anything when it comes to the user interface, as with educational software the client's familiarity with technology can vary a great deal. With this the app encourages the user to interact with the interface with no repercussions as the assets are designed to be interacted with. Through this method we found that the user learns quickly how the application works. We wanted the user to try and use this application as a stepping stone to trying other technologies as many of the features are universal in technology today.

Authors Danial Maloney: Software Development student Galway Mayo Institute of technology.

Kieran Redington: Software Development student Galway Mayo Institute of technology.

Chapter 1

Introduction

A prototype visual communication app was developed by David O Gorman in 2014/2015. This year for our final year project we were approached at the beginning of the year with a proposal. We were tasked with taking the app as it is, taking the core design and what it aims to do to and making it cross platform and add some extra features on request. A lot of work in researching this project and coming up with a design was done by David and it is important that his work is not over looked. The application is aimed at helping people who have communication trouble as a result of Autism to be able to use technology to improve communication and improve day to day life as a result. Much of the tedious work in researching papers and working alongside the client whom this app is developed for in order to come up with a basic design principle was done by David. For the sake of privacy we will refer to the client whom this software is developed for as Tom and his parents Michael and Mary. Michael was especially vital to the development of the software as we did not have any real world experience in the needs of a person with Autism. Working with Michael gave us an insight into the core requirements and gave us a link to the prototype as he was involved with David in his development. With this experience behind us we were able to avoid many potential pit falls that may have cost previous developments time and resources in their development. We were giving two main objectives from the beginning of this project. The app must be cross platform with a focus on an iOS it must also have the ability to import images from the user's local system and then be manipulated and added to different stories. Some of the character images will also need to be animated. The prototype had verbal clues and a few animations so it was important to the client that these features were not lost and if possible be improved. The client did approve the addition of our own concepts and design ideas but maintained that the original core design principles be maintained.

Required Project Features:

- File picker with the ability to import and save images to a story.
- Save a story that has been create.
- Have animations that allow the character to communicate feelings of hunger, happy sad, etc.
- Have this application available for iOS and android.

After reading David's dissertation we had an idea of what Michael and Mary wanted for a finished design. Once we had a design that we agreed on we needed to find a platform that we could use to give us an app that could fulfil these prerequisites across multiple platforms but still maintain the original design. An in detail review of the different platforms we reviewed can be seen under the Technology Review heading. Literary Review: During the term of this course we did some research on the topics of Technology in general education and a more relevant to our project we looked at technology in the education of people with learning difficulties. For more in depth information on this research you can read the literature reviews attached at the end of this document. What was important for us about these reviews is the insight into the basic structure of what we needed to create. When reading different journals we found that many technologies fell into the trap of been overly complex for their market. It is important that when designing these technologies for a person with learning difficulties that one does not design it with their own level of expertise as the target cliental. The key components of this app would have to rely heavily on sight and sound as the means to communicate the actions and requirements of the app. Visual clues backed up the a vocal confirmation is the ideal structure to help someone such as Tom and or client to communicate.

Chapter Breakdown

- Methodology takes us through which method we used to develop this project. It focuses a lot on describing the type of Agile method we used and gives a review of why we used the platform we picked.
- Technology Review discusses all the different platforms that we researched and tried before picking Unity. We will also discuss the different plugins that were used and when and where we needed to change and further develop these plugins.
- System Design shows the URL diagram for the project and brings attention to the key structural parts of the application.

- System evaluation we will discuss testing and how the results effected our build. We also discuss what aspects of the Design we may have done differently.

Chapter 2

Methodology

For this project we decided to use an Agile Methodology with a focus on scrum. With so many other projects been undertaken throughout the year it made sense to use a methodology which allowed us to set short term goals that we could achieved between other projects. We used the ethos of doing a little a lot rather than a lot a little. The obvious obstacles that come with numerous subjects meant that we had to be adaptive. It was not possible to do daily cycle but we did have a weekly cycles in which we set out our weekly tasks and discussed if and when we achieved any goals.

The most difficult aspect of this project at the beginning was trying to find a suitable platform. This was a task which we may have underestimated as picking a platform that would later make the project unfeasible later in development would have cost us a fatal amount of time. For this reason we spent a lot of time at the beginning picking a platform and running through the project plan in our scrums to see if the project was possible using that software. We picked 4 of the most popular cross platform software and decided to test the pros and cons of each one [enter reference of pros cons list here, Xamarin, Unity, Phone Gap, 5APP]. This also means that a lot of time was spent testing and researching platforms that we ended up not using. In the end we came to the agreement that Unity game engine was the ideal platform.

Unity offered us many advantages, one of which was that we were familiar with it from previous projects and that would help us make up any lost time that was spent looking for a platform. The fact that unity is a game engine also means that it will maintain the screen ratio and design across all the main platforms. The languages that are available for unity are Java script

Figure 2.1

and C. We decided to use C based on our familiarity with it and because we needed to build our own file picker it was important to have a solid grasp on the language as this tasks would provide the most treacherous obstacles.

Based on the platform that we used for this software and the market that it has been developed for the best method of testing we found was user testing. By allowing people with no knowledge of the project to use and even at times to try and crash the application we found that this found many of the kinks and fall downs very quickly. The best approached was to allow people of different ages to test the application and gather all opinions. By gathering these opinions and errors alike we developed a list of tasks to complete. Some of these tasks did not seem feasible in the time frame allowed so the most important and urgent tasks were moved to the head of the queue.

Chapter 3

Technology Review

The most important aspect of this project was the selection of our development tool. When developing the prototype David felt that Visual Studio was a very powerful tool but it was limited when trying to achieve the file picker and trying to get the images to interact smoothly. We also wanted to start with a tool that was advertised as a cross platform tool rather than having to develop for each OS individually. Below is the results of our search for the perfect Software that would fill most if not all our requirements.

Xamarin

Xamarin: Pros –

- The prototype was developed in Visual studio using C. Xamarin can use C to code all platforms.
- Clients main platform request is iOS and Xamarin has strong support for iOS and android in particular.
- Allows the creator to create apps that look native to the particular device.
- Native applications will run faster.

Xamarin: Cons –

- The Xamarin licence is to expensive
- OS updates will out-date the application if it is not update adequately as it will be native. We will not be available to update the application to new specification's.

- There is a deep learning curve, finished look of app does not justify time spent.

Summary – The Xamarin platform is very impressive and allows the developer to make apps that are native and run at fast speeds. With this kind of speed and native development the developer must be available to constantly update the code so that any OS updates do not cause bugs. We felt that the need for constant updating after we have finished with the project and the yearly price of running Xamarin does not justify using this tool. On A side not while researching the potential of this tool to create an interactive app with a heavy emphasises on texture manipulation the results were very limited without a team of programmers. This tool is more suited to text based apps with graphs and business type applications.

PhoneGap

Pros –

- Uses HTML JavaScript and CSS which have some experience with and can work with and learn more about.
- Has a wide range of plugins which also enables access to native APIs.
- Open source and free

Cons –

- Uses plugin architecture. This would mean that should we require a plugin that does not exist then we would have to develop one ourselves. This could potentially be a project in itself as many if not all the basic plugins are developed.
- Reviews of PhoneGap applications have suggested that the apps performance leaves a lot to be desired. The newer versions have closed the gap in performance but our application will require a lot of processing power.

Summary – The PhoneGap tool is an impressive piece of software considering it is open source. We originally decided to use this tool to develop our app. It had all the features that we needed to fulfil the requirements of the build. When it came to making the final choice on which platform to use the decision was made on the aesthetics of the finished project rather than the ability of PhoneGap itself. This is definitely software that we would work with in the future.

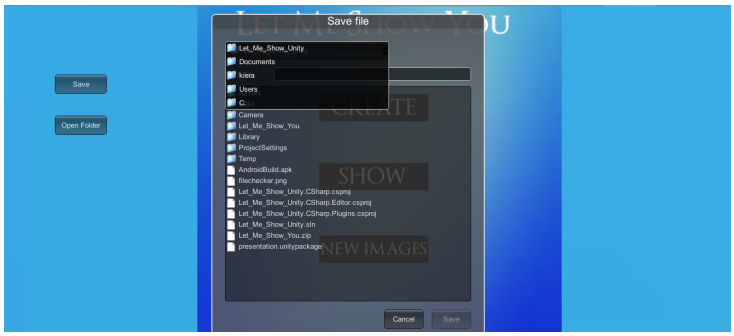

Figure 3.1

Unity Game Engine

Unity is a game engine that allows the user to develop apps and games across a number of devices including iOS, android and windows. The game engine itself comes with a range of features that benefited our development. The inbuilt animator was a used to give the user a seamless interchange between animations and pop out windows. The strength of Unity is in its ability to render both 3-D and 2-D development. This was important as we wanted to achieve a 2.5-D effect in the app. Unity is a nice balance between plugins and original coding from the developer. Many of these plugins can be limited in what they offer and often need to be coded heavily before they are of any use.

Plug-Ins

The biggest plug-in we used was the file navigator [1] [3.1]. This gave us a UI in which to navigate the local system. The uses of this was very limited in its original form as its only function was to identify files and return the name of the file location. In order to get this working to our own needs we needed to add a method that saves the file location and then import it into the Resource folder as a Texture 2D image so it could be used in the application. You can see from the code that we first hade to identify the file location before been able to extract a copy into the app. The SaveFile method takes the file location as a string.

```
void SaveFile(string pathToFile)
{
    var fileIndex = pathToFile.LastIndexOf(pathChar);
    message = "You're saving file: " + pathToFile.Substring(fileIndex + 1, pathT
```

```
    fileName = pathToFile.Substring(fileIndex + 1, pathToFile.Length - fileIndex
    imageSelect = pathToFile;
    if (pathToFile != null)
    {
    LoadPNG(pathToFile);
    children.Add(imageSelect);
    Debug.Log(imageSelect);
    }
    Fade();
}
```

The string is trimmed and passed into the LoadPNG if the string is not null. The LoadPng method gets an actually copy of the image and takes it in as a Texture. The texture is still a string until it is converted into a byte array allowing us to convert into the Texture 2d image that we require for unity. After converting the file to the Texture 2d it must be saved to a location of our choosing the in project itself.

```
public static Texture2D LoadPNG(string filePath)
{
        Texture2D tex = null;
        byte[] fileData;
        if (File.Exists(filePath))
        {
                fileData = File.ReadAllBytes(filePath);
                tex = new Texture2D(2, 2);
                tex.LoadImage(fileData);
                 dimensions.
        }
        SaveTextureToFile(tex, filePath);
        return tex;
}
```

In Unity in order to load images during run time all the required images and assets must be available in the Resources folder. In order for us to be able to access files that we already have loaded we had to save them directly into the resource folder. The SaveTextureToFile method determines where the files are saved during run time allowing us to access them on the fly. The fileSave variable is very important as by just changing the value of it allows us to have a number of storage locations in the Resources folder and separate the different images into different folders.

```
public static void SaveTextureToFile(Texture2D texture, string filename)
{
        byte[] bytes;
        bytes = texture.EncodeToPNG();
        File.WriteAllBytes(Application.persistentDataPath + "/" + fileName, byte
        System.IO.FileStream fileSave;
        fileSave = new FileStream(Application.dataPath + "/Resources/Stickers/St

        System.IO.BinaryWriter binary;
        binary = new BinaryWriter(fileSave);
        binary.Write(bytes);
        fileSave.Close();
}
```

Populating Side Bar

After importing new images to the application we need to be able to view and select these images and the stock images. To do this we created a side bar that can slide into view on command. The user can scroll through the assets and select different ones as they see fit to tell their story. To populate the sidebar we used the AddElement script alongside Unity's in built canvas and canvas scaler. The side bar button is located on the top right of the display and when selected it animates the side bar to appear using the sideBarButton() method in the the AddElement.cs script.

```
public void dropBarButton()
{
        if(!assetBarActive) {
                if (dropBarActive) {
                        slideOut(anim);
                        dropBarActive = false;
                        btnMenu.transform.GetChild(0).GetComponent<Text>().text
                        } else {
                        btnMenu.transform.GetChild(0).GetComponent<Text>().text
                        slideIn(anim);
                        dropActive = true;
                        }
                } else {
                        slideOut(animStickers);
                        assetBarActive = false;
```

```
        }
}
```

Once the menu is opened it has to be populated instantly and display all the given assets. To do this the populateMenus() in the AddElement.cs is activate when the AddElement script is started. This method will scan the given folder and display any Texture 2D files that are in there.

```
private void populateMenus(string resource, string prefab, GameObject parent)
{
        foreach (Texture2D t in Resources.LoadAll(resource, typeof(Texture2D)))
        Rect r = new Rect(0, 0, t.width, t.height);
        GameObject i = Instantiate(Resources.Load(prefab)) as GameObject;
        Sprite temp = Sprite.Create(t, r, new Vector2());
        i.GetComponent<Image>().sprite = temp;

        if (prefab.Equals("AssetSelector")) {
        assetSprites.Add(t.name, temp);
        }
        i.transform.SetParent(parent.transform);
        i.transform.position = i.transform.parent.position;
        i.name = t.name;
        }
}
```

For the most part we used C for this build. Some features of the plugins were programmed in JavaScripts however so we needed to fuse both languages together. The file picker UI is written in JavaScript. The segment of code below is from the UNIFileBrowser.js. Keeping in mind when choosing a file picker that the application must be cross platform we found that unity has many inbuilt code features to adapt to whatever device it is been used on.

```
// Touch scrolling in file area rect
#if UNITY_IPHONE || UNITY_ANDROID || UNITY_BLACKBERRY || UNITY_WP8
private var touchPos : Vector2;
private var touchScrolling = false;

function Update () {
        if (Input.touchCount > 0) {
                var touch = Input.GetTouch(0);
                var relativeTouchPos = touch.position;
```

```
                relativeTouchPos.y = Screen.height - relativeTouchPos.y;
                relativeTouchPos -= Vector2(fileWindowRect.x, fileWindowRect.y);

        if (touch.phase == TouchPhase.Began && fileAreaRect.Contains (relativeTo
                touchPos = relativeTouchPos;
                touchScrolling = true;
                return;
        }
        if (touchScrolling && touch.phase == TouchPhase.Moved) {
                scrollPos -= relativeTouchPos - touchPos;
                touchPos = relativeTouchPos;
        }
        if (touch.phase == TouchPhase.Ended || touch.phase == TouchPhase.Cancele
        touchScrolling = false;
        }
        }
        else if (touchScrolling) {
                touchScrolling = false;
        }
}
#endif
```

Animation

The animations feature of this project is one of the features that were outlined in the project requirements. Unity offers a very varied number of options when setting up animations. For our animations we needed to be able to select the character and display a pop-up display with a number of emotions to choose from. On selecting these emotions the character would animate respectively. To get the UI to display we first set up a script called GUIPopUp.cs to display a number of buttons with the associated emotions as both images and text. A separate method is used to initialise each animation. These methods are a derivative of code snippets found online. [2] [3]

```
public void OnGUI()
{
        windowRect0 = GUI.Window(0, windowRect0, DoMyWindowHappy, "Happy");
        windowRect1 = GUI.Window(1, windowRect1, DoMyWindowSad, "Sad");
        windowRect2 = GUI.Window(2, windowRect2, DoMyWindowHungry, "Hungry");
```

```
        windowRect3 = GUI.Window(3, windowRect3, DoMyWindowTired, "Tired");
        }
        public void DoMyWindowSad(int windowID)
        {
        wheatButton = (Texture)Resources.Load("sad");
        if (GUI.Button(new Rect(10, 20, 100, 20), wheatButton))
        {
        gob.transform.GetComponent<Animator>().Play("sad");
        }
        GUI.DragWindow(new Rect(0, 0, 10000, 10000));
}
```

This display could not be present at all times so in order to display this UI when the user selects the character the Emotions.cs activates its On-Mouse down method. This method detects if the character has been selected. This method will also hid the UI when selected again. Using a Boolean pop.enabled() we can set the GUIpopUp script to display or disappear.

```
void OnMouseDown()
{
        if (counter == counter1)
        {
        pop.enabled = false
        counter = 1;
        counter1 = 5;
        }
        else if (counter != counter1)
        {
        pop.enabled = true;
        counter = 5;
        counter1 = 5;
        }
}
```

Chapter 4

System Design

Designing of the system would prove to be one of the more difficult aspects associated with this type of application. It was important we designed the system at its most basic level, with audio cues, as little text as possible and without incorporating complicated touch gestures. When designing applications previously before taking on this type of project we could assume that the users of our applications could read and write to a basic degree and possessed core motor movement. This allowed us to create systems that could incorporate text heavy user interfaces for components such as navigation or displaying information to the user. It also allowed for us to create software for every type of device whether the software was used on a phone, tablet, or a computer as we are to assume that our users had core motor movement which in turn allows a user to use input devices such as keyboards, a mouse, or touch input on touch screen devices. From the research found in my literature review of how technology can assist those with autistic spectrum disorders (ASDs) I have found that autism can affect a child or a person in a number of different ways, therefore the designing of this particular type of software would need to adhere to Tom specifically, although those who possess similarities with Tom would also be able to benefit from this type of application. Taking in to account the research already performed by David and further discussion with Tom's parents we had to take in account the following when redesigning the system: .

- 1.Tom and autistic people alike tend to stick to particular routines, and deviation from these routines can result in behavioural issues.
- 2.Tom falls under the autistic spectrum where using input devices such as a mouse and keyboard is problematic due to motor control issues. Therefore input should consist of basic touch movements harnessed by touchscreen devices such as phones and tablets.

- 3.Tom does not read so any text should be accompanied by an aural cue for him to identify with the part of the application he is interacting with.
- 4.Interaction with a computer system resulted in a more positive experience for the user when they were visually represented in the application.

Platform Selection

As David has previously created a prototype using the Windows platform in mind, making use of Visual Studio and Microsoft Blend to handle the implementation of the project and the animations. Although the application prototype was successful in conveying the overall idea of what the user would like the application to visually look like, and possessed the functionality required with the application we were tasked with taking the prototype and making it cross platform for use with Android and iOS devices particularly. As David has previously created a prototype and had spent numerous hours building the porotype for Windows devices we were also tasked with preserving as much as the previous project as possible. With this in mind we originally tried to make use of technologies that offered a means of taking a build of a project and converting it to the other platforms such as Xamarin or PhoneGap. However after trying to reconstruct the project using these platforms almost none of the prototype could be reused and targeted for another platform other than Windows. After discussing the issues porting the prototype to another platform with Michael, we eventually decided the best course of action would be to take the original design and functionality of the prototype and reconstruct it using the Unity 2D-3D gaming platform. Although Unity is originally used for creating games it is not uncommonly used for creating "Storybook" style applications. We knew initially using Unity would eventually have drawbacks as we needed specific functionality such as a file picker and a way to save the scenes contents, which are not built in automatically. However the pros outweighed the cons as Unity promotes the following aspects needed for the project:

- 1.Unity promotes cross platform development.
- 2.Unity handles animations.
- 3.Unity can make use of sound clips.
- 4.Unity can handle touch input.

- 5.Unity can make use of a camera that can be used to follow the character in the scene.
- 6.Unity can make use of on an on screen graphic user interface that will always overlay onto the camera in the scene.

As Unity provided all of these necessary essentials needed for the project we set out to design the application.

Design Document

- 1.The initial design would need to be simplistic and menu systems should not be text heavy and over complicated.
- 2.Where possible audios cues would need to be implemented especially when selecting buttons and menu components.
- 3.A canvas system would need to hold all of the objects in the scene.
- 4.Elements would need to be loaded into a sidebar to allow the user to add as many objects as they see fit to the screen.
- 5.All of the objects should load into the scene from the menu and can then be positioned by the user.
- 6.Simple predefined animations should be in place for the characters to perform such as; “I’m hungry” and “I’m upset” for example.
- 7.The user should be able to change the background setting of the scene.
- 8.The user will need to be able to add their own pictures to the projects resources.
- 9.The user should be able to add multiple scenes to a given story so that they can show all of the events in a single day.
- 10.The user needs to be able to save the contents of the scene.

Figure 4.1

Creating the Scene Editor

The scene editor is where the bulk of the projects core functionality needs to be placed. Initially a default scene created in Unity is composed of only one main component the "MainCamera" with this in mind we are free to develop the scene editor how we saw fit. With this in mind we started to build the application from the ground up, knowing that we would need a particularly large area for the scene editor we decided set up the main camera first as with every Unity scene you are not limited to the screen size as the camera can be scripted to move around the scene freely. Initially we added a preloaded background to the scene to test the position of the camera and how the scene would look through the cameras perspective. Once the cameras values were set we had a basis for how the scene would look.

Creating the Player Game Object

After creating a temporary scene we decided to implement a player object which would eventually hold the animations, sounds and movement of the person controlling the scene. Making use of Unity's ability to handle animations we firstly set out to make the player object walk around the scene. Although creating moving characters seems like a simple task a lot of work

actually goes into creating a moving character in Unity due to the fact Unity is a gaming platform and therefore game objects in a scene are subject to the physics that are implemented in a created scene in Unity. The player character would need to have the following objects attached to walk around the scene with no limitations:

- 1.A "Transform" component that handles the positioning of the character in the X, Y and Z axis. If the Z axis is set to a value lower than the cameras perspective the player would not be visible in the scene. This component also sets the overall size of the asset.
- 2.A "Sprite Renderer" component, this component simply adds a default image to act as the visual representation of the game object. In this case a body, arms, legs and a head.
- 3.An "Animator" controller. The animator controller is of significant importance as the animated component attached to the game object will handle all of the conditions and requirements the game object will need to satisfy for the game object to perform an animation. For example if the game object exceeds a speed of 0.3, initialise the walking animation.
- 4."Colliders" will also need to be added to the game object to handle any collisions with other game objects in the scene. Although the player will not be able to interact with assets brought in by the user colliders would still need to be placed on the player to stop the player game object falling through the scene on instantiation.
- 5.A "RigidBody2D" component also needed to be added to the character as we would need to be able to freeze the position of the game object to only be able to move in the X and Y axis, and therefore not have the ability to move out of the cameras perspective.

Scripting the Player Game Object

Once the game object had the correct components applied, the next step was to add a script that would handle the movement of the player, firstly through keyboard input (for testing purposes) and then finally through touch input. We set up the script to make use of natural physics that the game object would be subjected to, therefore by setting a parameter that would change the speed of the game object when the object is moving we were able to set it so the animated controller would commence and the animation for the

Figure 4.2

player would change from idle to walking. We also implemented a method that would "flip" the game object should the user decide the player needs to move in the left direction. This method would simple change the values of the game objects "Transform" component, giving the appearance that the player character had faced the opposite direction.

Creating the Scene Panel

As one of the core features of this project is for the user to be able to bring in their own assets, changing the background image of the scene is therefore a necessity. Therefore we decided that one of the first options the user should have to interact with, is a way to change the background image of the scene. We had many different routes we could have chosen for changing the background however we thought that incorporating a canvas overlay that appears on top of the scene displaying the images located in the "Background" file of the application would be more suitable, as Tom relies heavily on visual representations of the things he would like to interact with. We started by using a simple canvas game object that we set to overlay on top of the cameras perspective, once that component was added we progressed to loading in just one image on to the canvas, and made the image available for selection. As we already had a background game object implemented into the scene we set the game object to change the image when the image was selected. However only one image at a time could be loaded in using this type of canvas functionality, and as the user adds additional background images to the application we need to display the entire contents of the background folder. After researching how to add more elements to a canvas component we came

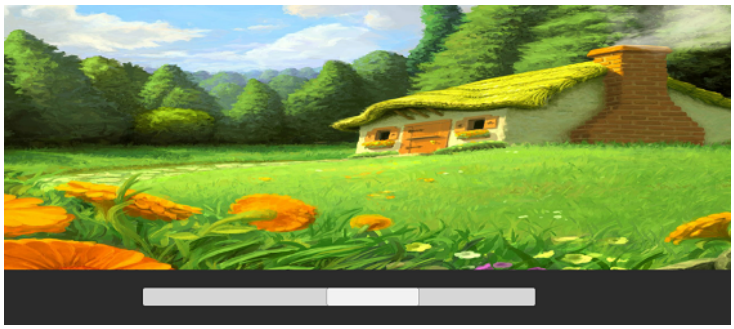

Figure 4.3

Figure 4.4

across a way to load in additional items into a canvas using an image panel and a scrolling bar, each image acts as it's on component and therefore tapping on one of the images in the selector would therefore add the image to the background component.

Asset import panel

Once the background has been set in the scene selector, the user will now have full control and will be able to edit the scene how they see fit. Here lies one of the more challenging components of creating the scene editor which was bringing in the assets of the user. As this application relies heavily on the application being able to make use of the assets imported by the user in this case Tom. We had managed to find a plugin that could handle the importation of the assets into the application with a Unity file picker plugin, however bringing in the assets into the scene editor was a he challenge in itself. To make this possible a folder had to be created within Unity known as the "Resource" folder, the resource folder will be populated with assets and

any assets imported into the project would eventually have to be placed in the resource folder for the application to recognise an imported asset. After creating the resource folder the next step was to create a user interface that handled selecting the particular assets found in the Unity resource folder. To do this we created a side panel attached to the right hand side of the scene editor. We could find no other solution than adding a button that activated the side panel, as without it the side panel would always be active and would therefore take up a lot of real estate on the scene editor panel. The button would be placed in the top left hand side of the screen and would trigger an animation that caused the side panel to move out of the cameras perspective, and subsequently into the cameras perspective. We are to assume that Tom initially will not be able to understand how to bring in the assets for the scene, therefore we needed to place audio cues to trigger when the scene editor had become active and when Tom would like to add his assets to the scene. Initially we populated the resource folder with just one folder that contained a dense amount of assets so we decided to split the resource folder into multiple folders depending on what the user wanted to add. As adding faces to the scene was one of the driving factors into making the application relatable to the user we split the resource folder into "faces" and "standard" folders. This was a small touch to the project that made it a little more user friendly. The side panel would also need to showcase the type of assets located in each of the folders so we set the side panel to use a picture that would hopefully depict what type of asset the folders contained. We decided to use a picture of a cartoon style face to depict that selecting this option would lead to the folder containing the imported faces assets. We used a house picture to convey that this folder would contain an assortment of objects. Once the asset folder was selected a second panel would active and load in all of the contents of that particular folder in the resource folder in Unity. There was an option to add a specific amount of assets to the panel however in theory as the resource folder would be never ending, we decided putting a limiter on the amount of assets loading in would be a hindrance to the creativity and possibilities the scene editor would eventually have. Therefore we added a vertical scroll bar (much like the one used for the background selection menu) to the panel that would cycle through all of assets. Once the user taps on an asset we needed to add that particular asset to the scene. To do this e made use of the Resource.Load methods that Unity provides. Once the user had selected a method we called the Resource.Load function to instantiate the asset into the scene editor. Difficulties arose when we tried to add anything more complicated than pictures such as advanced game objects which use box colliders, rigid body's and animations, however as the application is used to purely make use of picture elements we decided to stick with just bring in

pictures. Once the picture was selected we simple mapped the image to a game object and instantiated the asset into a position located in the middle of the scene editor. From there the user can freely place the asset wherever they see fit.

Chapter 5

System Evaluation

As many pages as needed.

- Prove that your software is robust. How? Testing etc.
- Use performance benchmarks (space and time) if algorithmic.
- Measure the outcomes / outputs of your system / software against the objectives from the Introduction.
- Highlight any limitations or opportuni-ties in your approach or technologies used.

Testing using Unity Editor

The softwares speed could not be tested by speed of processing of the elements in the application. Instead the application was tested at first using Unity's editor system. The robustness of the software proved solid when tested in the editors conditions. To Test it the application was used by persons who had no prior knowledge of the application and were asked to evaluate the simplicity of the functions and to see if they could crash the the application.

Test 1 Results Bugs.

- Loading too many assets crowd the scene and block functions [Fig 5.1].
- animations cause app crash
- Saving file saves to a Resource/ImportedImages/

Test 1 Results Fixes:

- Saved assets display

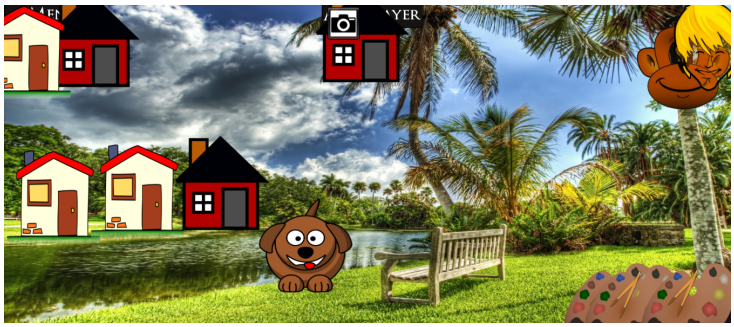

Figure 5.1

- assets enter scene on click
- Page navigation and general application structure solid.

These initial tests were carried out to ensure the robustness of the basic functions of the application that would later be important when we required added functionality such as scene save and loading assets. Keeping in mind that the app must be designed for multiple devices we had to ensure that as much code as possible is usable on multiple platforms.

Testing Load/Save from local device

The file picker GUI allowed us to access and save files from the local device to an in game folder called Resource. Despite this method working on the Unity Editor testing it was not possible when tested on Android devices. The problem that arroused after testing is that the Resource folder is compressed and when the application is built for device and becomes inaccesable. After extensive research we did not find much help in overcoming this problem. To overcome this problem we decided to take advantage of the persistantDataPath which is a folder that is created when the application is installed on all devices. This folder allows us a comman location in all devices to store and access new assets. Using the persistantDataPath with the FilePicker GUI allowed us to navigate the device and save to this folder. Another issue that was found when installing the aplication on an Android device was the access rights when accessing images on the local device. It was neccesary to force install the application to external sdCard in order for us to recieve access to these files. We were unsure wether this method of accessing and copying files into an application folder was allowed so we had to first check if it would be allowed. Despite no definate answer, there does not seem to be any problems

using this file path as there are built in methods to access this file and it is a commonly used file for storing application information post build.

Testing Builds on Android Device

Once we repaired the bugs that were found when testing on the Unity Editor we started testing the application on Android devices.

Test 2 Results Bugs.

- Canvas position and scalling correctly
- Theme images scale to large and block functions
- Navigating File Hierarchy possible too complicated

Test 2 Results Fixes:

- Pre-set data path for asset import so it opens directly to the DCIM/Camera location.
- Canvas and Assets were scaled to suit device. This is to be done for each device build.

Target Achievements

The ability to use images on the users local device was one of the original requirments for this application. The File Picker that we have emplimented can be used on iOS, Android and Windows. Taking advantage of the persistantDataPath location that the application adds on installation we were able to keep it the same on all platforms. After testing the feature has proved rebust and reliable. It is receseary to change the file start location for an iOS and Android build as their file hierarchy's are different. The search GUI is easy to use and works all common file browser windows. This feature could be improved be adding a more specific starting path location to reduce the chances of the user not understanding how to use it.

The SaveScene feature currently takes a screen shot of the scene and saves to a scene that displays all of the screenshots. This feature also takes advantage of the persistantDataPath where the picture is saved to and loaded from on request. The final build will contain a recording feature that will take values of each person character and save its movements so as to be played again on request. This feature is currently delayed to small bugs in the touch to walk feature.

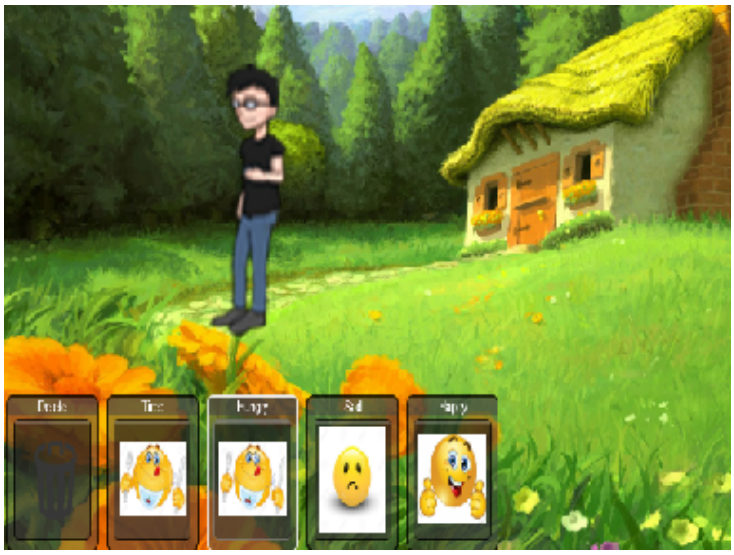

Figure 5.2

The animations that have been implemented allow the user to communicate emotions such as happy, sad and hungry.[5.2] the Animations GUI have proven reliable under all testing and have given us no problems. The structure of their design is reliable and the click activatation allows the user to hide and display them on request.

The application has been built in Unity so the development tool is designed to build for both android and iOS which are the main platforms that were requested for a build. Due to licensing a iOS developers licence is needed to put the application on the Apps Store. With windows and Android functionality working there is no foreseeable problems with an iOS build.

Chapter 6

Conclusion

The design and implementation of an application such as this required a lot of theoretical thinking outside of actual coding. Designing something that would be easy to use from the very first time a user access the application was important. We found that coding and implementing without thinking a number of steps ahead in the design would cost us time and resources. The user interaction with assets is obvious an does not require excessive user prompting. Voice prompts for the less obvious controls allows the user to access voice prompts. Using these click events the user can learn the basic functions of the application.

Having the ability to load images into the scene panel for the option to add them to the scene is important for personalising the application for each user. The access rights for Android and iOS mean that accessing images locally on devices is very tricky when we need to allow the user to select the images they would like. We found that to overcome these obstacles we needed to make sure the user installs the application to external sdCard. This prompt can be set when building the application in unity by changing the player settings in the build to preferred install to external. By doing this the application is granted access to the users local DCIM folder. The original idea was to find the images location on the device and load that file when the application would start to give the impression that the assets were in fact in the game. We found that this caused a problem if the image was deleted or the file name was changed. That is why saving a copy of the image to the persistantDataPath file location means the user will not loose and images if a picture is accedintally deleted. Having access to this path also allowed us to use this location for storing and retrieving SceneSave files.

The animations that are attached to the users characters allowing the user to express emotions are very simple and the images attached to them along with a text and vocal prompter leaves little room for error fr the user

when using them. The original idea was to have the character speaking along with text that is loaded with the animation. This was not possible due to the request to add personal assets as the head of a character. As the asset is imported from the local folder it cannot be animated on the fly to a sufficient level.

It was important that the application a high level of realism and did not become a game application. We did our best to use as little cartoon type assets as possible in the final build. Assets such as backgrounds will be pictures of areas that the client is familiar with so pre-loaded assets must be aesthetically similar to a certain degree. The hope was to have a all assets that are pre-loaded would be realistic as possible. It was not possible to have all realistic assets as we found that assets would not suit well with the picture backgrounds.

This project has been a huge learning experience on many levels. On the one hand it gave us a great insight into developing for a very specific user. We had to take into account many factors including the users abilities and understanding of modern IT. This forced us to adapt all the features to suit the client rather than develop something that we would like to use. Every feature had to have an aim. It had to provide a median to be used to help communicate. Another aspect of this learning experience is the lack of information available for some of the features. This forced us to come up with original ways to overcome obstacles. Features such as the FilePicker/FileSave made us think outside the box for new ideas and ways to solve these problems.

The research that was done for this project is important both for other developers who are designing similar applications and anyone who is interested in how technology can be used in education to help bridge any communication issues. The image manipulation using both the resizing and importing functions would be very helpful for any other developers. There is very little relevant help for these features online at the time of writing this. Matching the UNIFileBrowswer GUI with the persistantDataPath is a very useful method that could be of great benefit to developers looking to add image importing to their cross platform applications. The research that was done for this project include a literature review which detailed how technology can be used to help people with Autism in education.

Bibliography

[1] "Unifile browser gui."

[2] S. Horakeri, "Animation: Create level select scroll menu - unity 4.6."

[3] "Create a scrolling function."