Journal

# 15/10/12

I attended a planning poker session with my project supervisor and the other students who are using Kinect in their project. This helped a lot with identifying oversights in my user scenarios and in narrowing down time estimates for the user scenarios.

# 18/10/12 & 19/10/12

I set up all of my stories and sub-tasks on Jira. I could not set up a sprint as I do not have administration privileges. I have emailed my supervisor in relation to this but will work around it, for now. I downloaded and installed all the required software for development with the Kinect and Xbox360.

# 20/10/12

I went about setting up my Microsoft account for developing for the Xbox360. I could not register my account due to a glitch in the system. I submitted a support ticket containing all the required details and the reproducible steps I took. I shall develop for Windows until this issue is resolved as I cannot deploy Xbox360 projects without this registration. This has and will cause severe delays, due to the loss of time today and the extra time it will take to port the game to the Xbox360 once this is resolved.



# 21/10/12

I created a Windows game to work on while I’m waiting on a response from the Xbox360 developer support team. I implemented the Farseer physics engine to update on-screen sprites, using keyboard input to apply forces. The engine turned out to be easier to implement than I expected. It contains far more helper functions and classes than the average Box2D port. This has helped recuperate some of the time lost yesterday.



I created the input manager to place a layer of abstraction between the game and the input device. This will allow me to debug while the Kinect is not present/available. This will also make the game easier to port to different platforms, should the opportunity arise. I have skeleton code in place for a number of input types but have only fully implemented keyboard input so far.

I researched what was contained in the Kinect SDK API and what functionality I would require from this. I read up on these features to get to grips with what I would be using. I also added Kinect device support to the project. This does not perform input yet but the device is supported and managed correctly.

## 23/10/12

Read up more on implementing polling to retrieve data from the Kinect sensor as this is better suited to games than event driven frame requests. I implemented this on the skeleton stream to track joint positions for any active skeleton in the field of view of the camera. The resources I used for reference here were the “Developer Toolkit Browser” and multiple articles from the msdn Channel 9 Coding4Fun site.

Some of the potential issues which I have identified from this are inaccurate readings for the positions of joints which are not fully visible and trying to identify which skeleton is the correct player skeleton.

## 25/10/2012

Today I started to implement state management logic for menu updating. I also started adding gesture tracking to the project. I used the project at <http://kinecttoolbox.codeplex.com/> as a guide for handling gesture tracking. This project was built for the use with Windows Presentation Foundation so I have had to port the gesture management to suit XNA better. I have only implemented a base gesture detector class so far. I intend to implement a gesture fully over the weekend.

## 26/10/2012

Today I implemented a gesture manager and basic swipe gestures. I used the project at <http://kinecttoolbox.codeplex.com/> as a guide again. I experienced a few minor issues porting the event based code to XNA. I implemented the gesture manager class to deal with the events and allow for simple frame based polling from my input manager.

## 27/10/2012

I changed the structure of the gesture detector and gesture manager again to get rid of the last of the event based logic. I edited my sprite class so each object now handles its own loading. I separated the file paths for images from the main game class into a static class of constants to isolate the hard typed strings from the main game logic.

I implemented displaying of the user's hand active position on screen and the querying of this position for menu selection.

I ran into a number of issues implementing this:

1. I had to separate the displayed hand position from that of the skeleton being tracked so it could be scaled across the screen, otherwise the user could only reach a small portion of the screen.
   1. The original skeletal hand position is then only used to trigger the gestures for selection.
   2. The separated screen hand position is then used when querying what position the user selected.
2. The position of the users hand needed to be offset by the shoulder position to centre it on the screen.
3. The position of the screen hand then needed to be scaled dynamically so that the user could reach all of the edges of the screen.



I researched speech recognition using the Microsoft Speech Platform SDK and implemented basic voice commands in the project. I used “The Purple Book” (<http://channel9.msdn.com/coding4fun/kinect/The-Purple-Book-Using-Kinect-for-Windows-with-XNA>) and the “SpeechBasics-WPF” tutorial, from the Kinect developer toolkit browser, as points of reference when implementing this. I have only added basic voice control so far so have not come across any major problems.

## 28/10/2012

I implemented most of the required menu items and basic menu logic. This is taking a lot longer than I originally estimated. This is mostly due to the additional time required to ensure that both voice and touch input are catered for.

## 29/10/2012

I implemented the rest of the core of the menu system, created a few menu icons and created the main menu. I got to test the menu system also. Everything implemented so far is in order.



I implemented and tested both gesture and voice control in the menu system and implemented voice control to pause the game. This has taken less time to complete than originally estimated. This is mostly due to the extra time I spent on implementing the menu system correctly over the last two days.

Note: In the following video any actions which are happening without any on screen input are happening as a result of voice commands.



## 30/10/2012

I implemented game pad and touch input management logic in the input manager. I also refined some of the gestures which I implemented and refined the implemented menus some more.

## 31/10/2012

I went about building a stand-alone installer for the game but it did include all of my dependencies so it failed. I researched how to create “bootstraper” packages to add custom dependencies to a Visual Studio project on msdn (<http://msdn.microsoft.com/en-us/library/ms165429.aspx>). I also had to research how to get the GUIDs of installed applications so I could set the installer to bypass elements if they are already installed. I found out how to do this using PowerShell here (<http://elmaskubilay.blogspot.ie/2012/06/find-guid-globally-unique-identifier-of.html>). I then created bootstrapper packages for both the Kinect runtime and the Microsoft Speech runtime. This required a few trials to get the settings correct.

Once I got the installer working, I ran into another problem. The installed game would run fine if the Kinect sensor was not connected but on running it with the sensor connected the game crashed on initialization with no exception thrown. This was not reproducible on my own machine. To pinpoint where the error was I wrote a basic logger and output the initialization steps which were being hit. I pinpointed the issue to where the Kinect audio source was being started. I then output all of the relevant Kinect properties, only to find the status flag as “device unsupported”. The problem was that the Kinect SDK allows you to use the Xbox360 sensor but the Kinect Runtime only allows the use of a Kinect for Windows sensor. To fix this issue I added logic to check if the device is supported then fall back on keyboard input if it is not.

## 2/11/12

I implemented a vertical swipe gesture. I added logic to the gesture manager class to track both feet, using the vertical swipe gesture detector and detect if the user is running or jumping. This is done by logging the leg which last rose (via a swipe up gesture on an ankle joint) and the time. Dependant on the time between leg rises the action is detected as a jump action, a run action or ignored.

I also implemented the crouch and stand gestures. I based these on the vertical swipe gesture detector as well. The head is tracked in this case instead.

I then tested and tweaked these gestures by moving an in game object and changing the values settings to suit.

## 3/11/12

I altered the vertical swipe gesture detector with the ability to switch on and off detection of up or down gestures. This is useful for detecting jumping and running as I only wanted to track the up swipe for these and also saves compute time as it decreases the number of gesture entry searches.

I also created the basis of the stick man class. I am implementing the physics for this using three bodies and the method described here (<http://amazingretardo.simiansoftwerks.com/2010/02/17/platformer-character-control-farseer-physics-engine/>) for movement of the character.

This should allow the player to move freely over any terrain and allow me to switch on/off collisions with the different parts of the player based on their pose (standing, crouching, etc.)

## 04/11/12

I fixed the menu system so it is easily navigable using the keyboard or game pad. I swapped all the DateTime.Now calls in the project with DateTime.UtcNow calls as they are far less expensive. If you want to know more about this read the following; <http://www.eggheadcafe.com/tutorials/csharp/71b57428-6b59-4466-9762-ecb437ffac98/is-twitter-good-for-developers--and-datetimenow.aspx>, or <http://jason-mitchell.com/xna/xna-and-c-calling-datetime-now-is-expensive/>.