****

*Andrew Davis*

Evaluation Report

I am very happy with the overall outcome of my game. I achieved almost everything that I set out to include in my game.

One thing I did not include was the Mini-Game for generating popcorn (a currency in the game) to send to Facebook friends. I said in the original Project Plan that this was going to be an additional feature and would only be included if time permitted – so this did not affect me hugely. Instead of this feature, I added a simple button to send a 1 popcorn to the friend. I feel that this is a justified substitution to make as it took a lot less time.

As well as not having time to implement that optional feature, I also changed the primary target of the App from iPhone to iPad. This did not have a big impact on the overall game as the layout is consistent for the two platforms. However, in terms of testing, I found it easier to test the game on the iPad rather than the iPhone, purely because the larger screen size made it easier to see if something was wrong.

In the original plan, I specified that the main platform would be iOS, with the option for expanding to Android if possible. Although it was not possible to also run my game on Android during this project (again, due to time constraints), I have found that it is very easy to ‘port’ the game to different platforms, meaning that I could easily make an Android (and even a PC) version of the game if needed. This idea is also helped by my Facebook connection feature, which stores the player’s data in a database and can be loaded into the game. So the data is consistent across all platforms.

There were also several smaller additions that I would have liked to have made but I ran out of time. These features included additional jobs for the staff members to do and a ‘Premiere’ feature in which the player’s cinema would play host to celebrities for a night time special “bonus” level. It came down to a decision between implementing these additional small features, and attempting to implement Facebook integration. In the end I think I made the right decision as I got almost everything I wanted from the Facebook integration to work and I feel that this is a good feature to have as it improves connectivity between players. However, I plan to continue working on my game and ideally release it on to the iOS App Store so the features that I was unable to complete will likely be implemented before release.

I used an ‘Object Pool’ to generate and store the ‘Customer’ objects in my game. I found this feature massively improved the performance of the game. Before implementing it, the game started to ‘freeze up’ when too many customers were spawning – due to them being ‘Instantiated’ as and when they were needed. The Object Pool helped improve the performance as ‘Pre-Instantiated’ objects were stored so that the processing time-consuming ‘Instantiate()’ function (*Unity*) was run less often. The code of the actual Object Pool is in a Script called ‘*Object Pool*’ and the ‘*Time Controller*’ script accesses the pool to get objects for customers when they ‘arrive’ at the cinema. I also used the Singleton design pattern with the Object Pool.

Singleton involves a class storing a static instance of itself, so that data can be accessed from it without having to store a local instance of the class. Although I did not specify this pattern in the original plan, I found that it would be useful in some areas of my game, for the Object Pool and Facebook connectivity in particular. Evidence of the Singleton design pattern can be found in the ‘*Object Pool*’ and ‘*FBScript*’ scripts of my project.

I did, of course, include MVC in my game as well. Although I initially found it tricky to implement MVC in Unity, I think my final implementation of it is good. I have Models to store the details of each component of the game (Screens, Customers, Staff etc). These models are accessed by the various controllers of the game (*TimeController*, *StaffController* etc). Originally, I had one large controller which handled the majority of the functionality. However, with each new feature added, this controller grew and grew and I quickly realised that this approach was terribly inefficient and horrible practice! So I decided to split the controller into several ‘specialised’ controller – each controlling groups of related methods. Finally, the View part of MVC is managed by Unity’s GameObjects.

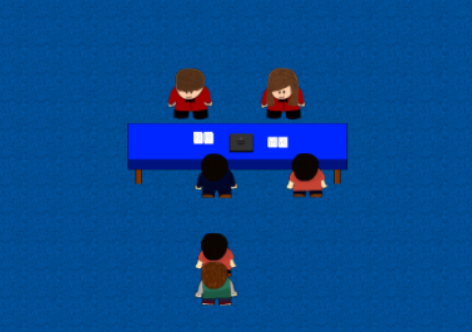
So for each action, the relevant Controller will perform any necessary calculations, then call the Model to update the necessary values (if any). Once these updates have been done, the Controller will call the method to update the relevant information on the screen (if needed). For example, when upgrading the level of a Screen, the Controller will:

1. Work out the new level of the screen, and perform other calculations such as getting the new capacity of the screen.
2. Update the necessary values in the relevant Screen model.
3. Change the sprite of the screen to match its new level.

MVC is evident is several areas of my code. The most obvious ones are for moving and updating customers (‘*Customer\_Controller’, ‘Customer’*) and purchasing items from the Shop (‘*ShopController’*).

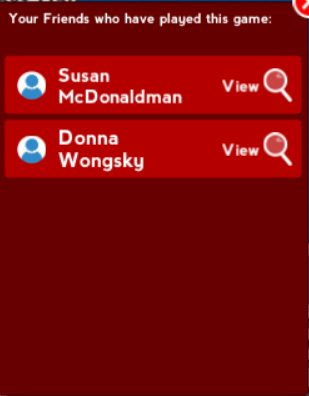
As well as using MVC for the main functionality of the game, I also used it will the Facebook integration feature of my game – both on the Client side and on the Server side. In the Unity project, the code is in the ‘*Login*’, ‘*UpdateDetails*’, and ‘*AddUser*’ classes. The Web API code (written in Java) is almost entirely in the MVC format. The View is the Game which calls the web servlet (Controller), which then calls the model to update/get details from the database. The retrieved data is then returned to the servlet which forwards it back to the game.

The ‘Update’ pattern is also used a lot in my game. I implemented this pattern using the Update() function in Unity, or the similar FixedUpdate() function on occasion. These functions will run once per frame so are ideal for moving objects on the screen or updating counts. The main implementation of them in my game is for moving customers around the cinema (‘*MovementScript*’) but it has also been used for various other elements of the game: from complicated things such as dragging the staff members to the desired ‘posts’ (‘*mouseDrag*’), to more straightforward things such as incrementing the time of day (‘*TimeController*’).

The Game aspect that is most prominent in my game is Concurrency. I have used this to good effect in the Queue system I created. Each staff member assigned to a queue runs its own thread (with varying delays based on their upgrade levels), meaning that several customers from the same queue can be served at the same time – at different times, with different delays. If a staff member is removed from the job/post, then the customer they were in the process of serving is returned to the head of the queue and the staff member’s thread is destroyed. The code for this can be found in the ‘*CustomerQueue*’ class. I am very happy with how this code works as it truly reflects how a queue would work in a real cinema.

Each staff member runs on a separate thread.

I also used concurrency when saving to the database. I found that the process of saving so much data to the database was a very slow one! And hence, when trying to run it in the same thread as the main gameplay, the entire game would freeze and wait for the save action to finish. This was obviously no good so I moved saving to a separate thread. This kept the main thread open for processing the game while the second thread carried out the save actions. The code for this is in ‘*MainController*’.

I also included online connectivity in my game – through Facebook integration. This feature means that the cinema data is stored in, and retrieved from, a database, meaning that the player can access their cinema on multiple devices using their Facebook login. The player can also view which of their Facebook friends have played the game – as well as being able to view their friends’ cinemas. They are not able to perform any actions in a cinema which is not their own (i.e. they can’t purchase any items, move staff etc) but they can send a gift of popcorn (a currency in the game as mentioned before) which can help their friend to improve the cinema. I am happy with the way the Facebook functionality works as it will increase competitiveness as friends compare their cinemas and compete to have the ‘best’ cinema. The code for the Facebook integration is in ‘*FBScript*’, as well as several areas for loading/saving in ‘*Controller*’. The Web API code a lso deals with Facebook integration.

List of friends who have played the game – and the option to view the Cinemas

AI – pathfinding – floor (image of customers going different ways)

Testing – tested on laptop – after each ‘feature’ then kept testing fixes until fully functional. Tested full game as I went – recorded bugs. Every few days (more frequently closer to deadline), put onto iPad – slow process. Found additional, iOS bugs = added them to list of bugs.

Do MVC from start with multiple controllers – save refactoring from one big controller