1 Overview

1.1 The Problem

The task we have been set is to create a market-driven online multiplayer game for last.fm, a popular online music site. This game should use their data (which includes wiki-like biographies for artists, listen counts of their userbase, events, new releases, and a plethora of other information about these musicians) as well as be designed in such a way as to be scalable (both functionally and in terms of gameplay) and integrate easily into their main website.

Generally speaking, market-driven online games can be split into two categories: those that emulate the stock market, and those that emulate football leagues. In the former, stock-market approach, users are assigned some amount of money at the beginning of the game which they invest into the various stocks on the market that they think will increase in value. These stocks then change prices (either according to some sort of in-game economy, or modelled on the actual stock market) and users buy and sell them in such a way as to make a profit. Football manager style games, on the other hand, make the users create a 'team' of famous footballers, whose 'value' fluctuates based on the real world performance of those players - the market based aspect comes from having the ability to trade players at given times, as well as in the prices of the actual players when they are sold/purchased.

Clearly, neither of the above approaches directly translates to a music and/or artist setting. Therefore, it is necessary to find a way to introduce markets into the context of music without making it feel forced, whilst ensuring that it is fun to play and the barrier to entry is reasonably low.

1.2 The Solution

The approach that the team decided on was to put the player into the position of a radio manager (or club manager), and then allowing them to buy the rights to play a certain artist on the radio (or in their club). The cost of the artist would be determined by two factors: firstly a 'base' price based on the various metrics from last.fm's database (such as unique listeners, loved tracks, and similar), and secondly according to the basic rules of supply and demand that are created in a restricted market (created by ensuring that only a limited number of people could have the rights to the same artist at a time).

There are a number of good points about this game idea. The first is that the roleplaying aspect of the game should keep it from being too dull to play - by adding in achievements and periodic rewards to the game for progressing through it, it should provide an incentive for people to continue playing (not to mention the potential integration with the user's normal last.fm account, if they have one previously). Moreover, by having a standard market the game economy should mostly balance itself, as long as there isn't a huge influx of players causing inflation, and there are sufficient money sinks to prevent this from happening. Finally, the game is open-ended enough that each individual player can have their own goal - whether it is to own a certain group of artists, or to own as much money as possible, or simply to have a large 'portfolio'.

2 Requirements

2.1 Functionality

Since the scope of the project is quite large, the below list is divided first by gameplay features, and secondly by technical requirements. There are also some 'would like' features listed (and marked as so). Starting with the gameplay features:

- The user should be able to log into the game using their last.fm account. If they don't have one of these accounts, then the game should redirect them to last.fm and prompt them to register there,

before redirecting them back to the game. The reasoning behind this is that it means that it would be easier for the users (since the majority of them would be coming from last.fm itself), and it would allow for their last.fm account data to be used for the game, such as recommending artists for the user to buy from their top played lists (although the latter is only a 'would like' feature).

- The game should have a list of artists available to buy and sell that the user can access. If possible, every artist in the last.fm database should be accessible on request, so that even people in other countries or with more obscure music tastes are still able to buy their favourite artists. Also, if possible, there should be ways of sorting these lists by genre, popularity or other metrics.
- Artists should each have a value associated with in the database, initially based on information from last.fm, and then influenced by the demand and supply of that artist in the market in the game. Optimally, this should be designed in a way such that the market is reasonably mobile and variable regardless of the number of users, since this would ensure that users continue to be engaged in the game.
- Each artist in the above list should have their own page, showing information about their historical performance - the number of people buying and selling it, as well as the value historically. If possible, similar artists should be shown on the artist page, as well as anything else that might be of interest to the user.
- Based on the above value, users should be allowed to buy and sell artists - if there are none available, then the users should be allowed to put in a 'future' bid for when more become available of the market. The process of buying and selling should mostly be a 'black box' for simplicity, with only basic data such as lowest price on the market currently being presented.
- There should be achievements for users - such as achieving a certain total networth, or buying a set of artists. This should go together with a leaderboard list, accessible and visible to all users, again to incentivise people to play the game and give them a sense of achievement as they progress in the game. Moreover, these leaderboards should be split into 'leagues', such that newer players do not feel alienated or unable to compete with those that have been playing the game for a long time, as well as allowing for competition between friends on last.fm.
- Each user should have their own profile page in the game - this should have their recent performance and transactions on it, as well as information on the artists that they have and potentially graphs showing their performance in the game over time. Moreover, their position in the various leaderboards should also be displayed and any other relevant information about that user.
- Outside of the profile page, the user should have access to a single page with their entire 'portfolio' on it, allowing for easy access to all of their current artists and allowing them to go to the profile pages of all of these artists. They should not be allowed to buy or sell from this page - the buying and selling should only occur, if possible, on a single page and only through that interface in the interests of consistency.
- The game should present the user with a settings page, which allows them to change any global options and to delete their account if they wish to do so.
- As an optional thing, if there is time then the game should have a tutorial system for new users which introduces them to the game and gives guidance on how to play it and what can done.

In terms of the technical requirements for the game:

- The game should use last.fm API as a core part of the system. This is part of the project specification, and it allows access to a vast amount of artist and user data as well.
- The game should be designed in such a way as to be easy to integrate with last.fm. This involves ensuring that the design of the website follows similar colour schemes and characteristics to the actual design of last.fm, and for the system to be designed in a modular and easy to use way.

- Following from the above, the entire system should be well documented, so that it could be potentially maintained and edited by someone outside of the team in the future.
- The system should be scalable as user numbers increase.
- The system should be secure from people attempting to abuse or cheat at the game, and be able to (within reason) prevent the most common methods of doing so.
- The system should be split into front-end and back-end, whilst providing an API which links the two together. If possible, this API should be made public to allow people to build their own user interfaces or methods of analysing the data from the game if they wish to do so.

2.2 Components

There are four major components in the system itself: the web interface, which is the only 'front-end' part of the project, and is what the user will see; the analytics backend, which pulls and manipulates the last.fm data; DATM, a module used for make data access easy and transparent; the API, which is used to join together the back-end database and the web interface of the game. They are described in more detail below:

2.2.1 Web Interface

This component of the system is, to some extent, self-explanatory. It consists of taking information from the game database using the API, then using this to generate the web pages that the user sees. Equally, it involves taking the requests of the users to do things within the game and passing these onto the API such that they're carried out in the game.

The most important thing for the user interface is that it presents all the features listed in the previous section to the user, hopefully in an easy to navigate and intuitive fashion - this includes all the artist pages, the ability to buy and sell stocks, leaderboards, etc. There are other considerations to be taken into account, such as the fact that the the UI should be fluid - any actions the user takes (such as clicking a 'buy' button) should have sort of instant feedback to the user. Moreover, the actual design of the website should (if possible) adapt to whatever device the user is currently using, regardless of screen resolution.

There are a number of possible pitfalls that need to be avoided as well. The first, and most important, is missing features, or features that are not obvious and easy to find - if the users cannot access or don't know that a part of the game exists, then essentially it is as if that part actually doesn't exist. Further to this, if the design is inconsistent from page to page then this could also cause issues - for example, if the ordering of the buttons in a menu changed between different identically looking pages, then clearly the users would get confused. Hence, consistency in the design must be considered as well.

2.2.2 API

The API is the part of the program which links together the 'back-end' and the 'front-end'. In other words, it is used as the channel through which data flows between the the web interface for the game, and the storage database which holds the game data. Functionally, the API should consist of two parts: a server, and a client library. The API server would be run on a remote server somewhere, whilst the API library would be provided to the web interface to allow them to call the commands that the API server allows. Optimally, the API server should use the DATM (explained below) to handle all of it's reading and writing from the database itself though.

There are a number of features that this API should support, the first being that it should provide all that is necessary to the web interface, for the interface to be able to implement the features listed under 'functionality' above. Specifically, this involves providing enough calls such that the web interface can both gather the data that it needs, and write back all the necessary results as well. Secondly, the API

should be able to handle a large number of concurrent requests (keeping scalability in mind), fail gracefully if it can't, and preferably be as stable as possible. Thirdly, it should also inherently support versioning, to avoid client/server mismatches if and when features are changed in the game.

A well-designed API is a must in order to ensure that the entire system works correctly. There are a number of potential problems: a badly designed API with unclear or excessive inputs and outputs would cause confusion for the web interface; a featureless API would mean that the frontend could not access the information it needs; a badly implemented API would introduce obscure and hard to identify bugs. Moreover, an undocumented API would be useless if other people were working on the codebase, hence of all the other parts of the project, the API must have the clearest documentation.

2.2.3 DATM

DATM is an acronym for 'Data Abstraction Transparency Module', and this library is a key component of the entire project. Essentially, it's goal is to provide a way for both the API and the analytics module (described below) to access both the game database and the last.fm API in an easy to use way, with the complexity of doing so being abstracted away. It is essentially a 'black box' for anyone that uses it.

Since it's role is so central to the functionality of the entire application, there are numerous requirements and guarantees that this module must provide. DATM must firstly be thread-safe, to allow for multiple instances of it to run such that concurrent access is possible in other parts of the program. It should ensure that writes and reads from the database are also carried out in a safe manner (free from all the common concurrency errors that can occur), whilst guaranteeing a reasonable level of performance so that the API (and consequently front end) are not waiting for database access.

2.2.4 Analytics

The analytics component of the program is tasked with the role of carrying out all of the calculations involving data from the last.fm website. This includes fetching the data in at a reasonable speed (through the DATM), processing it, and updating the database as required (again using the DATM). Since it is responsible for handling this data, this means that the analytics module is also at least partially responsible for ensuring game balance and stopping the markets from fluctuating too much or too little.

[Someone STILL needs to talk more about analytics here]

2.3 Acceptance Criteria

The acceptance criteria for the finished product are reasonably simple: [Again, I feel this is sparse and some editing would be nice]

- To produce a working game that can be run on a server somewhere and tested.
- To ensure that the above game has the functionality mentioned earlier on in this document.
- To ensure that the game has been properly tested and is reasonably free of bugs.

3 Management

The management strategy decided upon by the group is perhaps slightly unconventional, in that we chose not to have a group leader. The reasoning behind this was that having a single person in charge would mean that they would have perhaps too much control over the direction and concept of the game, leaving the ideas of others of the team ignored. Moreover, it is arguably far more effective to have every individual in the team making sure that everyone else knows what they should be working on, and are making progress on this, than to have a single person assigned to this task.

With regards to the technical responsibilities of each member, we decided not to have a hard assignation of people to different tasks, but rather to let them choose what they wished to do, and (within reason) let them switch between working on the different parts of the project - since some people were happy to work on anything, allowing people to do this would not come at the cost of some parts of the application never being written at all. In addition to this, certain people in the group had far more experience and knowledge in some areas than others did, which we decided would naturally lead to those more familiar with a certain area working in it, leading to (hopefully) a better implementation.

We also decided to use a versioning system (Git) to manage our codebase. This would allow multiple people to work on the code at the same time whilst maintaining a central repository where people would be able to see what others have done, and comment on and modify this code as necessary. It would also allow for the easy setting of milestones, the creation of individual branches for separate features, and rollback if seriously broken code was ever submitted.

For general communication, we decided on meeting at least twice a week in person (on Tuesdays and Thursdays during the allocated time), whilst communicating online far more often through various methods such as shared notepads, Google services (docs, hangouts), Facebook services (groups), IRC channels (also beneficial since the client provided us with a channel on their server that we could use) and a general email mailing list.