# Milestone 5

Super Smash Bros Brawl Management System

Team Shine: Seth RanChao Zhang, Trevor Krenz, and Richard Thai

16 February 2012

# Contents

# 1 Executive Summary

This document is intended to explain the Super Smash Bros Management System with the supplementation of a Relational Schema and an Entity Relationship Diagram . In addition, this document also contains description of the problem, a description of the solution (front-end and back-end), key challenges, and an analysis of the database design.

Super Smash Bros Brawl is Nintendo's best-selling video game in the history of Nintendo of America [1]. However, despite its immense popularity, information about the game is heavily-based on decentralized player research. Because of this, varying reputations of significant players are inconsistent or nonexistent across a multitude of communities. In addition to this, there is a glaring lack of any social networking aspect involved with the game. the solution that was created, directed by these key issues.

# 2 Introduction

This document is the final in a series that will describe the Super Smash Bros Brawl Management System. Also, this document is extended by a Relational Schema and an Entity Relationship Diagram. This document drives into the details of the problem at hand, the solution, the challenges that arose and how they were confronted, design of the database, and an analysis of the integrity of the design.

# 3 Problem Description

The release of the Nintendo Wii exclusive video game Super Smash Bros Brawl was received very well among fans of the previous games as well as newer players and even the casual player fan base. Despite this, there was a clear lack of certain features from the video game such as centralized leaderboards, social networking, and consistent game information (on characters/stages/etc) which is becoming an increasingly important factor in determining the success a series' sequels.

Ultimately, the goal of the management system is to solve these exact problems. A leaderboards system will be implemented by keeping track of match information such as the players involved, characters, and wins/losses. Also a friend system will be set in place where people will be able to look up players and add them as a friend–it should be noted that this is a one-way relationship; adding people as a friend does not make the relationship mutual, an aspect that we, as the designers, add for humor purposes.

# 4 Solution Description

The solution decided upon by Team Shine was to develop the desktop application with Micosoft's C# Programming Language (2) and use Microsoft SQL Server 2008 R2 (3) in order to maintain all of the information relevant to the management system.

## 4.1 Front-End Discussion

The front-end application was created in C# on the recommendation of a group member who commended the ease of use of C# (given our backgrounds in Java) as well as the GUI design and building capabilities of Visual Studio. In addition, the use of C# seemed like a logical step given that it is a Microsoft technology (which we assumed had implied an ease to integrate with the SQL database, an assumption we had found to be largely correct). Development of the front-end was developed with solving the problem in mind, which we took one step at a time. We created screens based on the order of interaction that we had scripted as a sequence of screens (i.e. work started with the login page, then the registration page, and finally the main page which would house the other pages). The final user interface was tested for usability under the developers, a scope that was found to be satisfiable given the extent of the project, and was ultimately found to be simple and satisfying to use.

## 4.2 Back-end Discussion

The client application interacts with the back-end via stored procedures. The username that the client uses to connect with the database is not allowed to do anything but connect to the database and execute a set of predefined stored procedures. This is to prevent any unauthorized and malicious code from being executed. Tables in the database contain usernames, passwords, match information, character, game and stage information, and friend information. The database exists on the Whale server of the Computer Science and Software Engineering Department of Rose-Hulman Institute of Technology. The database is likely to be removed after February of 2012. A more in-depth discussion of the back-end can be found in the Design Analysis portion of this document.

# 5 Key Challenges

## 5.1 Inexperience with the C# programming language

### 5.1.1 Solution

Have the person show us sample code on how GUI's are made, and leave any issue that the rest of us were having trouble with to him.

### 5.1.2 Analysis

This approach worked well. The only issue is that if this person could not make a meeting, it would mean that the meeting was pointless if the original point of the meeting was to develop the GUI. Additionally, the two of us not familiar with C# found the language to be similar to Java, and it was not hard to learn the essentials of the language. We found that the most difficult part of the project was actually trying to connect to the database.

## 5.2 Handling secure passwords

### 5.2.1 Solution

Have the client application hash the password (under SHA-512), and the back-end have a large enough datatype to accommodate the hashed password.

### 5.2.2 Analysis

Initially, we ignored password security as it was a lower priority than the other functionality of the system. When we decided to use hashed passwords, it made all of the player data obsolete because updating the passwords was complex, and we judged that wiping the match and player data for the hashed passwords was the option that took the least amount of time.

## 5.3 Character and stage images

### 5.3.1 Solution

Have the client deal with images rather than the back-end database.

### 5.3.2 Analysis

Storing images in the database seemed hard to do, so we the images are stored with the client. Images have the name of the character/stage that they represent. For example, the character Pikachu retrieves the image Pikachu.jpg.

# 6 Database Design

## 6.1 Stored Procedures

| Stored Procedure Name | Stored Procedure Purpose |
|---|---|
| **AddFriend** | Adds a user to the given user's friend list. A password for the given user must also be passed as a parameter for security reasons. |
| **AddMatchInfo** | Used by the game to input game match information. Information includes an ID, date, time, match type, and data for potentially up to 4 players: their name, the character that they selected, and the result. |
| **GetCharacterInfo** | Returns information about a character's tier, the game that they came from, and a brief description. |
| **GetCharacterList** | Returns a list of all character names in the database |
| **GetFriends** | Returns a list of usernames of the people that the given player friended |
| **GetMatchInfo** | Returns data for a match in which all given people have participated in. The returned value is a table in a format ready to be displayed by the client: match type, date, time, and up to 4 players: their username, character, and result |
| **GetPlayerInfo** | Returns the given player's username, win total, loss total, and their favorite character. |
| **GetStageInfo** | Returns the given stage's name, boundary size, the game it came from, and a brief description. |
| **GetStageList** | Returns a table with the names of all stages in the database |
| **LoginPlayer** | If the given player and password exists in the database, 1 is returned. Otherwise, and error is raised. |
| **RegisterPlayer** | If the given username does not exist in the database yet, the username is added with the given password. Wins and losses are set to 0, and no favorite character is selected by default. |
| **RemoveFriend** | Removes a player from the given user's friend list. A password for the given user must also be given for security reasons. |
| **SetFavoriteCharacter** | Sets the favorite character for the given user. The user's password must be passed in as a field for security reasons. |

### 6.2 Indexes

The only indexes used are the ones that are generated–by default–for the primary keys.

### 6.3 Triggers

There exists a single trigger in the database which ensures that there exist no two players with the same username with distinct capitalization. All users must have usernames with distinct characters, regardless of capitalization.

## 7 Design Analysis

### 7.1 Strengths

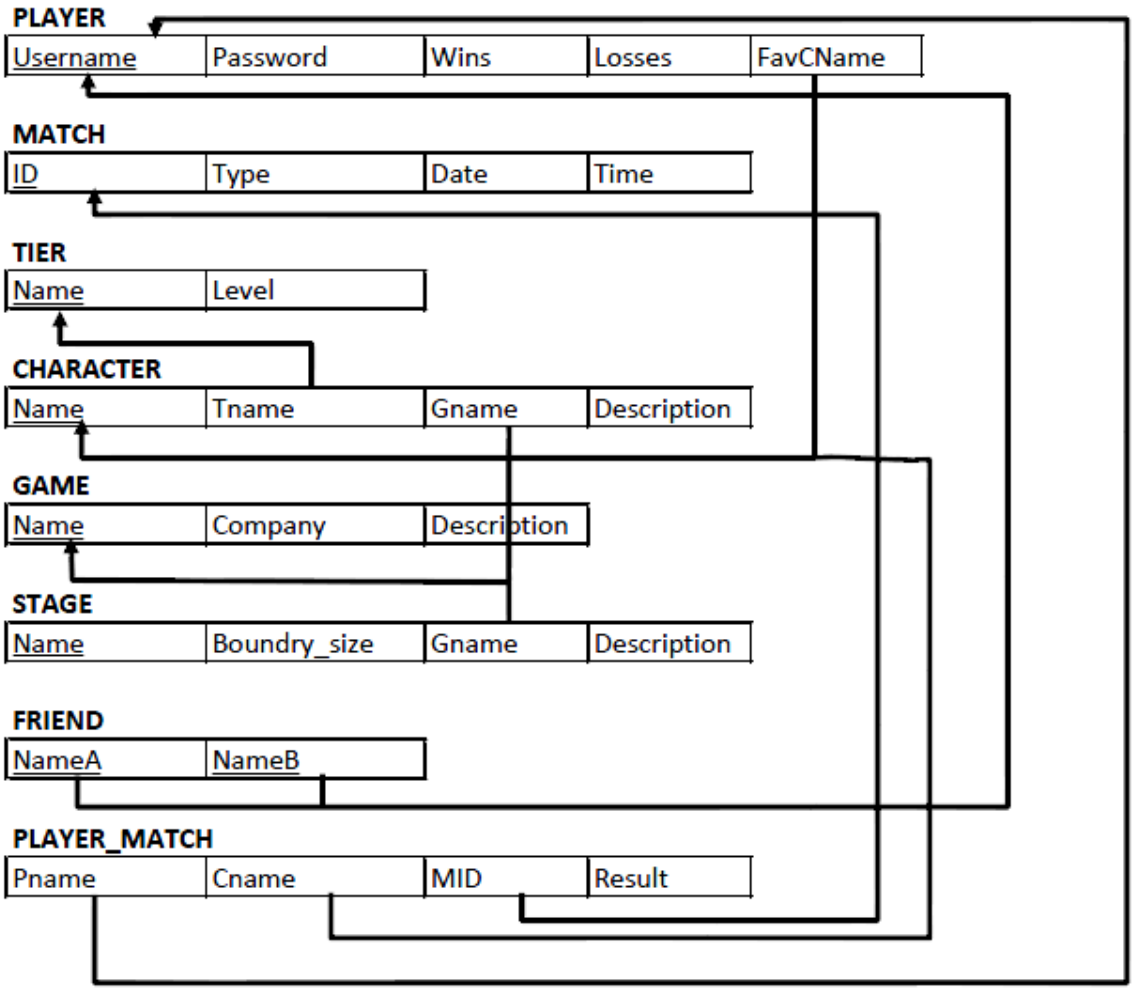
- The system achieves a high level of security by limiting the user that connects to the database to only run EXEC on the given stored procedures. This prevents people from running unauthorized SQL queries from 3rd party client programs. Also, this makes the system immune to SQL injections.
- The client hashes the passwords given to it. The password field in the player table is 256 characters long to accept the hashed password.
- Tables and attributes have intuitive names, with the exception of the Player_match table. The Player_match table was a result of normalizing the Match table so that it did not contain any null entries.
- When changing a player's data, the stored procedures require a password of that player. This is to ensure that one player does not modify another player's data if they were not to use the client and instead use the stored procedures.
- Stored procedures only examine primary keys of tables when executing, with the exception of passwords. This is to reduce the overhead of secondary indexes.
- Appropriate referential integrity constraints are in place.
- Raising errors in the stored procedures helps hide information about the database.

### 7.2 Weaknesses

- The 'LoginPlayer' stored procedure is susceptible to brute-force attacks for guessing a password. To mitigate this problem, the stored procedure could only be allowed to run for a given username a certain amount of times per minute.
- The increased security results in decreased flexibility. The user is limited to only executing stored procedures. If more flexibility is desired, more stored procedures can be added.
- There is duplicate information on the wins and losses of the player (they are recorded in both the Player table and the Player_match table). This is to increase performance at the cost of increasing redundancy. If the redundancy is undesired, the wins and losses column in the Player table could be removed, but this would impact performance because a player's wins or losses would have to be counted from the Player_match table.

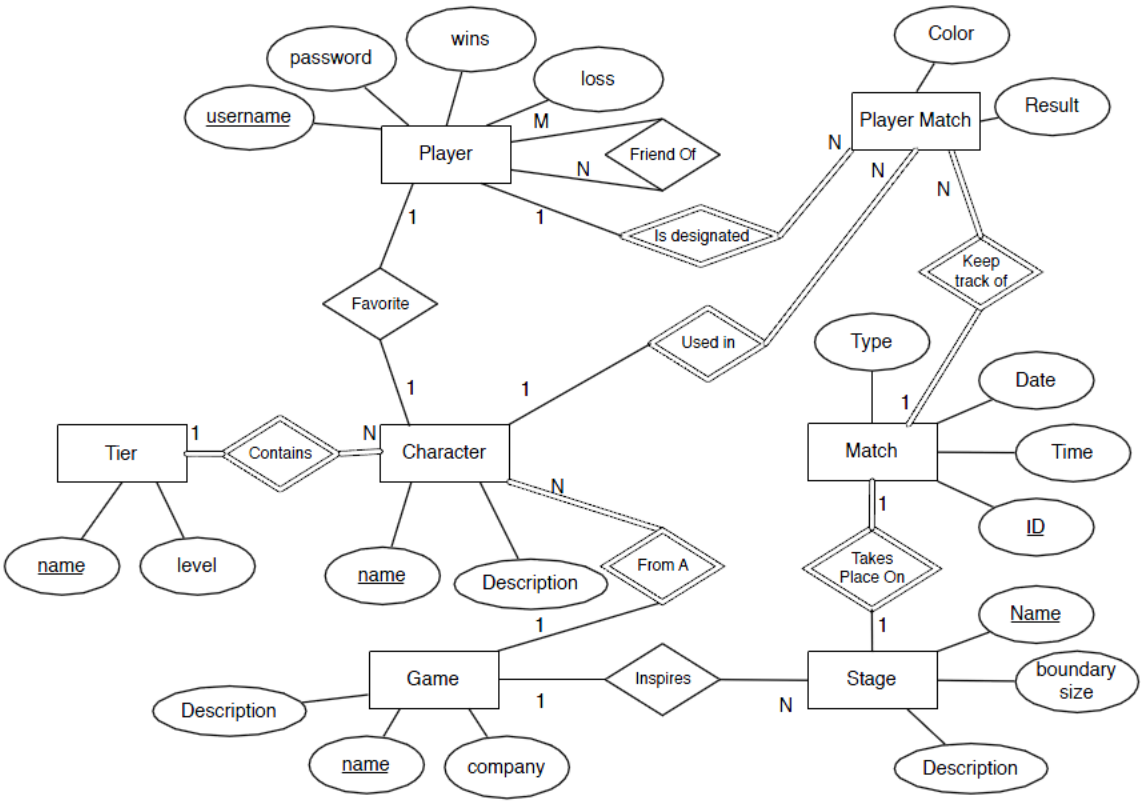
# 8 Appendix A

## 8.1 Relational Schema




Schema.png

### 8.2 Entity Relationship Diagram




### 8.3 Explanation of Entity Relationship Diagram

foo

## 9 Index and Glossary

**C#**: (pronounced 'see pound') multi-paradigm programming language that was developed by Microsoft. (3).

**Entity Relationship Diagram**: Abstract and conceptual representation of data. (3).

**Nintendo Wii**: Seventh generation video game console developed by the Nintendo company. (3).

**SSBB**: Super Smash Bros Brawl, a best-selling title for the Nintendo Wii console. (3).

**Visual Studio**: An integrated developer environment developed by Microsoft which is used to create windows applications. (3).

# 10 References


1 Super Smash Bros Brawl Smash Nintendo Sales Records. *http://www.nintendo.com/whatsnew/*.

2 Microsoft Visual C#. *http://msdn.microsoft.com/en-us/vstudio/hh388566*.

3 Microsoft SQL Server 2008. *http://www.microsoft.com/sqlserver/en/us/default.aspx*.