CS 368 Final Project
Overwatch Counter

**Team Members:**
Brett Abramczyk  l 907-009-8372 l babramczyk@wisc.edu
Richard Wollack l 906-967-8945 l rwollack@wisc.edu

## Introduction

The motivation for our project came from the game Overwatch and a scoring system that has been adopted online. Each Hero in the game has certain enemies that they counter and certain enemies that counter them. When choosing a team you want to be able to have the set of heroes that counter the enemy team most effectively in order to have an advantage during the game. One of the resources that was found online for rating heroes and their counters was organized into a spreadsheet that you would have to search across rows and columns manually, or start creating formulas and referencing different cells. We decided to build a C++ application to provide a better way of looking up the best counters for an enemy team.

When the user opens up the application they see a list of all possible heroes that is populated from a MySQL database. To build an enemy team the user selects heroes from a list and adds them to the enemy team with buttons and different keyboard commands. The enemy team can be added to and removed from at any time by using the buttons or keyboard commands underneath the enemy team container. A table is populated with counters and their composite score as heroes are added to the enemy team. The result table also shows an icon shaded in green for each specific hero from the enemy team that they counter, and an icon shaded in red for each hero that counters them.  After the team is fully assembled the list of counters is fully populated and sorted by their composite score.

## Design

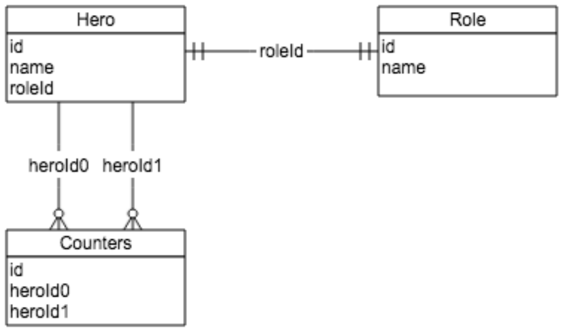

***Figure 1 - Database schema**

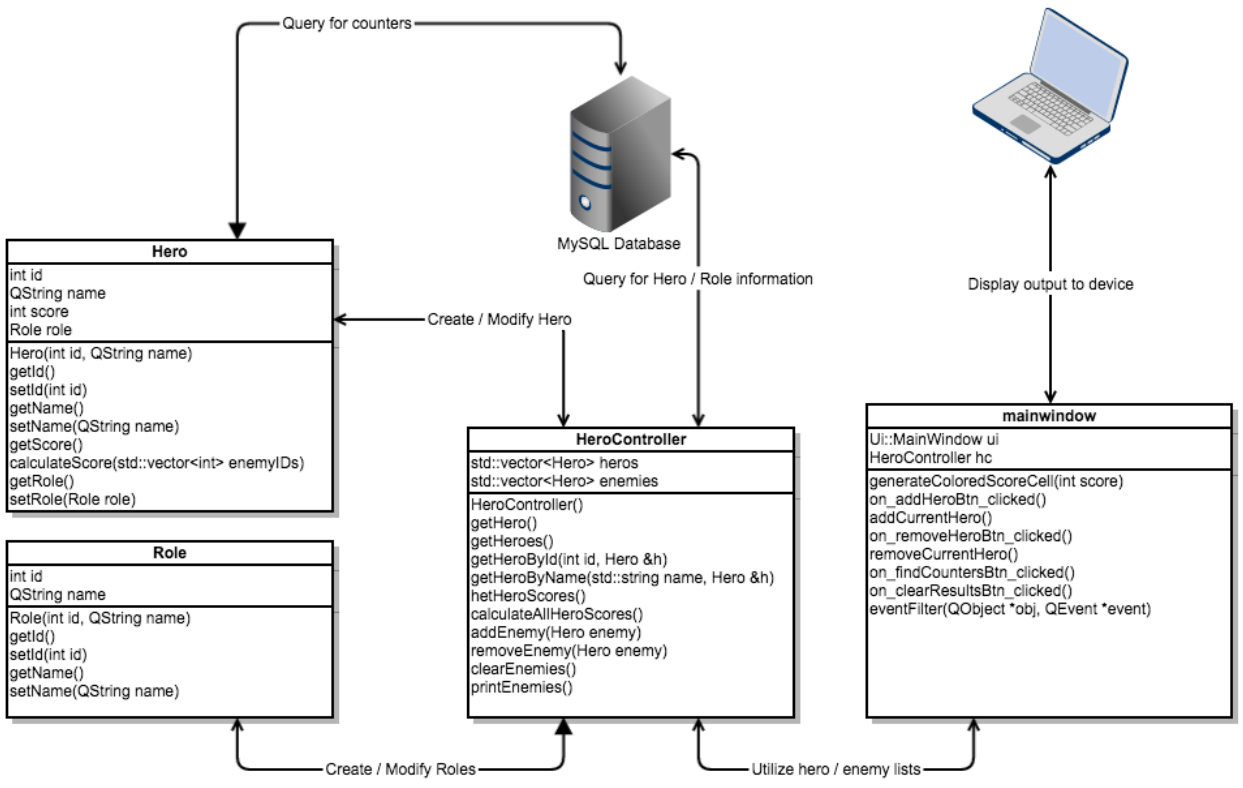

***Figure 2 - Application / Data Flow**

## Implementation

We decided to use MySQL to store the information about Heroes and their counters mainly because we were both comfortable with MySQL and we wanted to see how to utilize it in a C++ application. Neither of us had experience with a GUI framework for C++ so we decided to try out Qt for making the user interface. We were able to learn enough to build a useable interface for the purpose of our application.

The first thing we did was crate a database schema to use for storing information about heroes, roles and counters. The schema is depicted in Figure 1 above. We have 3 tables for storing data.

- Role - Stores an id and name.
- Hero - Stores an id, name and roleId. roleId is a foreign key on the id column in the Role table. Each hero can have 1 role.

- Counter - Stores an id, heroId0 and heroId1. heroId1 and heroId2 are foreign keys referencing the id column on the Hero table. The first key corresponds to the id of the hero that counters the hero of the second key.

We tried to separate the data from our view as much as possible, and there are 4 main classes that make up our application. Figure 2 on the previous page shows the relationships between them and how data flows through our application.

- Hero - The hero class is responsible for storing information about a hero such as id, name, score and role. We implemented simple getters and setters for the id, name, role and score properties. Id and score are stored as ints, and name is stored as a string value. Role is stored as a role object. The hero class is also responsible for calculating the score for a hero based on a list of enemy id's that is passed into the calculateScore function. The database is queried to get the values for calculating the score based on the list of heroes that is passed into the calculateScore function.

- Role - The role class is responsible for storing information about a role such as id and name. Each hero has 1 role, and we use roles in the application simply for display purposes.

- HeroController - The hero controller class maintains a vector of heroes, enemies and roles. The connection to the database is established in the HeroController, which is instantiated once during the application startup. After the database connection is established, the vector of heroes and roles is populated for use during runtime. As the user adds and removes heroes to the enemy team, the HeroController maintains the vector of enemies to use when calculating the composite score and retrieving counters for each hero. HeroController has different functions for accessing the hero and role lists such as getting specific heroes by name, getting roles by id, calculating all hero scores, and adding and removing items from each vector respectively.

- MainWindow - The MainWindow class is responsible for creating and updating the UI throughout the lifecycle of the application. The HeroController is instantiated once and is stored as a property in order to pass data to the UI for display purposes. The MainWindow class has functions for handling click events and keypress events that determine which heroes are added and removed from the list representing the enemy team as well as the result table. We utilized the QListWidget ad QTableWidget Qt classes for displaying data, along with the QPushButton and QLabel classes for the rest of the UI. One cool thing about the Qt display components is the ability to use CSS along with display properties attached to each component. We were able to avoid having to subclass the widget classes by utilizing a combination CSS and different display attributes for the QListWidget and QTableWidget classes that we used for the UI.

**Outcomes / Lessons Learned**

Overall we were very successful in building our application. We accomplished all of the goals that we set at the beginning of the semester to have a working MVP application. Getting started took a little extra work since we were still fairly new to C++ and neither of us had any experience working with Qt in the past. Furthermore each of us had issues in the beginning getting the MySQL drivers to work properly with Qt. After some debugging we were able to figure out how to link the library files from our system to Qt and found out about the install_name_tool utility for changing dynamic shared library install names.

We each learned a good amount about Qt by reading their documentation and trying things out in our app. The Qt environment is similar to VisualStudio and Xcode in the sense that they provide you with components to build the user interface and you can interact with them through properties and events in code. Each of us had a little experience using something similar in the past so it wasn't too difficult for us to pick it up. Along the way we learned how to connect to a MySQL database using the Qt MySQL drivers, how to utilize some of Qt's collection containers and buttons for building the user interface, how to implement custom functionality by intercepting events, and how to work with the Qt environment.  We also noticed that as we learned more C++ throughout the semester it became easier to implement more functionality into the app. We were able to utilize vectors, lambdas, iterators, and build some of our own custom classes.

Besides the application development we were also successful working as a team and meeting our deadlines. At the beginning of the semester we set up a slack channel for communication and a git repository for the code, both of which made working together easier. We were able to divide up responsibilities in the beginning and maintain good progress on the application throughout the semester.