1/1/2021

Daniel Ewers

NMIT

DAT602 Project

Milestone 2

Table of Contents

[**Milestone One** 4](#_Toc75368479)

[Game Description 5](#_Toc75368480)

[Game Concept 5](#_Toc75368481)

[Form Navigation 5](#_Toc75368482)

[Live Game Play 5](#_Toc75368483)

[Player registration 5](#_Toc75368484)

[Game administration functions 5](#_Toc75368485)

[Confirmation for a game 6](#_Toc75368486)

[Player selection 6](#_Toc75368487)

[Gameplay 6](#_Toc75368488)

[Main objective and Scoring 6](#_Toc75368489)

[Sketches 7](#_Toc75368490)

[Storyboards 9](#_Toc75368491)

[**storyboard 1** – game opening form 9](#_Toc75368492)

[**storyboard 2** – Login 9](#_Toc75368493)

[**storyboard 3** – register account 9](#_Toc75368494)

[**storyboard 4** – main lobby for users 10](#_Toc75368495)

[**storyboard 5** – main lobby for administrators 10](#_Toc75368496)

[**storyboard 6** – game menu (options) 11](#_Toc75368497)

[**storyboard 7** – administrator settings 11](#_Toc75368498)

[**storyboard 8** – leader board 13](#_Toc75368499)

[**storyboard 9** – game form 13](#_Toc75368500)

[**storyboard 10** – Admin creates new user 13](#_Toc75368501)

[**Storyboard 11 –** Admin edits existing user 14](#_Toc75368502)

[**storyboard 12** – game confirmation (new game) 14](#_Toc75368503)

[**storyboard 13** – game confirmation (join existing game) 15](#_Toc75368504)

[**storyboard 14 –** Incorrect username or password 15](#_Toc75368505)

[**storyboard 15** – Username ALready Exists 16](#_Toc75368506)

[**storyboard 16** – Change user information 16](#_Toc75368507)

[**storyboard 17** – Confirm deletion 16](#_Toc75368508)

[**storyboard 18** – Leave confirmation 17](#_Toc75368509)

[**storyboard 19** – Congratulations (Game finished) 17](#_Toc75368510)

[**storyboard 20** – Congratulations (Picked up asset) 17](#_Toc75368511)

[Screen Design Rationale 19](#_Toc75368512)

[Login/Sign Up Screens 19](#_Toc75368513)

[Home Lobby Screen 19](#_Toc75368514)

[Leader board Screen 19](#_Toc75368515)

[Game Lobby Screen 19](#_Toc75368516)

[Game Screen 19](#_Toc75368517)

[Administrative Settings Screen 20](#_Toc75368518)

[Logical ERD 0](#_Toc75368519)

[C.R.U.D Table 0](#_Toc75368520)

[Usage Scenarios 1](#_Toc75368521)

[User Creates an account 1](#_Toc75368522)

[User Deletes account 1](#_Toc75368523)

[User Creates a character 1](#_Toc75368524)

[User Creates a game 1](#_Toc75368525)

[User leaves game and Rejoins game 1](#_Toc75368526)

[User moves in game 1](#_Toc75368527)

[User finishes game 2](#_Toc75368528)

[Admin opens settings interface 2](#_Toc75368529)

[Admin Creates a User account 2](#_Toc75368530)

[Admin deletes a player 2](#_Toc75368531)

[Admin updates a current player 2](#_Toc75368532)

[Admin ends game 2](#_Toc75368533)

[Admin resets game 2](#_Toc75368534)

[DDL SQL 3](#_Toc75368535)

[**Create Table Procedure** 3](#_Toc75368536)

[**INSERT** 5](#_Toc75368537)

[**UPDATE** 10](#_Toc75368538)

[**SELECT** 15](#_Toc75368539)

[**DELETE** 18](#_Toc75368540)

[**Milestone Two** 23](#_Toc75368541)

[SQL Procedures and Transactions 24](#_Toc75368542)

[Live Game Play 24](#_Toc75368543)

[Player Registration 25](#_Toc75368544)

[Player Selection 25](#_Toc75368545)

[Game Administrative Functions 25](#_Toc75368546)

[Confirmation for a Game 26](#_Toc75368547)

[Test Data 27](#_Toc75368548)

[ACID 30](#_Toc75368549)

[What is ACID? 30](#_Toc75368550)

[CRUD 32](#_Toc75368551)

[Milestone Three 33](#_Toc75368552)

[GUI Implementation 34](#_Toc75368553)

[how have i implemented it? 34](#_Toc75368554)

[Login 34](#_Toc75368555)

[choosing from a list of online and available players 34](#_Toc75368556)

[game confirmation 34](#_Toc75368557)

[A game Display 35](#_Toc75368558)

[an administrator’s window allowing for player detail display 36](#_Toc75368559)

[Proof of gameplay 0](#_Toc75368560)

[Design Changes/additions 0](#_Toc75368561)

[Home Form 0](#_Toc75368562)

[Login Form 1](#_Toc75368563)

[Sign Up Form 1](#_Toc75368564)

[General user lobby 2](#_Toc75368565)

[admin lobby 3](#_Toc75368566)

[Leaderboard 4](#_Toc75368567)

[admin settings form 4](#_Toc75368568)

[Admin add user form 5](#_Toc75368569)

[admin edit user form 6](#_Toc75368570)

[game lobby 7](#_Toc75368571)

[Game Board Form 8](#_Toc75368572)

[Dialog Messages 9](#_Toc75368573)

[Database Changes/additions 10](#_Toc75368574)

[Additional Changes 11](#_Toc75368575)

[Gameplay Changes/additions 12](#_Toc75368576)

[Update to ACID Section 14](#_Toc75368577)

[How does MySQL implement ACID? 14](#_Toc75368578)

[Related MySQL atomicity features include: 14](#_Toc75368579)

[Related MySQL Conistancy features include: 14](#_Toc75368580)

[Related MySQL Isolation features include: 14](#_Toc75368581)

[Related MySQL Durability features include: 14](#_Toc75368582)

[References 15](#_Toc75368583)

Milestone One

# **Milestone One**

# Game Description

### Game Concept

I have come up with a concept for a multiplayer game that follows the idea of navigating a maze. The objective of the game is to start the game at the home tile and navigate through a variety of different pathways to reach the mazes exit. Throughout the maze will be randomly generated items that give the player extra points. There will be points given based on the order of completion of the maze. First place would get 100 points, second 75, third 50, and so on. You may choose to be the quickest to reduce the risk of losing points, or you could take your time to collect items at the expense of not being the fastest through. If you choose to go slowly, you may find yourself loosing points due to landing on trap tiles.

### Form Navigation

There will be several different forms within the application. This will include login forms, administrator settings, game creation and gameplay forms. These will be accessed by clicking sequences of buttons that will take you to different forms. This navigation will be simple and intuitive to give the best user experience. The user will start with a main form that give them two options of logging in or signing up. Once completed, they will then find themselves in the main menu of the game. This will display different options depending on whether the user is an administrator or not. The game creation form will display currently active games and options for users to create, reconnect, and refresh games, as well as the player selection. Administrative options will give the admin powers to do a range of different things such as deleting players, assign administrative privileges, stop games, and restart the games.

### Live Game Play

When a user is in a game, they can see other players that are nearby. This will allow the user to follow them, possibly take points from them, or scare them off. The games will continue to run when a player has left allowing the other users to continue with their game. This will stop the disruption of the gameplay.

### Player registration

As stated before, the user will first open a form that gives the user an option to log in or sign up. The game will implement a registration feature that allows a user to sign up with a username, password, email, first name and last name. By allowing them to input an email and username, it gives them an option to use either when they want to log in. This stops them from being locked out of their accounts if they have forgotten their username.

### Game administration functions

A user that has administrator rights has the privileges to perform a variety of different tasks. This includes such things as resetting a player score, removing the player from the database, unlocking a locked account, change a user password, give administrator rights, end a current game, or restart the game. This allows us management of games and players within the game. The benefit of this is so we can stop games that don’t have players in it.

### Confirmation for a game

When a user creates a new game, a simple popup box will alert the user to tell them that they have successfully created a game. There will be an additional popup when a user clicks the button to join an existing game. This will allow them to accept or decline the game just in case it was a miss click.

### Player selection

When you go to join a game, create one, or join back to your existing game, there is an option to choose what colour you player will be. This will be a set list of different colours that you can choose from. The colour you pick will be displayed within the maze representing your player. This will help you identify yourself and where you are in the maze.

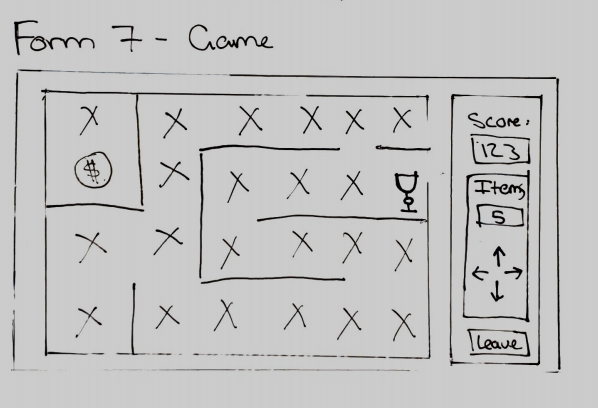
### Gameplay

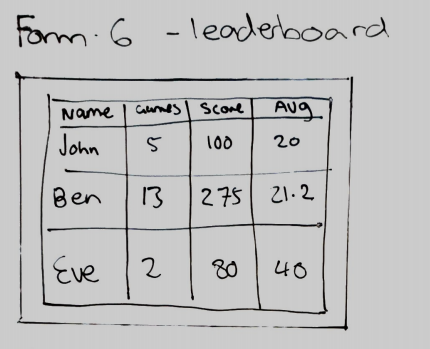
The user will start the game located at the home tile. They will have a set of controls (Up, down. Left, right) that will allow them to navigate through the maze. This will be single movements from tile to tile. They will have a small window where they can see around them limiting their vision of the maze. They will navigate through the maze looking for items to gain additional points, or rush for the exit. The player will want to have the best luck not to land on random chance tiles that could potentially take away their points. Once a player is satisfied with the items that they have collected, they can exit the maze. There will also be rare items within the maze that are kept with the user indefinitely. These will give the user special abilities for future games.

### Main objective and Scoring

The main objective of the game is to exit the maze with as many points as you can. This is because there will be a leader board that displays the statistics of the players. The better your score, the higher you will be placed on the leader board. This will be based off total score, games played, and average score across the total games. You can choose how you decide to get points whether it be collecting items or racing for the end. Whatever you choose to get points, the main objective is to finish the maze with these points.

# Sketches





The first step to creating my game was the planning phase. I decided to start off by brainstorming possible ideas of possible games that would meet the requirements of the assessment. Each game had different ideas branching from them of possible ways of making it a multiplayer game or other ideas within the game. I also planned out what forms might be found within the application. This helped me gain an understanding of what I would need to implement to meet the specifications of the app.

# Storyboards

From my initial sketches, I made each form using Adobe XD. This gave me a better understanding of how the final product may look like. These are not the final design of the forms as I may come across some additional or redundant features further down the line.

|  |
| --- |
| **storyboard 1** – game opening form |
|  |
| 1.1 Button that takes user to storyboard two  1.2 Button that takes user to storyboard three |
| **storyboard 2** – Login |
|  |
| 2.1 Input field for the user to input their username or password  2.2 Input field where a user can input their password  2.3 confirmation button which sends the information to the database and returns a result. If the username or password is incorrect, storyboard fourteen opens. If the user is an administrator, the user is taken to storyboard five. If the user is a regular player, they are redirected to storyboard four.  2.4 Button that takes the user back to storyboard one |
| **storyboard 3** – register account |
|  |
| 3.1 Input field where a user can input a username  3.2 Input field where a user can input a password  3.3 Input field where a user can input their email address  3.4 Confirmation button which sends the information to the database and returns a result. If the username is already taken, storyboard fifteen opens. If the information is accepted, the user is taken to storyboard four.  3.5 Button that takes the user back to storyboard one |
| **storyboard 4** – main lobby for users |
|  |
| 4.1 Button that takes user to storyboard six  4.2 Button that takes user to storyboard eight  4.3 Button that takes user to storyboard seventeen |
| **storyboard 5** – main lobby for administrators |
|  |
| 5.1 Button that takes administrator to storyboard  5.2 Button that takes administrator to storyboard |
| **storyboard 6** – game menu (options) |
|  |
| 6.1 List box of all active players in the game. Users can join the games of these active players.  6.2 Button that takes user to storyboard thirteen  6.3 Button that takes user to storyboard  6.4 Active player item in the list  6.5 Button that takes user to storyboard thirteen  6.6 Button that takes user to storyboard  6.7 Combo box that allows a user to select their character  6.8 Button that refreshes the active player list |
| **storyboard 7** – administrator settings |
|  |
| 7.1 Combo box for the administrator to select a user from the list  7.2 Button that resets the players score to zero in the database  7.3 Button that removes all records of a user in the database  7.4 Button that takes administrator to storyboard eleven  7.5 Tick box that gives the user administrative rights. This updates the user\_isAdmin item in the database to true or false  7.6 Tick box that locks or unlocks a users account. This updates the user\_accountStatus item in the database to locked or unlocked.  7.7 Button that takes administrator to storyboard ten  7.8 List box of active games. The items can be set to active by clicking on them. This allows the admin to choose what game they want to edit.  7.9 Button that takes administrator to storyboard five  7.10 Button that removes all records of a game in the database  7.11 Button that resets all records of a game in the database |

|  |
| --- |
| **storyboard 8** – leader board |
|  |
| 8.1 Grid box that contains the leader board  8.2 Button that takes user to storyboard five  8.3 Column for usernames  8.4 Column for games played. This updates when a game is played  8.5 Column for game score. This updates when a game is played  8.6 Column for average score. This updates when a game is played |
| **storyboard 9** – game form |
|  |
| 9.1 Asset tile. This can be occupied by any player. If a user clicks it, they receive the item and changes it to a regular tile. On clicking the tile, this opens storyboard twenty  9.2 Score of the current player. This increases as a player collects more items and progresses through the maze.  9.3 Number of items that a player is carrying.  9.4 Player tile. This cannot be occupied by another player while you are on it  9.5 Empty Tile. This can be occupied by any player  9.6 Display of controls to show the user how to play the game  9.7 Button that takes user to storyboard eighteen  9.8 Chat list displays text when user submits it by clicking the send button  9.9 Button that sends the text within the text field to the chat and displays it in the list  9.10 Input field where a user types the message |
| **storyboard 10** – Admin creates new user |
|  |
| 7.5.1 Button that takes user to storyboard five  7.5.2 Input field for the admin to input a username  7.5.3 Input field for the admin to input a password  7.5.4 Input field for the admin to input an email  7.5.5 Text box for the admin to check if they want the user to be an admin  7.5.6 Confirmation button which sends the information to the database and returns a result. If the username is already taken, storyboard fifteen opens. If the information is accepted, the administrator is taken back to storyboard seven. |
| **Storyboard 11 –** Admin edits existing user |
|  |
| 7.4.1 Button that takes user to storyboard five  7.4.2 Input field for the admin to input a username  7.4.3 Input field for the admin to input a password  7.4.4 Input field for the admin to input an email  7.4.5 Text box for the admin to check if they want the user to be an admin  7.4.6 Confirmation button which sends the information to the database and returns a result. If the username is already taken, storyboard fifteen opens. If the information is accepted, the administrator is taken to storyboard sixteen. |
| **storyboard 12** – game confirmation (new game) |
|  |
| 6.6.1 Button that creates a new game in the database. In doing so, this removes all previous game records that this user has created. This then sends the user to storyboard nine.  6.6.2 This button cancels the user’s actions and returns them to storyboard six |
| **storyboard 13** – game confirmation (join existing game) |
|  |
| 6.2.1 Button that takes user to storyboard nine  6.2.2 This button cancels the user’s actions and returns them to storyboard six |
| **storyboard 14 –** Incorrect username or password |
|  |
| 2.3.1 Button that closes box and returns the user to storyboard two |
| **storyboard 15** – Username ALready Exists |
|  |
| 11.1 Returns to storyboard three for users or storyboard ten for administrators |
| **storyboard 16** – Change user information |
|  |
| 7.4.6.1 This button updates any changes to the database then returns the user to storyboard seven  7.4.6.2 This button cancels the administrator’s actions and returns them to storyboard seven |
| **storyboard 17** – Confirm deletion |
|  |
| 10.1 This deletes all records of the user from the database. If they are a user, they are returned to storyboard one. If they are an admin, they return to storyboard seven  10.2 This button cancels the actions and returns them to storyboard seven if they are an admin or storyboard four for users. |
| **storyboard 18** – Leave confirmation |
|  |
| 9.7.1 Saves the users current position and returns them to storyboard six  9.7.2 Cancels the user’s actions and returns them to storyboard nine |
| **storyboard 19** – Congratulations (Game finished) |
|  |
| 12 Stores the new data in then database and returns the user to storyboard six |
| **storyboard 20** – Congratulations (Picked up asset) |
|  |
| 9.1.1 Updates the users inventory then returns the user to storyboard nine |

# Screen Design Rationale

### Login/Sign Up Screens

I wanted to create a simplistic design that was aimed at achieving the required goal without trying to create something unique. Quite often it is better to go with what users are use too rather than creating something new. Therefor, I design simple login and sign up screens that are simple and intuitive for the user. Due to the requirements of the assessment, I designed it so that if the username already exists or the password is incorrect, error boxes appear to notify the user. If the username doesn’t exist, they have the option to open the sign-up form instead. This gives them the option to create a new account.

### Home Lobby Screen

I decided that I wanted to create a simple home screen so that the login step doesn’t take the user straight to the game lobby. I like this idea as I can display a variety of different options such as the leader board button, settings button, delete button, and game button. I added the minor touch of the “Dungeons and Players” versus the “Dungeons and Administrators” as a bit of a gimmick to identify the different user interfaces. This subtle change gives the users a simple change in interface. In doing this, I also changed what options the users and admins have as the settings button would be redundant for an ordinary user.

### Leader board Screen

I decided to add a leader board with a simplistic design that is intuitive for the users with simple headings and a basic grid layout. The idea for this is to make it efficient for the user when they want to look for their details in the leader board. It only displays what I believe as to be the necessary information, that being username, games played, overall score, and average score. The purpose of the leader board is for users to compete to reach the higher positions and view other people scores. In the top right is a home button that takes the user to the home lobby screen. This was a gey element as the user needs to back out of the page if they did not wish to access it.

### Game Lobby Screen

The game lobby was designed with the systems requirements in mind. I started off by planning what is needed for the assessment. I created a simple list on the left that displays all the active users in their game. This gives users the option to join these pre-existing games. In the centre is where we find the basic options. These being continue game, new game, colour (character) selection, and refresh games. These were placed here as the central region is where you would generally place the key elements. To the right is a field that displays the game type you are playing. This is a nice feature as it adds design to the overall page.

### Game Screen

The game screen was design with the idea of keeping the key element in the centre. This is so that the user can keep their focus on the centre rather than looking off to the side. The elements on the game screen are bold so that it is easy for the user to identify features. I placed the player information on the left-hand side as we tend to find the chat on the right of games (as seen in my design). Like other features throughout my app, I wanted to keep to the general layout convention of apps as they are a lot more intuitive than coming up with potentially confusing ideas.

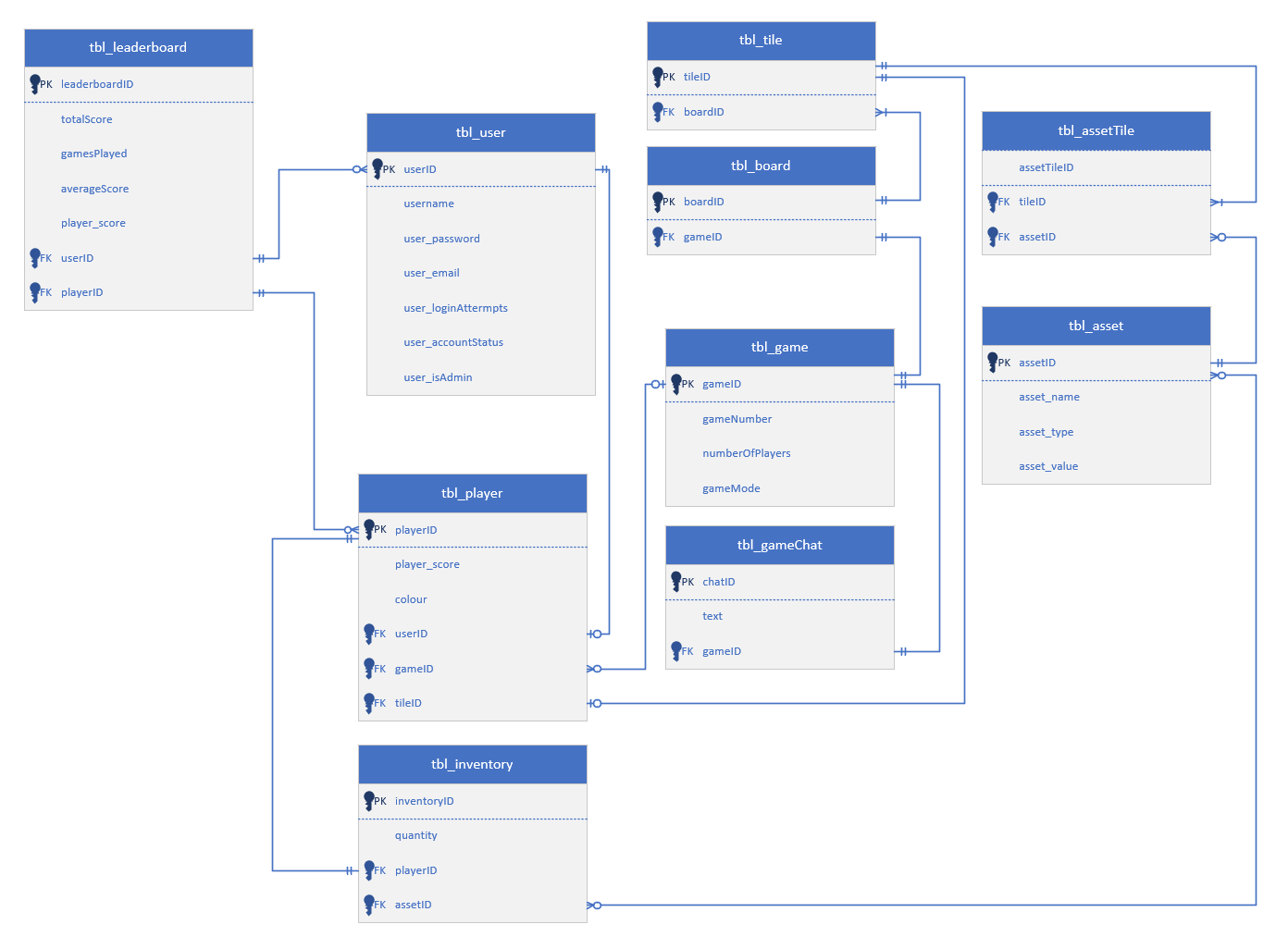
### Administrative Settings Screen

Like the other screens, the administrative screen was design with simplicity and intuitiveness in mind. I originally had the settings split into two screens based on their purpose. I decided to combine it into one screen as this increases the efficiency of game management for the administrators. The elements within the form are large and bold so that they are easy to read while keeping uniform shapes.

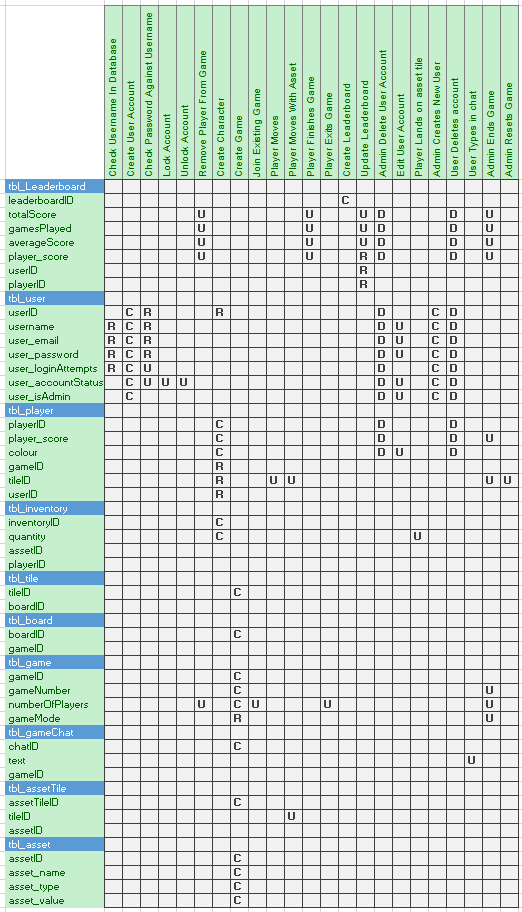
As stated in the assessment, the requirements for the administrative settings is for them to kill games, add new players, update existing players, or remove players. I added a drop down that displays all the player accounts. This allows the admin to select a player that they want to manage. From here they can delete the player, reset the player, modify player details, give admin privileges, or unlock account. This meets the requirements for the player management. There is a button at the bottom that opens a screen to create a new player account. This meets the requirements of adding new players. On the right side we have a list of active games. When the admin selects the game, they can then end the game or reset the game. This meets the requirements to kill active games.

For the create user, I reused the sign-up form with the addition of the give admin tick box. This keeps a informality across the forms.

# Logical ERD



# C.R.U.D Table



# Usage Scenarios

### User Creates an account

A new user will click the sign-up button. This will open a form that allows a user to enter a username, password, and their email. This gives them two options when it comes to logging in. When a user enters a username, the system will check to see if this is already taken. If the username is not taken, they will be prompt to enter a password then alerted that they will be creating a new user account. If the player name is an existing name, the player is prompted for a password and the password they submit is checked to see if that password is correct. After five tries the player is locked out, and an administrator email is presented. On accepting the new users account, they will be added to the list of online players. If an existing player succeeds at logging in, the player’s name and details are added to the list of players online

### User Deletes account

A user has an option to delete their account from the home page. This is represented by a trash can icon. This will prompt a dialogue box for the user to confirm the deletion of their account. If they choose yes, this will redirect them back to the login/sign up page. Upon deleting the account, all records of the user will be removed from the database. They will no longer be able to login to that account. In doing so, this will allow that username to be used. The other option is to cancel if they do not wish to delete their account

### User Creates a character

In the game menu, the user has the option to select between a range of different characters. These are represented by a range of pre-set colours that will display as a coloured circle when you play the game. There will be a limited number of characters to choose from as there will be a set limit of players in the games. The user is not limited to the same character every game, they can pick a different character each time if that character is available.

### User Creates a game

In the game menu, a user can choose to create a game. When they click the new game button, they will be met with a popup dialogue box. This will ask them if they are sure if they want to create a game. If they choose too to create a game, the previous game they created will be removed (to reduce the overall amount of games). They will then be directed to the game screen. The player moves around the maze from one tile to another, collecting items, competing with other players who are playing within the same tiled maze.

### User leaves game and Rejoins game

When a player leaves the game, their current state is kept in the database. When the player returns to the game the player, if the tile they were on is currently empty, they return to the tile on the map they were on when they left the game, otherwise the player must choose a different neighbouring tile to continue playing the game.

### User moves in game

Players start on the “Home Tile”. Apart from the “Home tile” only one player can be placed on each tile at time. The player who achieves the first “click” on an empty tile moves to that tile. Once on a tile the player can click on the items on that tile to gain or lose points. When a player moves from the tile, the tile becomes empty. The current tile that the user is on is stored within the database for future reference. If the player has an asset in their inventory, the asset tile ID will update as the player moves around the board.

### User finishes game

When the user has made it to the end of the maze, their points will be added to the leader board. This will add to the total points as well as calculating the average score. Additionally, the number of games played will increase by one. If the user does not finish the game, the total number of games played will still increase having a negative impact on their statistics.

### Admin opens settings interface

When the admin clicks the setting button, this will open an interface that displays a series of different options for managing player accounts, creating new accounts, and managing games. This will include such things as ending games and deleting accounts, to giving administrative rights.

### Admin Creates a User account

When the admin clicks the plus button, a window will pop up that looks the same as the sign-up form (with the addition of a give admin rights check box). This will allow an admin to create an account for a user. On completion, the new user account details will be added to the database. Like the user sign up form, the admin will be prompted with the same error messages if there is duplicate data.

### Admin deletes a player

Unlike the users that can only delete their own account, an admin has the option to delete any account that ahs been created. They will be prompted with the same popup for them to confirm the deletion. If they choose to delete the player, all records of that user will be removed from the database.

### Admin updates a current player

Within the admin controls, the admin has the power to update information about the user. This includes changing their username, password, email, giving admin rights, or account status (if they are locked out). Like the other admin controls, a popup box will allow the admin to confirm or cancel these changes.

### Admin ends game

The admin has rights to end any game that is in progress. If they choose to do so, all records of that game will be removed, and the players will be kicked from the game. This will remove the game from the list of active players. Like the other admin controls, a popup box will allow the admin to confirm or cancel this action.

### Admin resets game

The admin also has the rights to reset a game. This will move all players back to the start of the game and set their score to zero. In doing so, the map will also rest. This will refresh the assets on the map. Players will then have to restart the game. Like the other admin controls, a popup box will allow the admin to confirm or cancel this action.

# DDL SQL

## **Create Table Procedure**



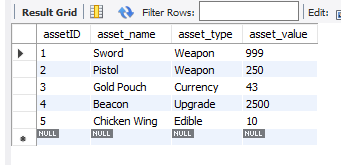




## **INSERT**

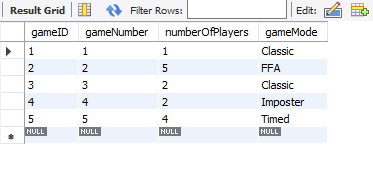
#### tbl\_asset





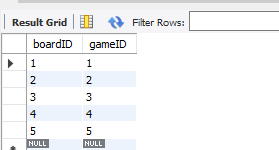
#### tbl\_game





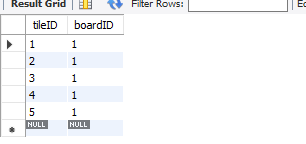
#### tbl\_board





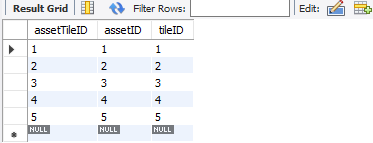
#### tbl\_tile





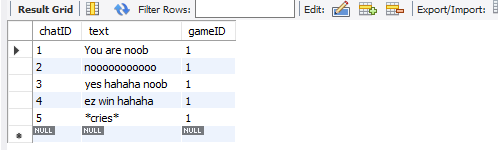
#### tbl\_assetTile





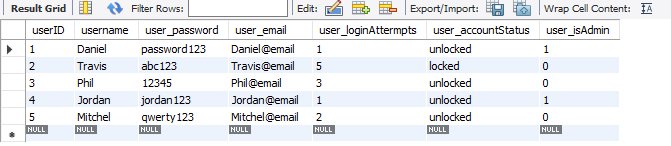
#### tbl\_gamechat





#### tbl\_user





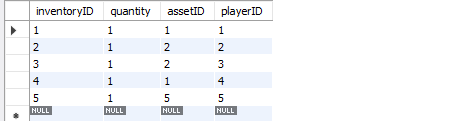
#### tbl\_player





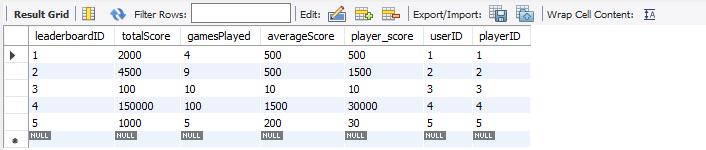
#### tbl\_inventory





#### tbl\_leaderboard



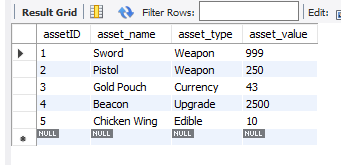


## **UPDATE**

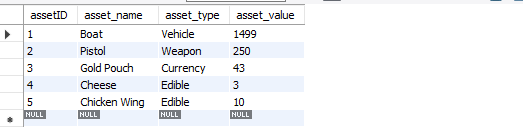
#### **tbl\_asset**



**Before**



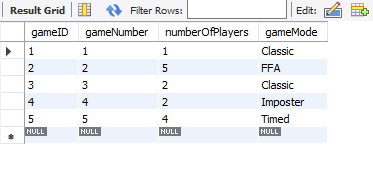
**After**

****

#### **tbl\_game**



**Before**



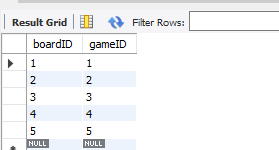
**After**

****

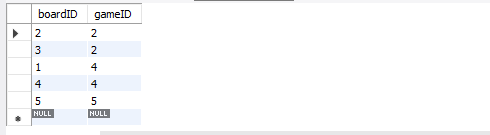
#### **tbl\_board**



**Before**



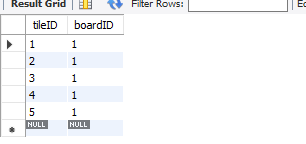
**After**

****

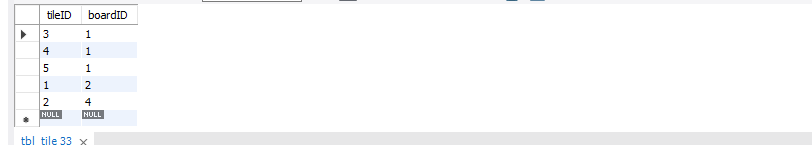
#### **tbl\_tile**



**Before**



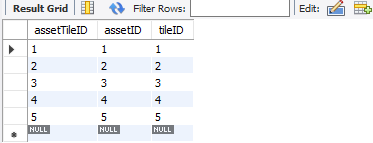
**After**

****

#### **tbl\_assetTile**



**Before**



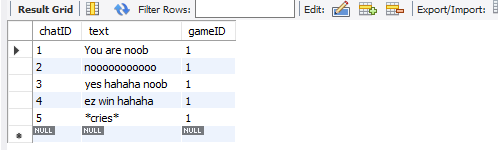
**After**

****

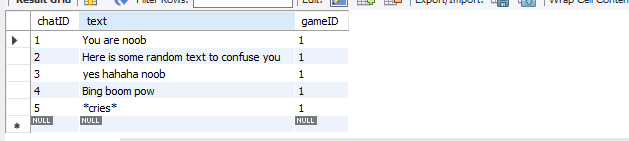
#### **tbl\_gamechat**



**Before**



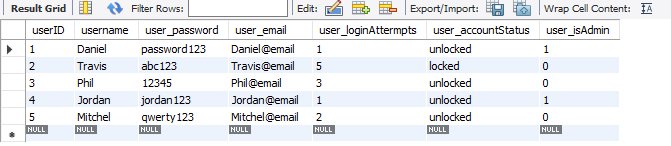
**After**

****

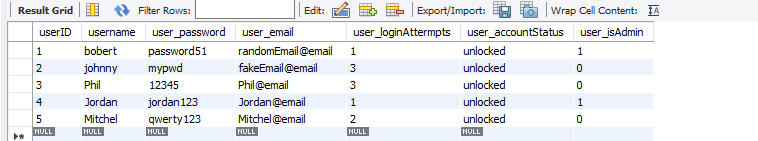
#### **tbl\_user**

****

**Before**



**After**

****

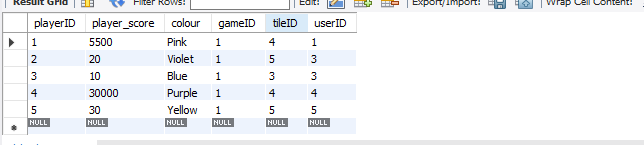
#### **tbl\_player**



**Before**



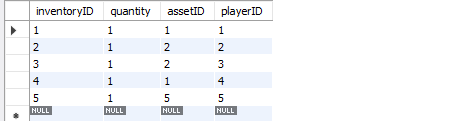
**After**

****

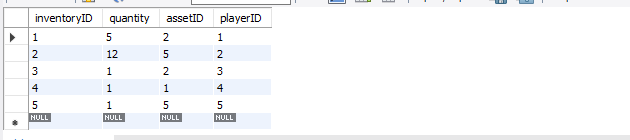
#### **tbl\_inventory**



**Before**



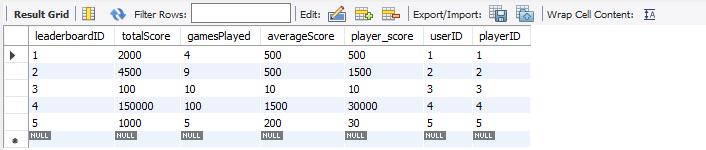
**After**

****

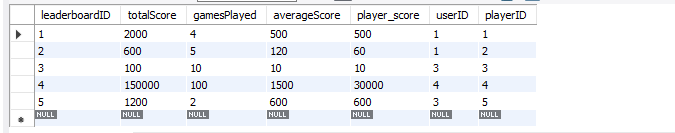
#### **tbl\_leaderboard**



**Before**



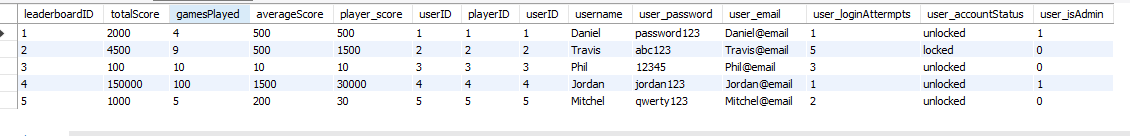
**After**



## **SELECT**

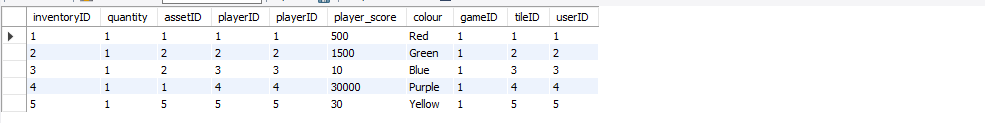
#### tbl\_leaderboard and tbl\_user





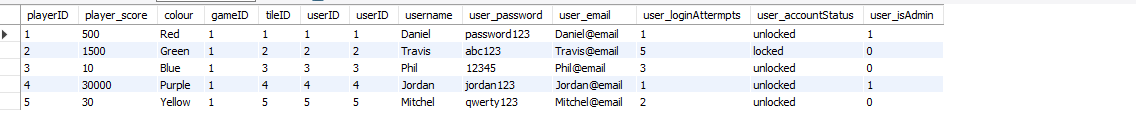
#### tbl\_inventory and tbl\_player





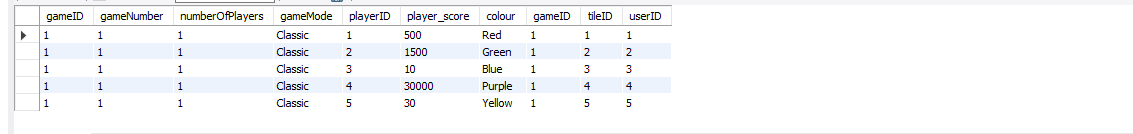
#### tbl\_player and tbl\_user





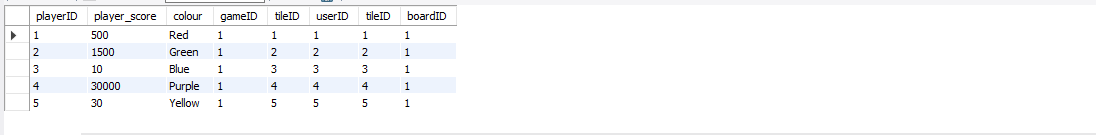
#### tbl\_game and tbl\_player





#### tbl\_player and tbl\_tile





#### tbl\_assetTile and tbl\_tile





#### tbl\_board and tbl\_tile





#### tbl\_board and tbl\_game





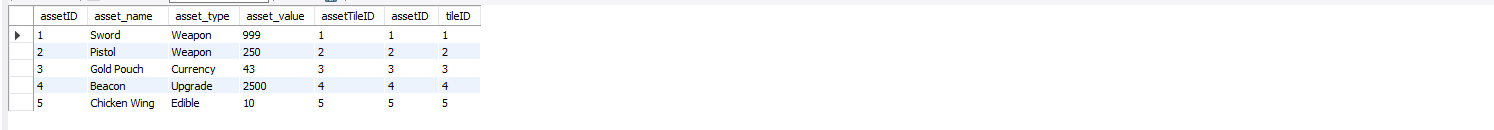
#### tbl\_game and tbl\_gamechat





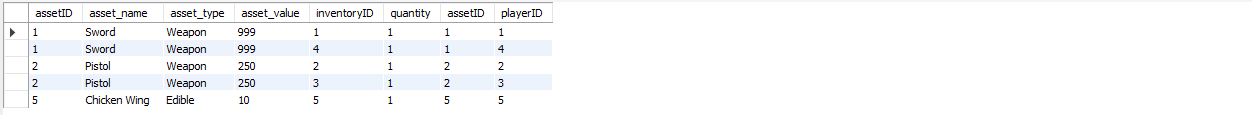
#### tbl\_asset and tbl\_assetTile





#### tbl\_asset and tbl\_inventory



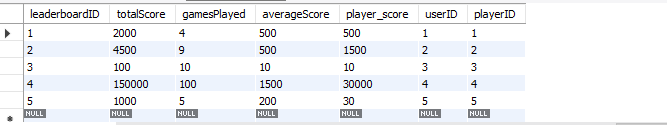


## **DELETE**

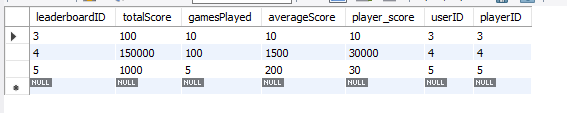
#### tbl\_leaderboard



**Before**

****

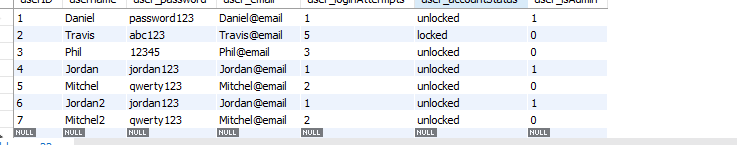
**After**

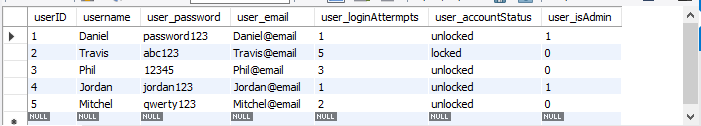
****

#### tbl\_user



**Before**

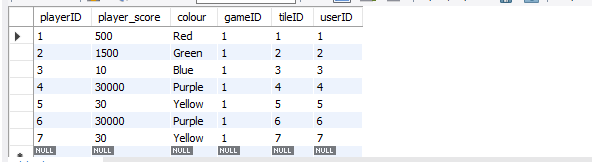
****

**After**

#### tbl\_player



**Before**

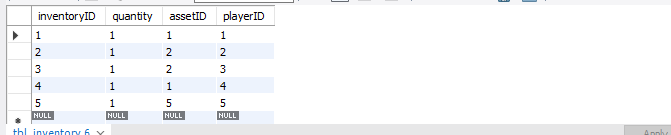
****

**After**

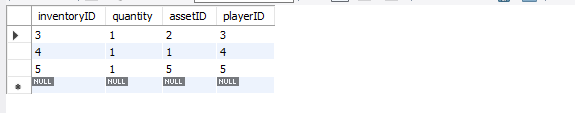
#### tbl\_inventory



**Before**

****

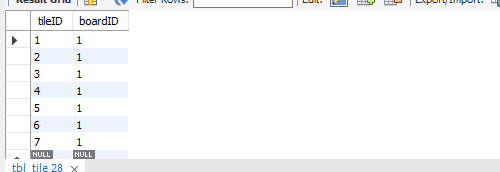
**After**

****

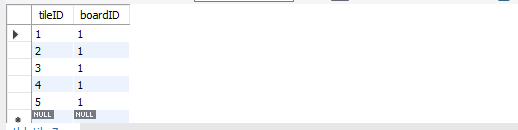
#### tbl\_tile



**Before**

****

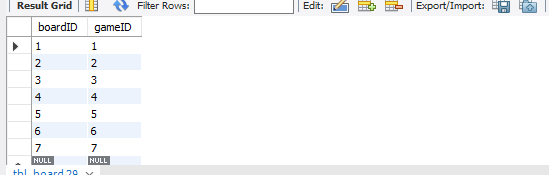
**After**

****

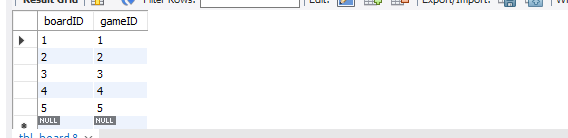
#### tbl\_board



**Before**

****

**After**

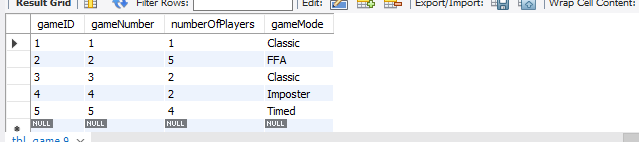
****

#### tbl\_game



**Before**

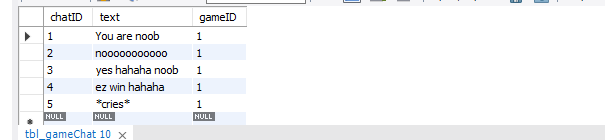
****

**After**

#### tbl\_gameChat



**Before**

****

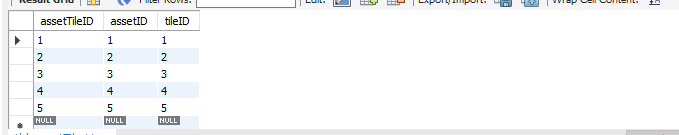
**After**

****

#### tbl\_assettile



**Before**

****

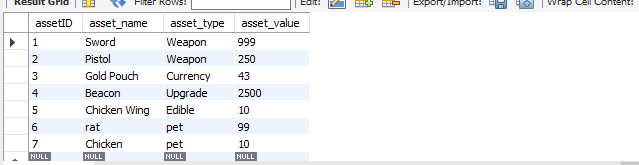
**After**

****

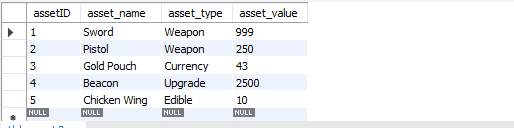
#### tbl\_asset



**Before**

****

**After**

****

Milestone Two

# **Milestone Two**

# SQL Procedures and Transactions

*I have added a Visio Diagram to the repository that shows how the system works.*

## Live Game Play

I have a Stored procedure that processes all of the character movements and one for checking the next tile before the application tries to call the player movement procedure. The validity checker stored procedure is called **CheckValid**. This passes in two parameters, the tile number that the player is trying to move too, and the game number. This calls a select statement to check if the tilePlayer column is null at the row of the specified tile and game. This is called with a bool method in c#. The query will read the column and return a bool value for the column being null or not null. If the procedure returns a value that is not null, the bool method will return false, else true for a null value. When false, it will throw an error and if it returns true, it will call the **UpdatePlayerTile** stored procedure. This passes in five parameters, **uname** (player username), **tileVal** (the new tile value), **lastVal** (last known position), **score**, and **num** (game number).

The validity checker runs every time a player tries to update their position and the **UpdatePlayerTile** runs when the previous procedure allows it too. The UpdatePlayerTile preforms many checks on the current status of the player and updates depending on the validity of the statements within. This stored procedure processes the player movement, the asset collection, and finish game updates.

The way I have handled the asset collection is to make the asset specific to the game that the player is in and will only be removed from the board if the player stands on the asset tile. This means that only that player can collect the item and not have duplicate item id values in the system. This is important for tracking what has and has not been collected from a game. A game will still run even if there are no items left on the board. Players can still play that board until it is deleted, but they cannot acquire more points from playing the game. This will mitigate the risk of idle farming of points.

When a user leaves the game, a stored procedure called **ExitGame** is called that passes three parameters, **uname** (username), **currentTile** (the players active tile), and **num** (the game number). This procedure updates the player table and sets the current game of the passed usernames player to null and stores their current position. It also sets the tiles tilePlayer in the tile table of that game to null so that players can access the tile when a player left. The number of players in that game is also decreased by one.

I then have a stored procedure called **JoinUser** that is called when a user wants to join an existing game, whether it is their own, or another user’s game. This passes two parameters, **uname** (username), and **gameNum** (the number of the game they want to join). If the current position of the user reads null, then their **tileID** is set to one as they are joining as a fresh player. If they have a previous tile value stored, then their position will be set to that value, else they will be offset to another tile.

## Player Registration

When a user tries to register an account, it runs through some checks before it does so. This included username checking, sign up attempt, and setting login status. The username checker calls a simple select statement procedure called **checkUsernameSignUp**. This is called in C# with another bool statement. If the select statement finds a record of that username, then it will return true and an error message displays telling the user that the username already exists. For testing purposes, if the username exists, it will then try to login with the entered details. This calls a stored procedure called **CheckInfo** that passes the parameters **uname**, and **password**. The procedure checks for a match in the database and returns an error if the password is incorrect. If the username and password is correct, it will then call a bool method that checks the account status of the user. This calls a stored procedure called **AccountStatus**. This returns a bool value called **isLocked** where a value of one is a locked account and the other is unlocked. If this procedure passes the user as not being locked out, it will then run a checker on the account type. The two outcomes are an admin or a regular user. This is another bool method that calls a stored procedure called **CheckIfAdmin** where a true value is an admin and false being a user. Depending on the returned value is to whether the settings option is displayed to the current user.

When a user tries to login and enters the wrong password, an error message will display letting them know of the failed attempt. This will also call a stored procedure called **FailedAttempt** with the parameter **uname**. This will update the user’s login attempts to the current value plus one. This also checks to see the total number of login attempts. If this matches five, then the account status is set to true (locked). The next time a user tries to login and the **AccountStatus** procedure is called and the current users account status is locked, it will throw an error saying that the account is locked, and they need to contact an admin.

## Player Selection

The player selection isn’t very exciting as the user can decide on what character they want to be before they when they are creating a new game. For the purpose of the console application, this option is decided in the test data, but when it comes to the GUI version, there will be a short list of options the user can choose from. When a user creates a game, a stored procedure called **setCharacterType** is called that passes the username and specified character type to the database and updates the player tables character to the passed type depending on the given user. This is separate from the procedure for creating the game as it is also used when a user wants to join an existing game, since the user can select a different character each time.

## Game Administrative Functions

The admin has several different options that they can choose from under two main categories, these being Game Settings and User settings. The game settings allow an admin to delete a game that is currently running and reset the games as well. User settings allow an admin to create users, delete users, update users, and set user status.

Each of the user settings runs its own stored procedure relative to what the admin is doing. These being:

**AdminCreateUser** – This is different to when a user registers as the admin has the option to set the new users administrator rights to true or false whereas this is not available in the sign-up process.

**AdminDeleteUser** – Deletes the user record from the database

**AdminAccountStatus** – alternates between locking and unlocking account

## Confirmation for a Game

When a user chooses to create a new game, the **insertGameData** stored procedure is called. This is the biggest procedure in the database as there is a lot of steps involved with creating a new game. The order in which this works is as followed:

* Declare two variables
* Insert a new game into the game table
* Insert a new board into the board table
* Insert a new chat into the chat table
* Update the player table and set the current game to the new game
* Set the brdID variable to be the last insert ID
* Create a statement that creates the tiles in the tile table
* Update the tbl\_game table and sets the games number to be the same value as the games ID
* Update the player table and sets the players tileID to be 1
* Update the tile table and sets the tilePlayer of the tile one to be the username of the current user
* Runs a set of insert statements that creates the default items on the board. This uses maths to set the range of each position and randomises their location. Each item has a range in which they can spawn so that there is no overlap. This is simple for board one as its range is from 1 – 100, but as you increase the boards, you have to change the formula (as shown below)

If this is all successful, the game grid is then drawn, and the user is displayed on the board. The items are not visible which stops users from running straight to the items and gaining points. They are expected to locate them by standing on the correct tiles.

Formula for board one assets:

Range 1: FLOOR(RAND()\*(32-2+1))+2

Range 2: FLOOR(RAND()\*(65-33+1))+33

Range 3: FLOOR(RAND()\*(98-66+1))+66

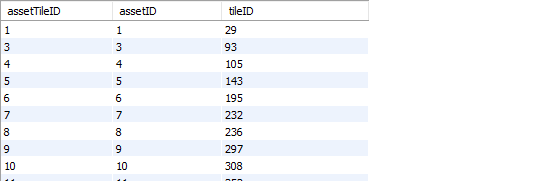
Formula for other assets:

Range 1: FLOOR((RAND()\*(32-2+1))+ 2)+((brdID-1)\*100)

Range 2: FLOOR((RAND()\*(65-33+1))+33)+((brdID-1)\*100)

Range 3: FLOOR((RAND()\*(98-66+1))+ 66)+((brdID-1)\*100)

Below is an example where we can see that every three assets, the tileID increments the range by 100.

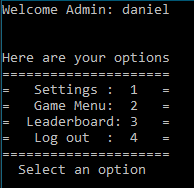


# Test Data

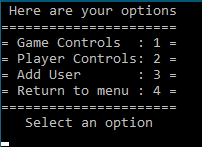
In the Program class, there is an instance of the Player class. This relates to the test data in the database for our admin account. When the system runs, the user has the option to login or sign up. With the test data that is shown below, if the user tries to sign up, it will inform the user that the account already exists. This will then try to log the user into the account. For the purpose of the test, the password is correct. If we were to change the password (“123”), it would then throw an error saying that the password is incorrect and return to the home screen. If the user chooses to login with the data, it will redirect them to the admin page as the **CheckIfAdmin** procedure will return that the user in the given test data is an admin.



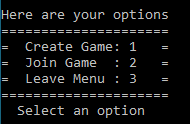
I have created two different home pages depending on the test data. This prevents users from accessing administrative features available to a user with administrator access. When an admin enters their correct credentials, the home screen will display as shown below with a settings option. Otherwise, a menu will display with all options except for settings. By logging in, the active user is set to that of the given test data. This means that we can reference the username of the current user to variable when we call the other stored procedures.



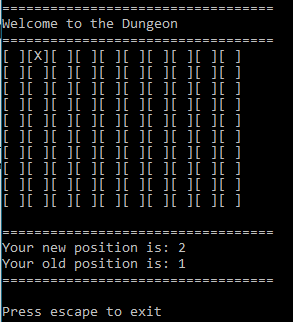
In the settings menu, there is a set of test options that the user can try, that call stored procedures depending on the data provided. I have decided to categorise them based on what the actions are that are performed. The game controls allow the user to select a game they would like to perform actions on. These being reset game and delete game. The player controls allow the admin to enter a player’s name from the list of test data, edit the players details, delete the player, or toggle the account between being locked and unlocked. The add user option is similar to that of the update user method, where you type in the details. This has an additional parameter to the sign up stored procedure as the admin can declare the user as a



In the game menu, there is two game options, create game, and join game. If a user decides to create a game, the **InsertGameData** stored procedure is called, and the user is redirected to the game grid. When a user chooses to join a game, a list of games is displayed, and they can enter the game number that they’d like to join.



Below we can see the board:



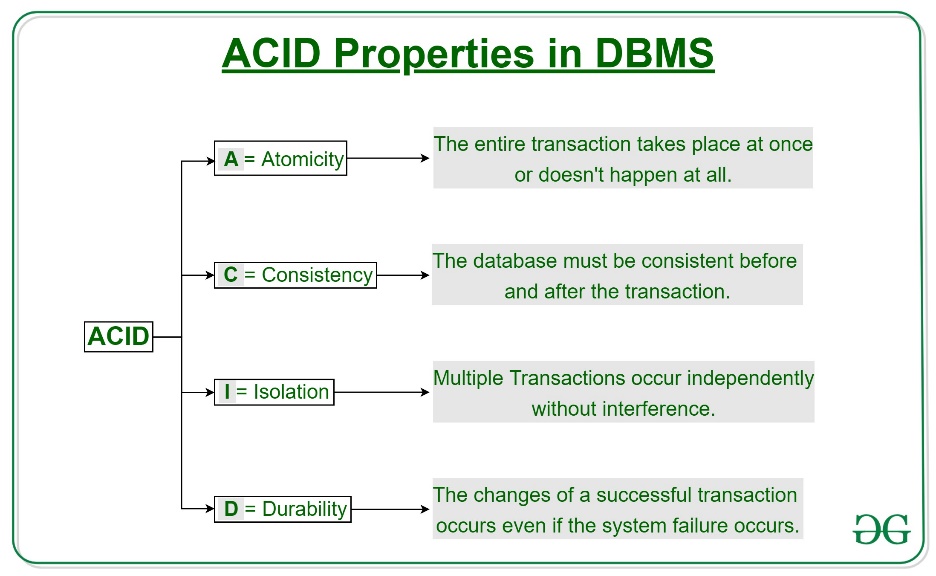
The user controls use W, A, S, and D to traverse the board. This allows me to demonstrate the use of tile validation. Each time you press a move key, the **IsTileValid** procedure runs to check if the user can move to that tile. If it is a valid position, the system will then use the new data as parameter values for the stored procedure “**UpdatePlayerTile**.” Each time this method is called, we can see in the database that the player has moved to that tile, and it is set to an occupied tile. Part of this procedure is the asset tile checker. If the new position contains an asset, it will be added to that user’s inventory. If a user leaves in the middle of the game, it will call the **ExitGame** procedure that stores the current location of the player in the database and remove them from the game. This decreases the player count by one. The player can then join back to the game in the same position that they left off.

I believe this is the best way to demonstrate that the system interacts with the database based on what my game is and how it runs. Below is a table that covers what stored procedure/procedures are used for each task:

|  |  |
| --- | --- |
| Stored Procedure | Task |
| Account Status | Checks the status of the user account when they try to log in. |
| AdminAccountStatus | Toggles user account status between locked and unlocked in the administrator settings. |
| AdminCreateUser | Called when an admin wants to create a user in the admin settings |
| AdminDeleteGame | Deletes the game record from the database |
| AdminDeleteUser | Deletes the specified user from the database |
| CheckGameNum | Checks to see if the game that the admin has selected exists in the database. |
| CheckIfAdmin | Checks to see if the user is an admin when they log into the game |
| CheckInfo | Checks the passed username and password against the records to see if they are correct. |
| checkUsernameSignUp | Checks to see if the username already exists when a user tries to sign up |
| CheckValid | Checks to see if the next tile is valid that the user is wanting to move too |
| createtbl | Creates the database tables |
| CreateUserAccount | Creates the user account when the all of the sign-up requirements are met |
| ExitGame | Updates the database when a user leaves during a game |
| FailedAttempt | Sets the users login attempts to increase by one when they have had a failed login attempt. |
| GetDataLeaderboard | Gets leader board data to display on the leader board |
| GetGameNumbers | Gets a list of game numbers for the administrator settings |
| GetGameNumber | Gets the players current game number when they create a new game |
| GetPlayerUsernames | Gets the usernames of all of the players to display in the administrator settings. |
| insertGameData | Inserts all of the game data when a user creates a game |
| JoinUser | Updates the user data when they join another game |
| LoadData | Gets the users data when the admin selects the edit user option |
| Logout | Updates the users account status to be logged out |
| ResetGame | Resets the games data in the database |
| setCharacterType | Sets the users character type to the one they have chosen |
| setGameNumber | Sets the users game number when they join a existing game |
| SetLoggedIn | Sets the users login status to be logged in |
| TestData | Some dummy data users to display on the leader board |
| UpdatePlayerTile | Updates the players position in the database during the gameplay. |
| UpdateUser | Updates the users details when the admin changes them |

# ACID

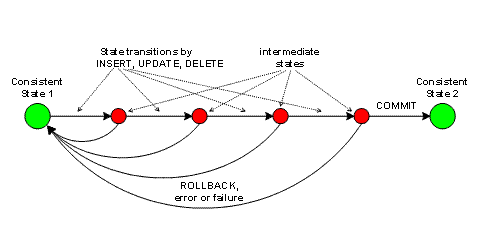
## What is ACID?

  
ACID is an acronym for Atomicity, Consistency, Isolation, and Durability. These are a set of properties related to database transactions, with the intention of ensuring that the database transactions are passed reliably. A transaction is a logical unit that performs tasks such as accessing data, adding new data, or manipulating existing data in the database. The idea of ACID is for us to understand how our database handles errors that might occur when we are trying to pass or manipulate data in our database. We need to make sure that data remains consistent and accurate even when a failure occurs during a transaction. These properties need to ensure that the database remains consistent both before and after the transaction is executed.

(GeeksforGeeks, 2021)

**Atomicity** – The Idea of atomicity is that when a set of transactions are run, we have an all or nothing process. If one part of a transaction were to fail, the overall transaction fails. The database requires that all of the transactions work in order for the data to be manipulated or accessed. This means that we don’t have some transactions passing and some not. To put it simply, the transactions do not occur partially. If the transaction aborts (fails), the changes It has made to the database are not visible, whereas if the transaction commits (passes), these changes are made visible (GeeksforGeeks, 2021).

In terms of my application and database, when a stored procedure is called, it will either perform the task that is defines, or it will throw an exception error that writes to the console. These generally occur when an index is out of range, a data type is invalid, trying to insert duplicate data, or constraint errors. When these errors are thrown, the stored procedure cancels its actions and does not perform the task it was originally intended to do. For example, if the application tried to pass a string value as a parameter value that is intended to be an integer, it will throw an exception error saying that the datatype is incorrect, rather than inserting the data into the table anyways.

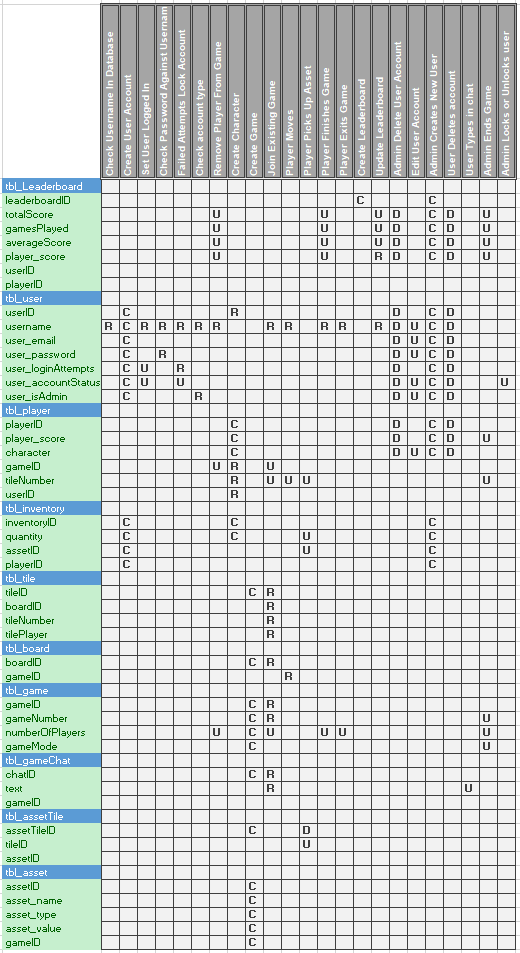
**Consistency** – Constancy refers to the idea that the data in your database refers consistent according the constraints, cascades, data rules and data types defined in the tables. This includes such things as not creating null values where the column is set to “not null,” or deleting, adding, or updating records that have constraints with other tables. I have used cascades and foreign key constraints to ensure that the data remains consistent when a transaction occurs. For example, when a user account is deleted, their leader board records are removed, their inventory data is removed, and any other relevant data that should not remain in the database when the user account is deleted. The use of constraints stops the system from modifying certain data without modifying all of the data relevant to that record. We should not update a game number without other records in the database updating that are relevant to that data. This will prevent random data staying in the database when records are deleted. This is seen in the diagram below:

Above we can see the lifecycle of a transaction. In the scenario where we want to transfer data from one location to another, the data must remain consistent both before and after the transaction. An example of this in the system could be transferring points from one player to another player. If player one has 10,000 points and player two has 5,000 points and we want to give player two 2,500 points, a successful transaction with consistent data should ensure that player one has 7,500 points and player two has the same amount after the transaction has committed. In the situation where the transaction aborts, both player one and player two would have their original points.

**Isolation** – The idea of Isolation is that transactions occur without interference from other transactions running simultaneously. The purpose of this is to ensure that there are not inconstancies with the current state of the database. Likewise, the transaction cannot read any data from another transaction unless that other transaction has executed successfully. The overall goal of this is to ensure that no transaction is affected by another transaction during its execution. As I am not trying to run multiple procedures simultaneously, I do not believe that this affects what I am doing. If I were trying to retrieve large quantities of data from multiple tables at the same time, this would be more of an issue that I would need to consider. I have split off tasks to ensure that they run successfully before the next task is executed, such as checking if the user exists before trying to create a new user.

**Durability** – Durability refers to the idea that when a transaction commits, the data will be written to the system and will remain persistent in the event of a system failure. These additions or modifications are stored permanent. Even if the system were to crash immediately after the transaction commits, it will remain in the system rather than having data loss. The purpose of this is to ensure that when the transaction is committed to the database, it must not be lost. MySQL contains multiple layers of protection to help support durability. As stated by Blum (2018), “MySQL writes all transactions to a log file, writes the changes to the double-write buffer area, and then writes them to the actual database files. If the system crashes during this process, most of the time MySQL can recover the transaction within the process.”

# CRUD



Milestone Three

# Milestone Three

# GUI Implementation

## how have i implemented it?

In order to meet the requirements, I have Implemented as many features as I can to ensure that it does what it required. Below is a breakdown of the parts of the application that I must implement. There are additional features that I haven’t broken down, as they are not required, but I have implemented them into my code

### Login

The first part I have implemented is the log in system. I have used a form that has a username field as well as a password field. When the user types in their username and password, and click the confirm button, it calls a method classed **CheckValidLogin** from the **DataAccess** class. This method passes the text that is in the textboxes as its parameters. This will call the **CheckInfo** stored procedure that receives the text and tries to select a record that matches the parameters. If the database returns a match, then the method called **OpenMenu** is called, but if the table row count is equal to zero, then it will say that the username does not exist, and the user needs to create an account.

The **OpenMenu** method then runs a few checks. The first is calling the method **CheckLock** from the **DataAccess** class that returns a bool value based on the returned value from the database. If the value is one (true), then the account status is locked, an error message is displayed, and the code finishes. Else, if the value is zero, then the account is not locked, and the code will continue. Then another method is called that also returns a bool value. This is the **CheckIfAdmin** method from the **DataAccess** class. This passes the username to the database with the **CheckIfAdmin** stored procedure to check if the user is an admin. If the returned value is one (true), then the user is an admin, when zero, the user is a regular user account. After this method is called and a value is returned, the user is redirected to the home page and the **SetLoggedIn** method is called which passes the username to the database which calls the **SetLoggedIn** stored procedure which sets the column value to one.

### choosing from a list of online and available players

The datagridview in the lobby form has a data source that is set to the returned value of the **GetLobbyData** method in the **DataAccess** class. This method is a static Data Table that is set to the values passed using the **LoadLobbyData** stored procedure and written to the data table using a MySQL data reader. The returned values are then displayed to the datagridview.

A method called ActivePlayers\_CellClick allows the user to click the cells in the datagridview. If they select a valid player that is in the game, their game number is displayed in a textbox below. If a player is not in a game, then it will throw a message saying that the user isn’t in a game and to select another user.

### game confirmation

When a user is in the main lobby and they have successfully selected a game mode and a character, they will; click the new game button. If the setCharacterType and the InsertGameData stored procedures execute correctly, then a message box will display with the message "Game created successfully."

When a user has successfully selected a user’s game that they would like to join, they will then click the join game button. If the JoinUser stored procedure executes correctly, then a message box will display with the message "Game created successfully."

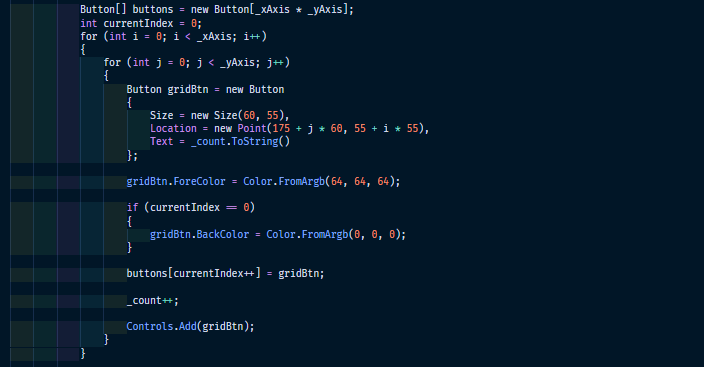
There isn’t much to say about game confirmation as there is only two possibilities. Either they successfully join/create a game, or an error is thrown.

### A game Display

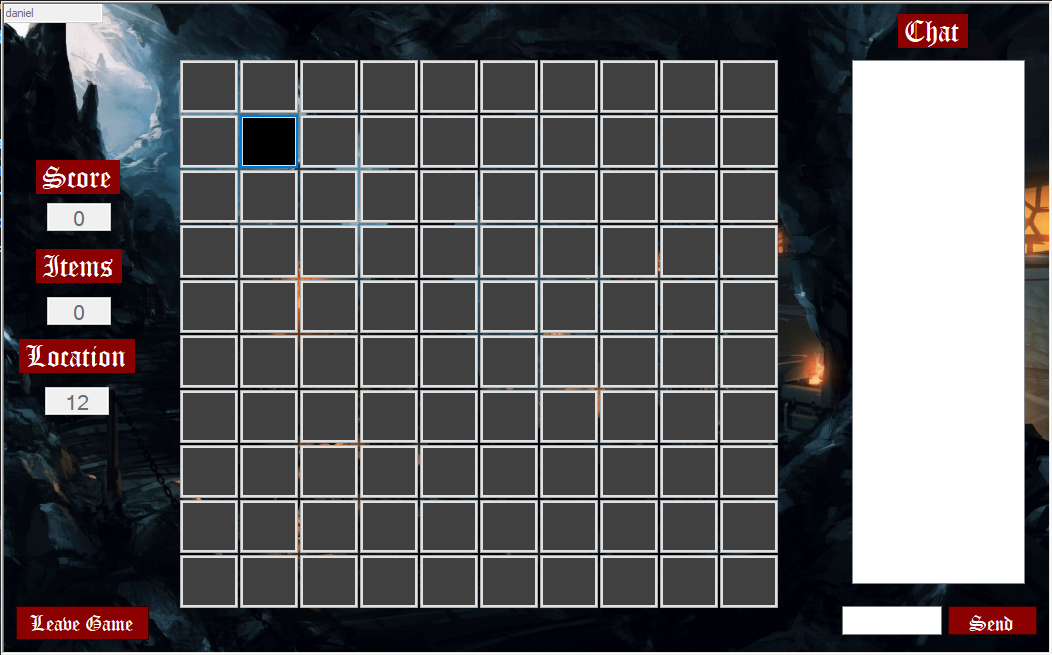
The original way I designed the windows form was by manually placing a 100 by 100 grid of buttons. This was a very inefficient method that made the form both laggy and not very future proof ready if I wanted to expand the board without having to make the form far larger than it already is.

I decided that I would create a 2d button array that would have two inputs, the x axis and the y axis (width and height). These can be adjusted depending on how large you want the grid. The only changes that would need to be made are the location of where you want the grid to be placed.

This uses the method below:



Which in the current configuration, draws this board:



A change I could make to this would be to add a stored procedure that returns the x and y values so that the board size can be defined database side. The different game modes in the lobby could define what size the board is, and when the game is created, the board size is set in the database. This would allow me to have a range of different sized boards for different experiences.

I could also change how the game is displayed by only showing the tiles that are directly next to the current tile. I would then change it from a click method to arrow keys and the user has to move around without seeing the whole board. I could then set a random finish tile so that the player cant go straight to the bottom corner.

### an administrator’s window allowing for player detail display

I decided to split the administrative settings into three forms to reduce the overall congestion. I have a general settings form, an add form that is different from the registration form, and an edit user form.

When the main form loads, a the **LoadData** method calls the **LoadSettingsData** method from the **DataAccess** class. This calls the stored procedure GetPlayerUsernames and GetGameNumbers which populates the games combo box and the players combo box.

Admin deletes user

* Calls the **AdminDeleteUser** stored procedure passing the player combo box text as its parameters, deleting the users records from the database.

Admin resets player score

* Calls the **AdminResetUser** stored procedure passing the player combo box text as its parameters, resetting the users score in the database.

Admin creates player

* Opens the create player form. When the admin enters details and clicks confirm, the **AdminCreateUser** method is called from the **DataAccess** class passing the username, password, email, and check box status’s as parameters for the stored procedure **AdminCreateUser**. Like the signup process, the **CheckUsername** method is called which passes the username text to the database with the stored procedure to see if the record exists. If the table count returns one, then the username already exists and the error message "This Username Already Exists, Please Enter a new one" is thrown. If the table count returns zero, then the username is unique and the **AdminCreateUser** stored procedure is executed.

Admin edits player

* Opens the edit player form passing the player combo box text as the user that is being edited. When the form is loading, the **LoadUserData** method is called passing the username, which is the parameter of the LoadData stored procedure. When the procedure is called, a data reader gets the returned values and sets the text field and checkbox values.
* When the admin clicks the confirm button, the **UpdateUser** method is called from the **DataAccess** class passing the username, password, email, and check box status’s as parameters for the stored procedure **UpdateUser**. This procedure updates the record that matches the username.

Admin ends game

* Calls the **AdminDeleteGame** stored procedure passing the game combo box text as its parameters, deleting the game records from the database.

# Proof of gameplay

I have two instances of the game running at the same time.

|  |  |
| --- | --- |
| Player One | Player Two |
|  |  |
| As we can see in the top left corner, player “daniel” is logged in and is on the first tile. | As we can see in the top left corner, player “steven” is logged in and is also on the first tile. |
| If we look in the database, we can see that they are both on the same tile: | |

|  |  |
| --- | --- |
|  |  |
| As we can see, the user “daniel” has moved to tile 38 as displayed on the left. A message has been sent to the chat displayed on the right. | As we can see, the user “steven” has moved to tile 34 as displayed on the left. A message has been sent to the chat displayed on the right. He has also picked up an item that has been added to his inventory. This has increase the number of items by one and his score is now 5386. |
|  |  |

|  |  |
| --- | --- |
|  | |
| If the player “daniel” tries to move onto the same tile as player “steven,” a message box will display saying that this tile is occupied | |
|  |  |

|  |  |
| --- | --- |
|  |  |
| When the user reaches tile 100, a message box displays saying that they have finished the game. They are then removed from the game | As we can see, user “daniel” is no longer in a game but is still online, whereas user “steven is still in the game” |
|  | Steven is in the game |
|  | Daniel isn’t in the game |

# Design Changes/additions

The wireframes that were created for milestone one was an initial design that were subject to change. As I slowly progressed through the development of my prototype, I made many design changes to make the forms a lot more aesthetic. Although there were not many functionality changes, there were many visual changes to make the forms look more appealing. The initial design of my game only really consisted of the parts of the application that I really needed, rather than going into detail of how the current product looks.

The general theme of my game is a fantasy genre game. Therefore, I wanted to make sure that my design was consistent across all of the forms. Each form was designed with the genre in mind. I wanted to ensure that the background image stood out in each form, this was the reason for making each form large, with lots of space around each of the controls. Where possible, the titles, buttons, labels etc were placed in positions where they would not conflict too much with the image in the background.

## Home Form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I added an image of a castle on the rock with some gif art around it to give a fantasy Dracula like them to the form. This matches the theme of the game that I am going for.
* I added an exit button to the form as my original design did not include one of these. This further increases the functionality of the form. I added dripping blood to the button to further increase the visual appeal as well as representing “killing” the game.
* The overall colour scheme of the form was made a lot darker as it matches the theme better and gives more contrast with the white header text.

## Login Form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* The image I chose is of a knight that has been struck down in a fantasy land.
* I made the overall size of the form to be a lot wider to create a consistency across some of the other forms. This also gives more room to make the background image the centre of attention. I offset the controls so that they would not conflict too much with the background image.
* I gave the text fields default values to let the user know what they need to insert into those fields.
* I changed the exit button from an X to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.

## Sign Up Form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I made the overall size of the form to be a lot wider to create a consistency across some of the other forms. This also gives more room to make the background image the centre of attention. I offset the controls so that they would not conflict too much with the background image.
* I gave the text fields default values to let the user know what they need to insert into those fields.
* I added an exit button to the form as my original design did not include one of these. This further increases the functionality of the form.

## General user lobby

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I added an exit button to the form as my original design did not include one of these. This further increases the functionality of the form. I made it look like a dungeon door to match the theme of the application
* I offset the controls so that they would not conflict too much with the background image.
* I added a small textbox in the bottom left corner that displays your username. This doesn’t really effect the application too much, but was more for knowing that you are correctly logged in and there hasn’t been an issue with the database.

## admin lobby

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I added an exit button to the form as my original design did not include one of these. This further increases the functionality of the form. I made it look like a dungeon door to match the theme of the application
* I included a play button to the form because the administrators might want to play a game as well. The original design only had a leader board button and a settings button.
* I offset the controls so that they would not conflict too much with the background image.
* I added a small textbox in the bottom left corner that displays your username. This doesn’t really affect the application too much but was more for knowing that you are correctly logged in and there hasn’t been an issue with the database.
* Like the previous forms, I chose a background that fits well with the theme of the application. The image I chose creates great contrast with the header which makes more readable. The light blue at the top also contrasts well with the exit button.

## Leaderboard

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I changed the exit button from a house to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* I added a refresh button on the top left of the screen so that the user can see any updates that have been made since the data loaded. This increases the functionality of the form and the validity of the data that the user is viewing.
* I changed the column header text to be less vague. This allows the user to understand the data better
* Like the previous forms, I chose a background that fits well with the theme of the application.
* The general layout of this form didn’t change too much, as I still have a datagridview in the centre of the screen.

## admin settings form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I changed the exit button from a house to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* I changed the game section from being a list box/datagridview to being a combo box. This was to reduce the amount of space that it takes up as well as the fact that it wasn’t really necessary to use a list box.
* I removed the checkboxes for making a user an admin and changing the account status and moved them to the edit player form. This is because they are more relevant to editing data rather than just general management.
* I chose a steam works background that ties in with the them of the application as well as being similar to the idea of it being the settings page.

## Admin add user form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I made the overall size of the form to be a lot wider to create a consistency across some of the other forms. This also gives more room to make the background image the centre of attention. I offset the controls so that they would not conflict too much.
* I changed the exit button from an X to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* I gave the text fields default values to let the user know what they need to insert into those fields.
* I changed the background image to be that of some mystical dragon looking creatures as it matches well with the theme of my application.

## admin edit user form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I made the overall size of the form to be a lot wider to create a consistency across some of the other forms. This also gives more room to make the background image the centre of attention. I offset the controls so that they would not conflict too much.
* I changed the exit button from an X to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* The text in the text fields are set to the values of the user that has been selected in the previous form. This is so that the user has the option to only change one field rather than having to type out every other field. I also moved the account status modifier and admin privilege modifier to this form as it fits with the consistency of modifying a user.
* The header text was changed to black in this form as white would not be as visible on the background that I chose.
* I chose a mystical looking forest for the background of this form as it matches well with the theme of my application.

## game lobby

|  |  |
| --- | --- |
| Before | After |
|  |  |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I changed the exit button from a house to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* I removed the arrow buttons from the player list section and made a single button for when you want to join an active game. This reduces the overall amount of controls that need to be on the form. I also added a text field that displays the game of the user that you have clicked.
* I changed the player list to be a datagridview that displays both the username of the player and the game that they are in. If they are online but not in a game, it will show this field as being empty but still display that they are online.
* I moved the refresh button in the top left of the form and changed it to a refresh arrow. This allows the user to refresh the list of online players so that they can view any modifications to the data without having to reload the form. This means that any new game or game changes can be viewed.
* I changed the game mode section to be a combo box as the original design only displayed a single game mode and did not allow users to change this. I made a set list of different game modes that currently don’t do anything other than change the name of the game, but in future these could be different game types. I could also add an image above the combo box that displays what type of game mode it is.
* Like the game mode, I changed the character selector to be combo box. I changed the character selection from being a colour selector to being a list of character types. The user can select one of these and their character will be updated when the create a new game. This will remain the same character if they join another user. If they want to change characters, they will need to create a new game. Like the game modes, in the future I could add reasons for selecting the characters and add images based on what the character is that the user has selected.
* I added a small textbox in the bottom left corner that displays your username. This doesn’t really affect the application too much but was more for knowing that you are correctly logged in and there hasn’t been an issue with the database.

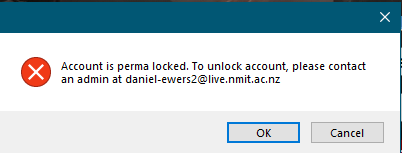
## Game Board Form

|  |  |
| --- | --- |
| Before | After |
|  |  |

* I changed the font of the header text to be old English as it fits more with the theme of the game
* I changed the colour and style of the buttons to create consistency across the forms while matching the theme of the application. I also changed the text font to old English to match the headers
* I changed the border of the form from the thick grey to a fixed 3d border as the thick border significantly reduced the room that I had to work with. It was not aesthetically pleasing and drew too much attention from the purpose of the form. The border is thin and white which creates great contrast between the form and the application in the background.
* I made the overall size of the form to be a lot wider to create a consistency across some of the other forms. This also gives more room to make the background image the centre of attention. I offset the controls so that they would not conflict too much.
* I changed the exit button from an X to a door to fit the theme of the game. The door looks like one that you might find in a dungeon or a castle.
* I added a small textbox in the top left corner that displays your username. This doesn’t really affect the application too much but was more for knowing that you are correctly logged in and there hasn’t been an issue with the database.

## Dialog Messages

I had planned to create custom dialog boxes that would be used for the error messages and confirmations. I decided that it would be better and more efficient to use the dialog result boxes that are built into windows forms. Although these are not as aesthetically pleasing, they are only there to perform a certain task of displaying what is going on.

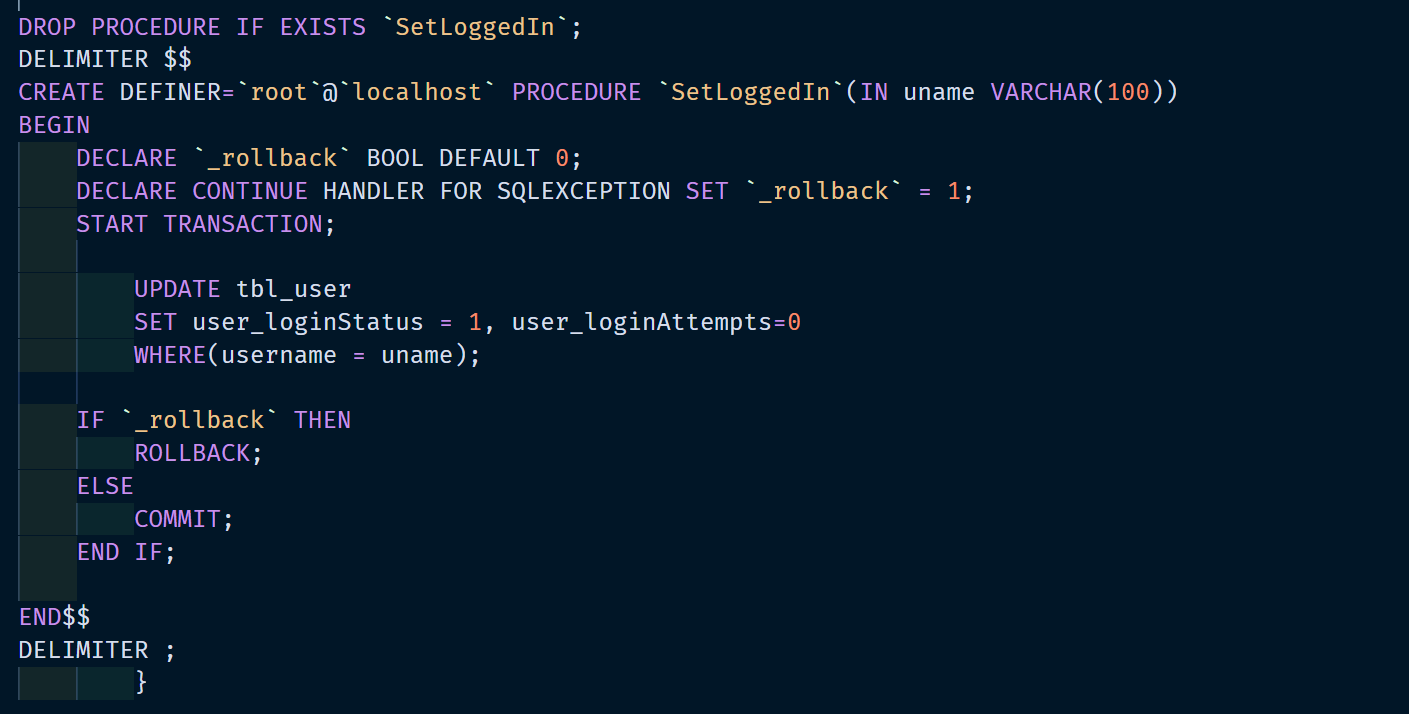
Here is the dialog result box that I have added that I missed in milestone one:  


Below is a list of all of the dialog messages in my application:

* "Account Has Been Created Successfully"
* "Player is not a player. Please select a real player"
* "Game is not a game number. Please select a real number"
* "Are You Sure You Want To End The Game"
* "Game Ended Successfully"
* "You can not delete this player. Try selecting another one."
* "Are you sure you want to delete this user?"
* "User deleted successfully"
* "Player Reset Successful"
* "Are you sure you want to change these details?"
* "Congratulations, you have finished"
* "This tile is occupied"
* "Invalid Move"
* "Please enter a message"
* "Please select a game!"
* "Are you sure you want to join this users game?"
* "This player is not playing a game right now"
* "Please select a character and game mode"
* "Please select a character"
* "Please select a game mode"
* "Are you sure you want to start a new game?"
* "Are You Sure You want to Log Out"
* "Are you sure you want to delete your account?"
* "Account deleted successfully"
* "Password is incorrect or username does not, would you like to create a new account?"
* "Account is permanently locked. To unlock account, please contact an admin at daniel-ewers2@live.nmit.ac.nz"
* "Critical Error"
* "This Username Already Exists, Please Enter a new one"
* "This Username Already Exists, Please Enter a new one try logging in"
* "Please Enter all your details"
* "Your Account Has Been Created Successfully"
* "Game created successfully"
* "Game joined successfully"

# Database Changes/additions

We were informed that we needed to implement transactions into our SQL procedures. Therefore, I have added transactions to each of the procedures. Here is an example of this:



I added additional stored procedures to further increase the functionality of my application.

|  |  |
| --- | --- |
|  | This procedure gets the players score at the current time that it is called. This returns a value that is used to display the player score in the game form as they move their player around the board.  When the score changes in the database, the onscreen value updates as well. |
|  | This procedure gets the players item count at the current time that it is called. This returns a value that is used to display the item count in the game form as they move their player around the board.  When the user collects an item and the item count in the players inventory increases in the database, the onscreen value updates as well. |
|  | This procedure is used to send a message in game. When a user clicks the send message button, the text that is in the textbox is passed to the database and added to the table with the reference to that specific game. |
|  | This procedure is used to retrieve the messages from the database. Currently I have it so that it does it every time the player moves (not ideal, but good for the performance), but in future I would possible add another thread that calls the method every so often for the duration of the game. |

## Additional Changes

* I was informed in my feedback that I needed to indent my SQL properly. Therefore, I went through the code and indented the code where I believe was necessary and wrapped a lot of the SQL so that it flowed better with how it what it was doing.
* I also was informed that I need to make sure that my SQL was consistent with the case that I would use. Therefore, I went through the code and changed the case to be upper case where it was needed (WHERE, IF, SELECT etc). I believe that I got it all but is very possible that I have missed something.

# Gameplay Changes/additions

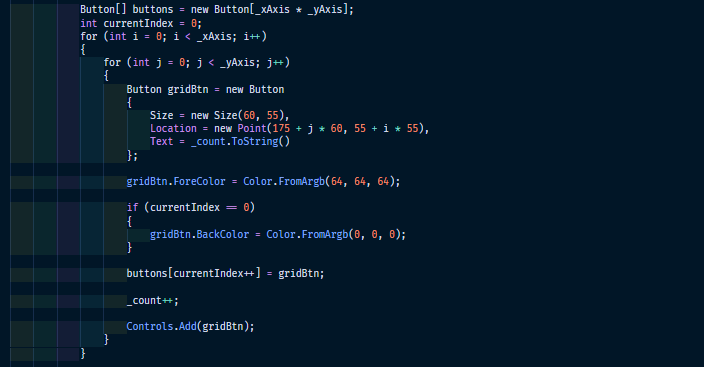
I decided that I like the idea of the game being more of a mystery. Having the items displayed on the grid would make it far too easy for the players to increase their score and collect them. Thus, I decided that I wanted to make the items invisible. This means that the players have to move around to find them without knowing where they are. There are only 3 items on the 10x10 grid which are in a random position every time a game is created. This means that the user cannot learn a pattern of where the items are. I have also made it so that the users cannot see each other, nor directly track whether or not a player has already picked up the items. The only way for them to know if they have found an item is the item count on the left of their screen goes up, as well as their score. A player could move around and never pick up an item and never know that someone else has already taken the items on the board.

The original design showed a circle on the tiles for where your player is, but I decided that I would rather just set the tile colour to be different to all the other tiles. It was not necessary to set them to custom colours as the other players are hidden from you.

The gameplay method was actually rather simple and required far less code than I had expected it to be. Below is the method for moving:



This handles both generates both the button tiles and the click methods. I defined two variables, \_xAxis and \_yAxis. These variables define how many buttons horizontal is displayed vs how many vertical. These can easily be modified to create a far larger or smaller board.



If I had more time, I would likely implement keypress methods that would act as button click events to move the player around. I don’t believe that this would be overly difficult to achieve but is highly possible.

I made it so that the user can move up, down, left, or right. If they click anywhere other than this, it throws an error saying that it is an invalid move. If the user clicks a tile that is occupied, it will not update their position front end or backend and throw and error saying that the tile is occupied.

Rather than designing the board as a maze, I decided that I would create it as an open board where the player is free to move in any direction as long as it is a valid movement. This was to help prevent users from blocking paths in the maze (diagram one below). This still poses an issue with my current design that players can stop others from finishing the game. Currently users can access the home tile but can be blocked from reaching the final tile if users surround it (diagram two below). In future, I might add a method that times how long a user is idle. IF they are sitting still for more than 30 seconds or so, they will be kicked from the game.

|  |  |
| --- | --- |
| 1 | 2 |
| A user (red) is trying to move through the maze, but the pathways are being blocked by the other users (blue). | A user (blue) is trying to reach the finish tile (green) but is blocked by the other players (red). |

# Update to ACID Section

## How does MySQL implement ACID?

MySQL contains multiple layers of protection to help support durability. As stated by Blum (2018), “MySQL writes all transactions to a log file, writes the changes to the double-write buffer area, and then writes them to the actual database files. If the system crashes during this process, most of the time MySQL can recover the transaction within the process.”

The way MySQL handles Atomicity is by storing the transaction results in a memory buffer. These results are written to disk and the binary log once the transaction is committed successfully. To handle consistency, MySQL records any changes to the database and provides an audit trail for transaction recovery. As stated by Vaswami (2015), “In addition to the logging process, MySQL provides locking mechanisms that ensure that all of the tables, rows, and indexes that make up the transaction are locked by the initiating process long enough to either commit the transaction or roll it back.”

As stated earlier, MySQL logs the modifications made to the database. This also enhances the longevity of the database as we can see retrieve these logs reasonably quickly if there are any difficulties such as a system crash or hardware failure. “By default, InnoDB tables are 100 percent durable (in other words, all transactions committed to the system before to the crash are liable to be rolled back during the recovery process,” according to Vaswami (2015).

### Related MySQL atomicity features include:

* The autocommit setting.
* The COMMIT statement.
* The ROLLBACK statement.

(MySql, n.d.)

### Related MySQL Conistancy features include:

* The InnoDB doublewrite buffer
* InnoDB crash recovery

(MySql, n.d.)

### Related MySQL Isolation features include:

* The autocommit setting.
* Transaction isolation levels and the SET TRANSACTION statement
* The low-level details of InnoDB locking.

(MySql, n.d.)

### Related MySQL Durability features include:

* The InnoDB doublewrite buffer
* The innodb\_flush\_log\_at\_trx\_commit variable.
* The sync\_binlog variable.
* The innodb\_file\_per\_table variable.

(MySql, n.d.)

# References

Blum, R. (2018, June 14). *Making Sure Your MySQL Database is ACID Compliant*. Dummies. https://www.dummies.com/programming/php/making-sure-mysql-database-acid-compliant/

geeksforgeeks. (2021, April 7). *ACID-Properties* [Diagram]. Https://Www.Geeksforgeeks.Org/Acid-Properties-in-Dbms/. https://media.geeksforgeeks.org/wp-content/cdn-uploads/20191121102921/ACID-Properties.jpg

GeeksforGeeks. (2021, April 7). *ACID Properties in DBMS*. <https://www.geeksforgeeks.org/acid-properties-in-dbms/>

MySql. (n.d.). *MySQL :: MySQL 5.6 Reference Manual :: 14.2 InnoDB and the ACID Model*. Retrieved June 23, 2021, from <https://dev.mysql.com/doc/refman/5.6/en/mysql-acid.html>

Vaswami, V. (2015, September 24). *MySQL and the ACID Properties*. SolarWinds. https://logicalread.com/mysql-acid-properties-mc13/#.YNKOZOgzYuU