2021

Project Report Milestone 3

DAT602

DALE STEPHENSON

Table of Contents

1.0 Milestone One

1.1 Game Description

The development of the game initially seeks to establish the business and functional requirements as described in the project brief. A database must be developed that allows for the creation of a multi-user, 2D 'point-and-click' game application based on a tiled map layout. A prototype will be developed that will test the concept. Requirements to meet include:

Business Requirements

- Auto registration of new players on the login input screen
- Registration requires password input
- New players are displayed as online to indicate availability for gameplay
- If a username exists in the database a player has five tries to submit the correct password at login
- Administrator accounts used for unlocking 'locked' accounts, end gameplay, add new players, update existing player data and remove players
- Two or more players play against each other in each game instance
- Multiple games can operate concurrently
- Players can start a new game that other players can join
- Players start on a defined 'home tile'
- Only one player can be on a tile at any one time except for the 'home tile'
- Players move around the board to a tile adjoining the current tile
- Players who click on an empty tile first, move to that tile
- Tiles contain items that are collected by players
- Items gain players a varying amount of points
- If a player logs out of the system, their last position is stored in the database
- When a player returns to a game their location and state is restored
- If a player returns and another player is in their last location, they must choose one of the adjacent tiles
- Players collect items as assets that increase their score in the game instance
- Administrators can delete their player account
- Players can also be administrators

Functional Requirements:

- User registration requires the player to log in after the account is established
- New players are added to a table in the remote database
- The system must check if the user already exists
- Prompt for password on user login if username found in database
- If five login attempts are incorrect system locks the account
- Player turns are stored on the database as live gameplay occurs
- The system displays all players currently online and their high scores
- The system must track players, their current location and the assets they have collected
- The system must record the location of assets or the player that has collected the item

1.1.1 Design Concept

The game will reference and be based around the famous 'Seven Dwarfs' story, which has the benefit of creating a starting point for some system rules. The design concept limits each game to a maximum of seven players, the 'items' players collect are 'gems' that have been lost by the seven dwarfs on their way home from the mines.

To make collecting the gems more fun each player will be assigned a character which is one of the seven dwarfs, they will compete against one another to collect the available gems on the board.

Each gem item has points assigned to it based on the gem type; however, this is unknown to the players meaning they have to match the gems to points as they move around the board. Players will have to work out and remember which gems offer the greater points as they progress through the game, this will help them to maximize their high scores by strategically collecting gems.

Players are taking a risk when they land on a tile as many tiles will have multiple gems to choose from, and other tiles have none. If they have a choice of gems, they can either choose one that they know its number of points, or risk selecting another which may, or may not have higher points.

The game is over when the system calculates that all the gems have been collected.

1.1.2 Login Process

When accessing the game a player is presented with a login screen that allows them to enter their username and password. The user inputs their username and password and clicks 'login', at this point the database is checked to confirm the username and password entered and that they match.

If the username is not found a message is displayed asking them to register as a new user. If the username is found but the password is incorrect a message is displayed stating the wrong password was input and to try again, the player has a maximum of five attempts before a message is displayed stating their account is locked.

If the player is new to the game they can click to register as a new user. On this screen, they can type in their email and select a username and password. The email and username are checked against the database, if they are already taken a message is displayed asking them to try a different email or name.

1.1.3 Game Home

Once a successful login is confirmed players are directed to the games 'home screen', which displays navigation buttons for the 'home screen', 'admin screen' and 'logout'. Also, the screen displays the open games and the number of players partaking in the games as well as a list of available players including their high scores.

If the player clicks on the admin button the database is checked to confirm they have admin privileges, if they do they are directed to the admin screen. If they do not have admin rights a message is displayed stating they are not administrators.

A player could choose to click on a game with fewer than seven players and join the existing game, or they might choose to create a new game and wait for other players to join. If a player clicks on a game that already has seven players a message is displayed stating they must select a different game.

1.1.4 Game Objective

Players can see other player information displayed around the game board, players will be assigned to one of the seven dwarf characters which relate to a colour on the board. The home tile is always black, if a player is not visible it is because they are either on the home tile or are not logged in.

The objective is to move around the board in the most efficient way possible, by tracking other players as their points increase. Identifying select tiles where other players points increase players can start to understand where gems may be located, also, players must remember which gems gain them the highest points.

When a player moves to a tile either a message is displayed stating that tile has no gems, they should try and remember this so they don't move to it again, or list the gems available to collect. Other players can track your movement and see that tile gained you no points. This could also be a bluff, there might be some 'low hanging fruit' gems that you want to come back and collect later and don't want other players to guess. If there are gems on the tile you select, the found gems screen is displayed where each gem on the tile is listed. You can select the gem you want which is then added to your inventory and you get allocated the associated points.

If a player tries to move to a tile not adjacent to their current tile a message is displayed, likewise, if they try to move to a tile that already has a player, they will be asked to select a different tile.

The player with the most points and not the biggest stash of gems is the game-winner. Each losing player gets notified of the winner, the winner is notified they have won and the game ends.

1.1.5 Administrators

Players can also have administrator privileges. When in the game home screen administrators click the admin button, their access level is checked against the database and if true, the admin screen is displayed. The admin function allows these players to 'kill a game', which ends the game. Before the game being killed a message is displayed which acts to confirm the admin request.

Admin functionality also includes the ability to add a new user or update the details of an existing user. The create user screen requires the input of similar data to the new registration screen, but with additional admin functionality. Likewise, with the new registration screen, if the username or email is checked against the database as existing the administrator will be asked to select alternatives. The update user screen contains the same input fields as the add user screen, but includes the data related to a player, this can be amended by the administrator. To open the update user screen a player must first be selected.

To remove a player first the player must be selected from the list, before the user being deleted a message is displayed as a secondary security measure requesting confirmation of the deletion.

1.2 Storyboards

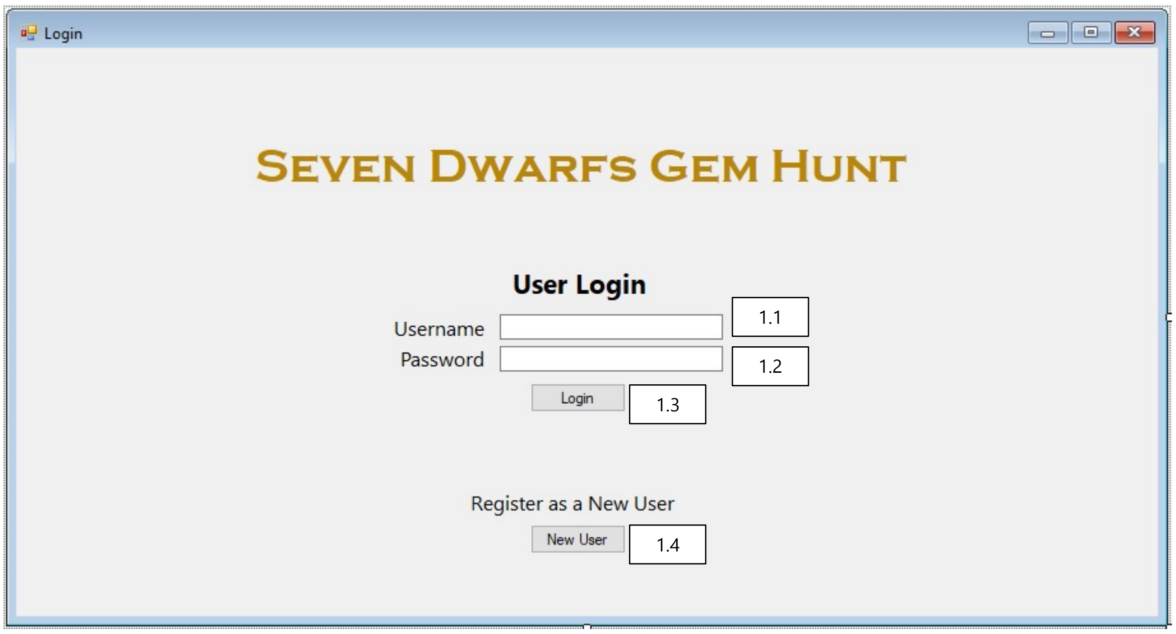
Understanding the game design concept, in addition to the requirements, allows attention to be given to the development of storyboards. Storyboards help development teams to understand the human-computer interaction (HCI), which is important for both useability and to create an effective design for the graphical user interface (GUI).

Creating storyboards allows the design to be reviewed by stakeholders and users early in the design process, so feedback can be given, and changes made before expense has occurred in creating the deeper analysis and design elements or any physical build and programming.

The purpose of storyboarding is to develop a set of images that can be understood quickly by both technical and business users, these are not manuals so shouldn't require lots of reading or an understanding of complex development terminology. The project team should consider:

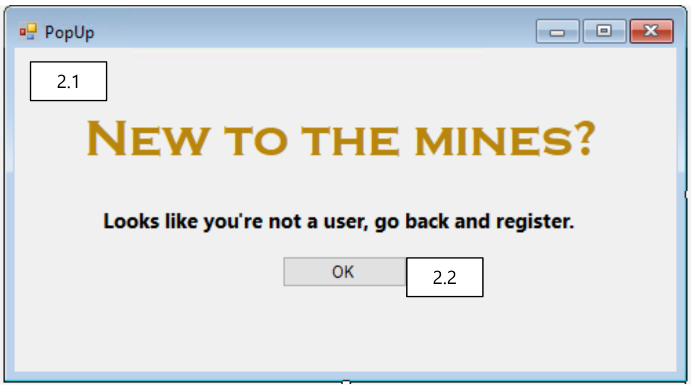
- The ease of screen accessibility might be achieved through a menu function that allows users to quickly return to the main menu
- Consistency in screen layout and design including any action buttons or tabs
- Consider white space so items are easy to locate
- Meaningful and easy to understand screen titles
- Consideration made to font style, size, colour, and the appearance of the background to ensure that information is readable by the intended user
- Clear labelling of control or command buttons with appropriate labelling of symbols and images
- If controls or commands are not meant for general users, then access should be denied
- Measures to mitigate the entry of invalid data through quality and validation controls

Storyboard 1 – Login



1.1 Textbox for the player to enter a username, checked against the database
1.2 Textbox for the player to enter a password, checked against the database
1.3 Login button, if database finds correct username and password then move to storyboard [7], if the username is not correct move to storyboard [2], if the password is not correct move to storyboard [3], if the password has been attempted incorrectly 5 times move to storyboard [4]
1.4 New User button, for new players that want to register, move to storyboard [5]

Storyboard 2 – Not a registered user

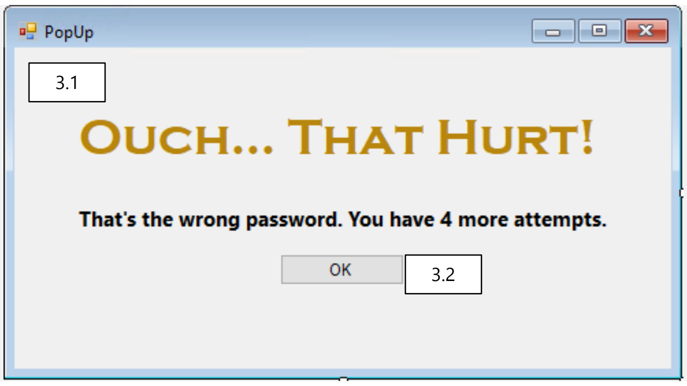


NEW TO THE MINES?

Looks like you're not a user, go back and register.

2.1 Dialog window appears if the username is not recognised
2.2 OK button to go back to log in, move to storyboard [1]

Storyboard 3 – Wrong password

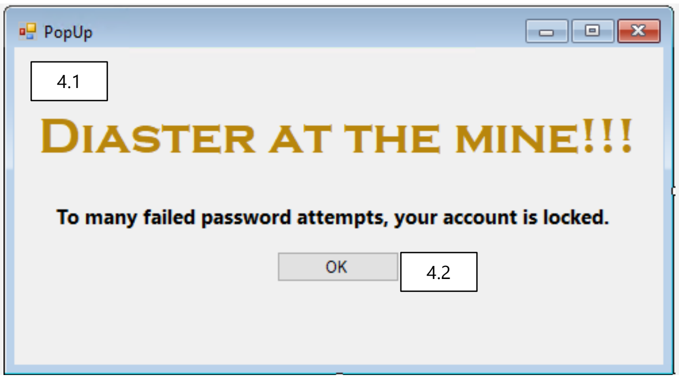


OUCH… THAT HURT!

That's the wrong password. You have 4 more attempts.

3.1 Dialog window appears if the incorrect password entered
3.2 OK button to go back to log in, move to storyboard [1]

Storyboard 4 – Locked account

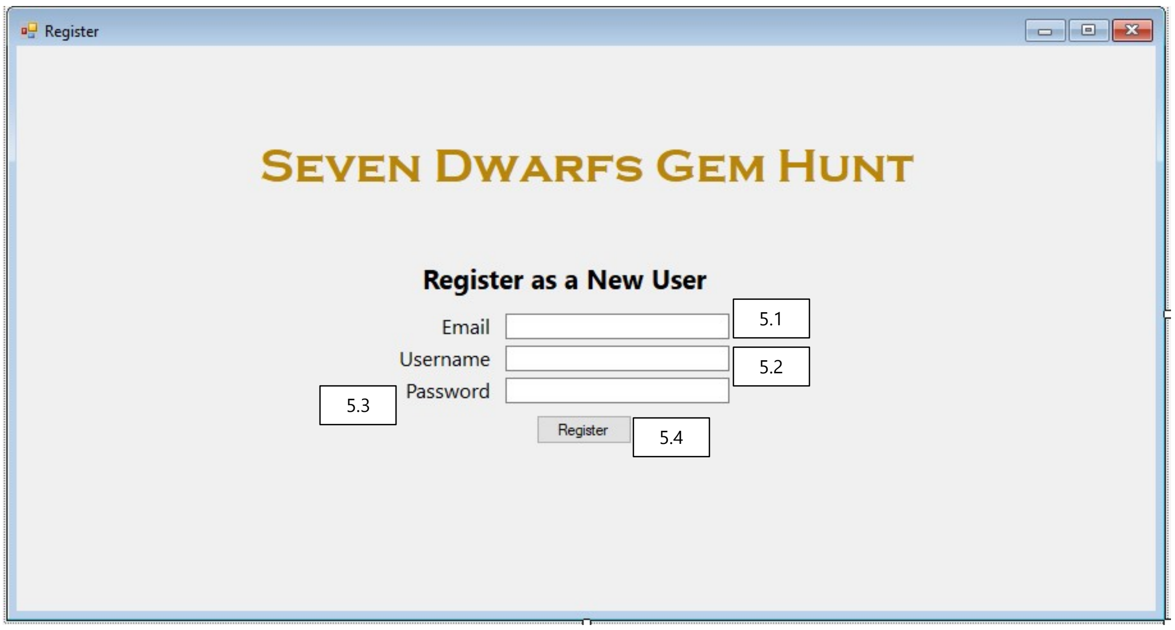


DIASTER AT THE MINE!!!

To many failed password attempts, your account is locked.

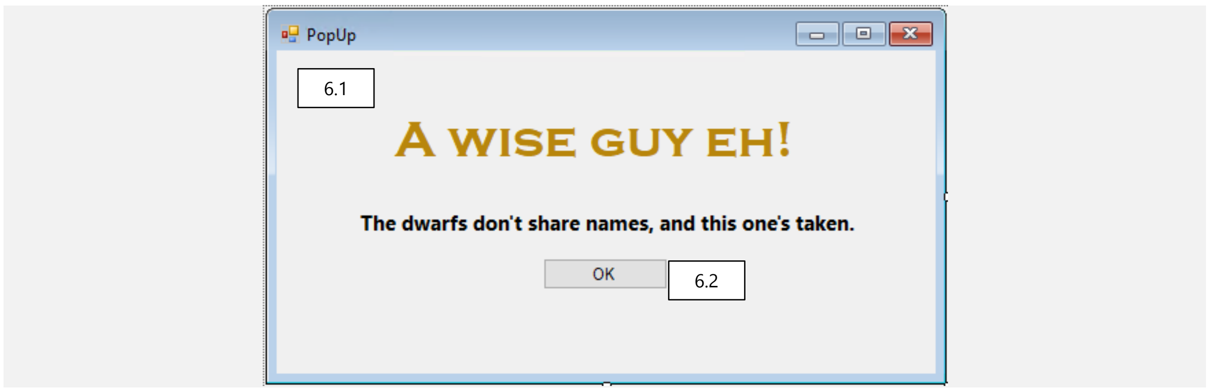
4.1 Dialog window appears if the incorrect password entered 5 times
4.2 OK button to go back to log in, move to storyboard [1]

Storyboard 5 – Registration



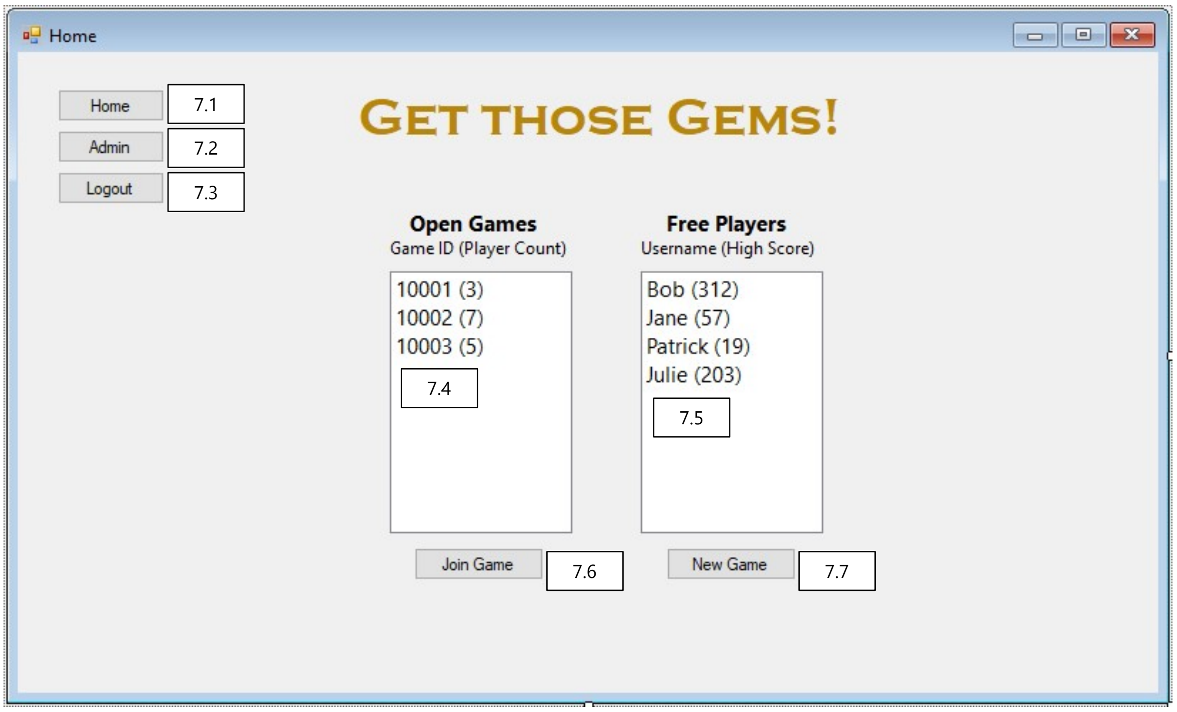
5.1 Textbox for the player to new enter email, checked against the database
5.2 Textbox for the player to new enter username, checked against the database
5.3 Textbox for the player to new enter password
5.4 Register button, if the database doesn't find a conflict with the email and username then move to storyboard [1] to log in, if email or username is already taken then move to storyboard [7]

Storyboard 6 – Username taken



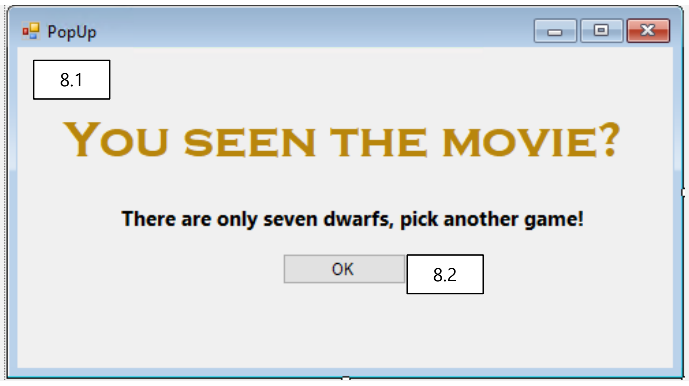
6.1 Dialog window appears if conflict found with email or username
6.2 OK button to go back to the registration screen, move to storyboard [5], if the new user is being created by admin go back to add user screen, move to storyboard [21], if the updated user has had their email or username updated by admin go back to update user screen, move to storyboard [21]

Storyboard 7 – Game home



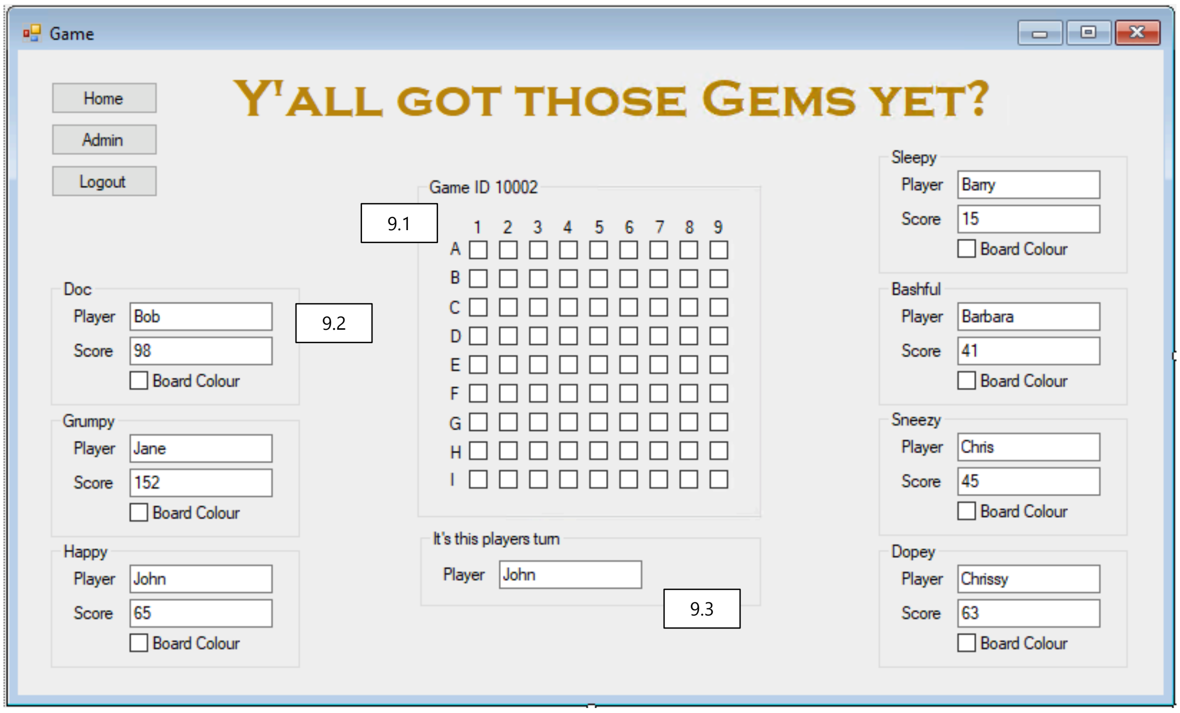
7.1 Home button, navigation for the game home screen
7.2 Admin button, navigation for admin screen, check the database for admin privileges, if found, move to storyboard [19], if the player does not have admin privileges move to storyboard [18]
7.3 Logout button, move to storyboard [1]
7.4 List box displaying all current games and the number of players playing, games can be selected, and a request made to join the game
7.5 List box displaying all current players and their high scores
7.6 Join game button, once a game has been selected player can join a game, move to storyboard [10], if the game already has the maximum number of players move to storyboard [8]
7.7 New game button, creates a new game that other free players can join, move to storyboard [9]

Storyboard 8 – The game has enough players



8.1 Dialog window appears if the player tries to join a game that has its player limit
8.2 OK button to go back to game home, move to storyboard [7]

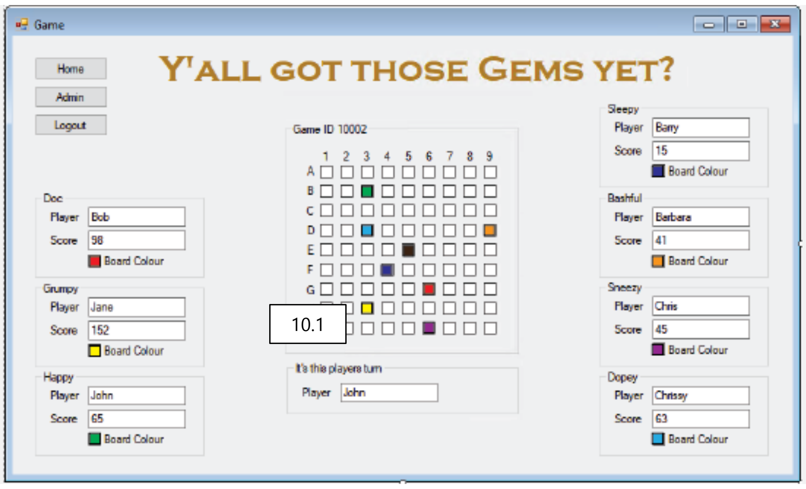
Storyboard 9 – Gameboard



Doc — Player: Bob — Score: 98
Grumpy — Player: Jane — Score: 152
Happy — Player: John — Score: 65
Sleepy — Player: Barry — Score: 15
Bashful — Player: Barbara — Score: 41
Sneezy — Player: Chris — Score: 45
Dopey — Player: Chrissy — Score: 63
It's this players turn — Player: John

9.1 Playing grid, if it's a player's turn, they can click on an adjacent tile, if it's not a player's turn move to storyboard [15], if they don't click on an adjacent tile move to storyboard [14], if there is a player already on the tile move to storyboard [13], if there are no gems on the tile move to storyboard [12], if there are gems on the tile move to storyboard [11]
9.2 Current players information box, shows the player name, current game score and board colour
9.3 Display which players turn it is

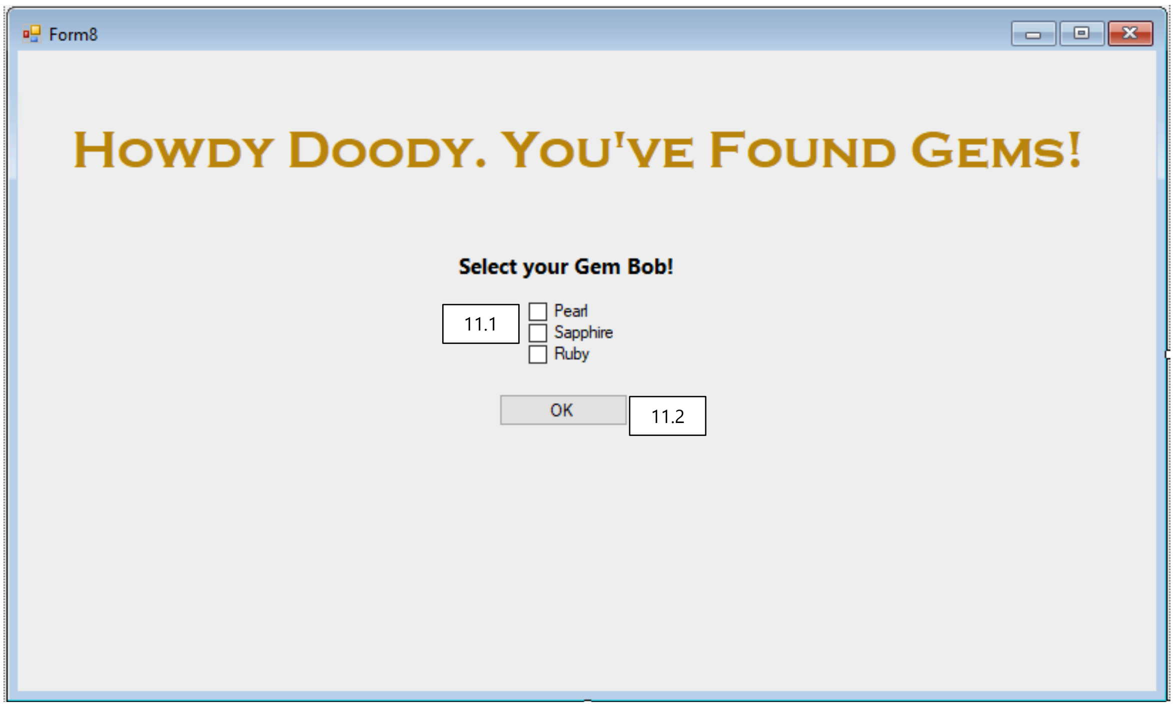
Storyboard 10 – Gameboard in play



Doc — Player: Bob — Score: 98
Grumpy — Player: Jane — Score: 152
Happy — Player: John — Score: 65
Sleepy — Player: Barry — Score: 15
Bashful — Player: Barbara — Score: 41
Sneezy — Player: Chris — Score: 45
Dopey — Player: Chrissy — Score: 63
It's this players turn — Player: John

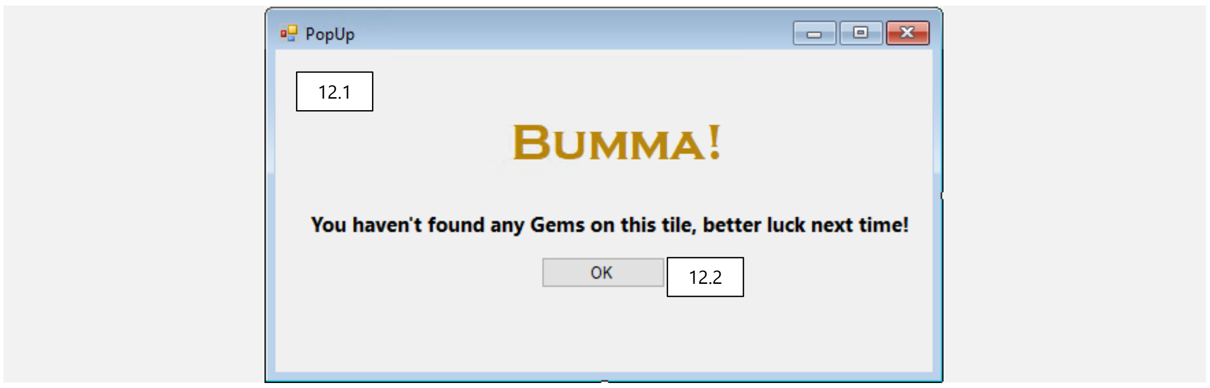
10.1 Playing grid after a game has begun showing players identified by colour on the game grid, the home tile stays black, if a colour is not on the screen then the player has either logged off or is on the home tile

Storyboard 11 – Gems found



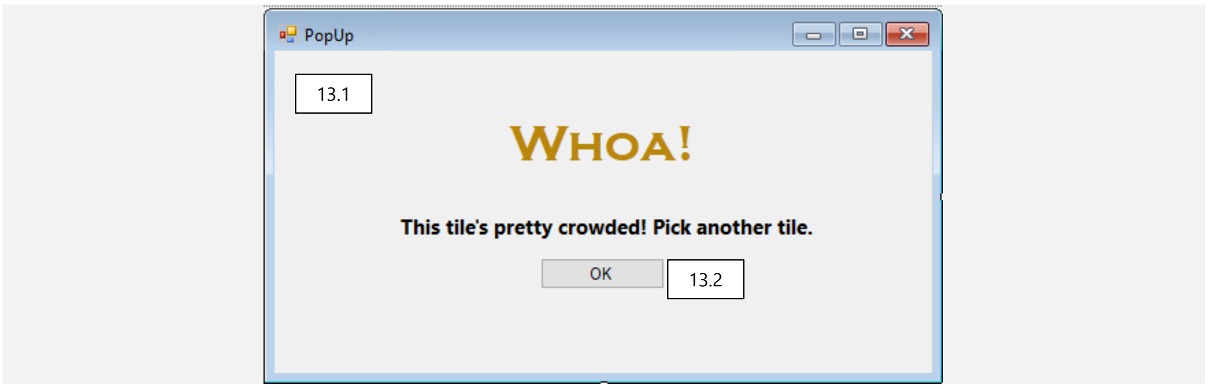
11.1 Player can click to select one gem from the gems located on the tile and their points are updated in the gameplay
11.2 OK button to go back to gameboard in play, move to storyboard [10]

Storyboard 12 – No gems found


PopUp
12.1
Bumma!
You haven't found any Gems on this tile, better luck next time!
OK
12.2

12.1 Dialog window appears if the player moves to a tile with no gems
12.2 OK button to go back to gameboard in play, move to storyboard [10]

Storyboard 13 – Player on tile

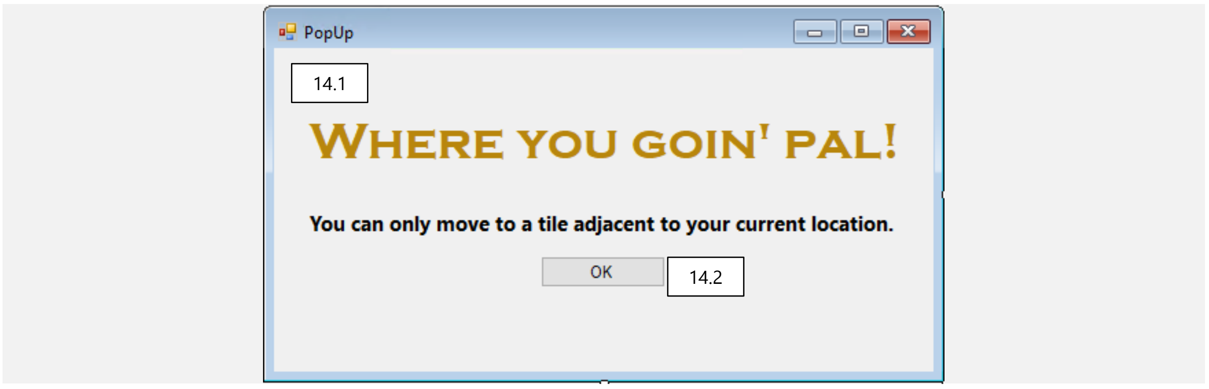


WHOA!

This tile's pretty crowded! Pick another tile.

13.1 Dialog window appears if the player moves to a tile occupied by another player
13.2 OK button to go back to gameboard in play, move to storyboard [10]

Storyboard 14 – Tile is not adjacent

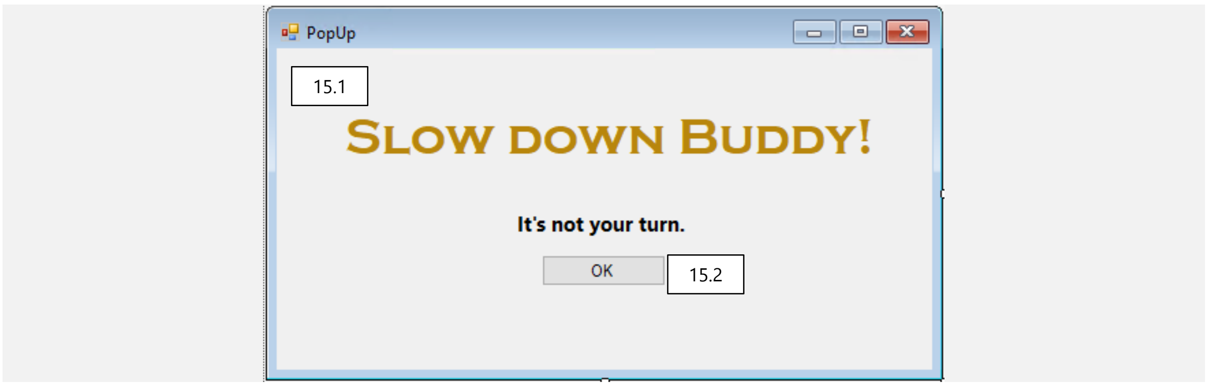


WHERE YOU GOIN' PAL!

You can only move to a tile adjacent to your current location.

14.1 Dialog window appears if the player tries to move to a tile not adjacent to their current tile
14.2 OK button to go back to gameboard in play, move to storyboard [10]

Storyboard 15 – Not players turn

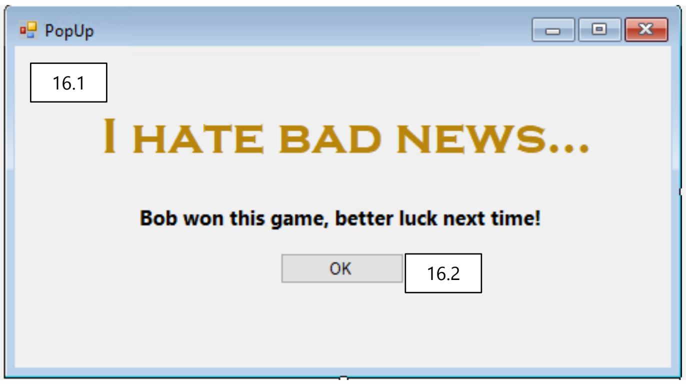


SLOW DOWN BUDDY!

It's not your turn.

15.1 Dialog window appears if the player tries to move to a tile when it is not their turn
15.2 OK button to go back to gameboard in play, move to storyboard [10]

Storyboard 16 – Game lost

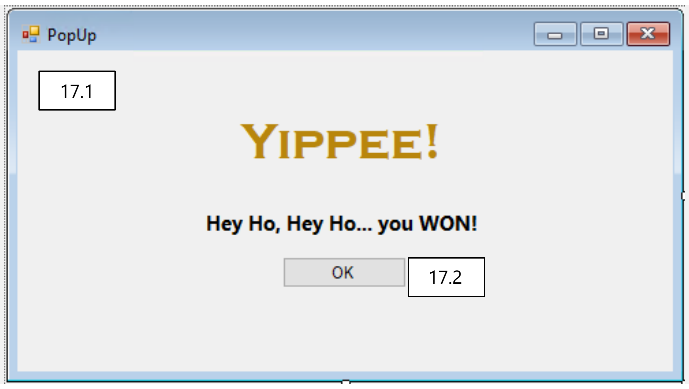


I HATE BAD NEWS...

Bob won this game, better luck next time!

16.1 Dialog window appears if another player has won the game after all gems are collected
16.2 OK button to go back to game home, move to storyboard [7]

Storyboard 17 – Game won

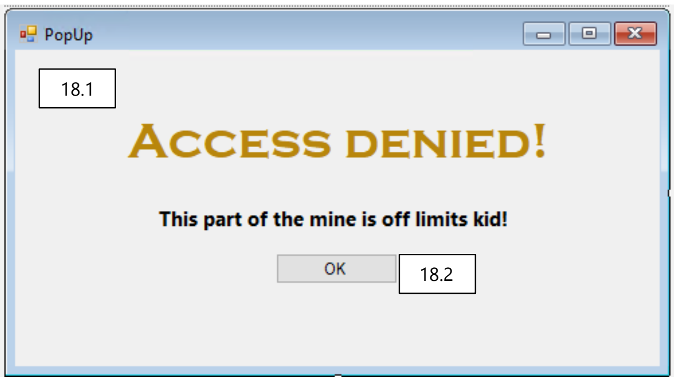


YIPPEE!

Hey Ho, Hey Ho... you WON!

17.1 Dialog window appears if the player has won the game
17.2 OK button to go back to game home, move to storyboard [7]

Storyboard 18 – Access denied

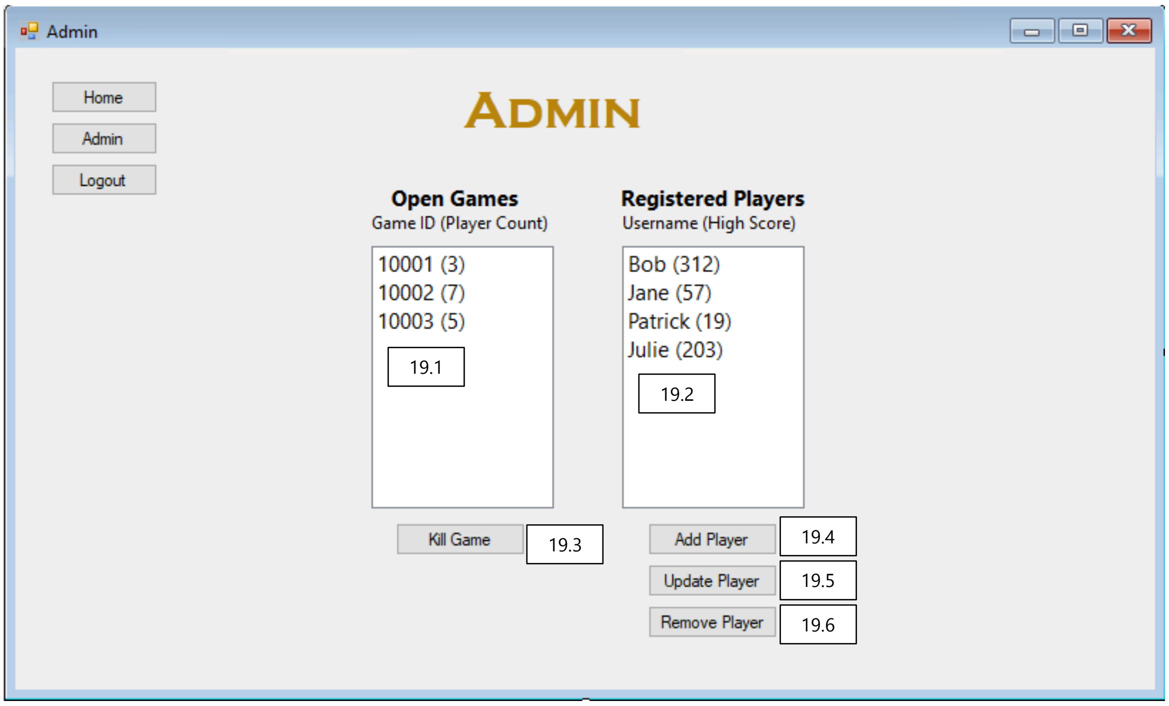


ACCESS DENIED!

This part of the mine is off limits kid!

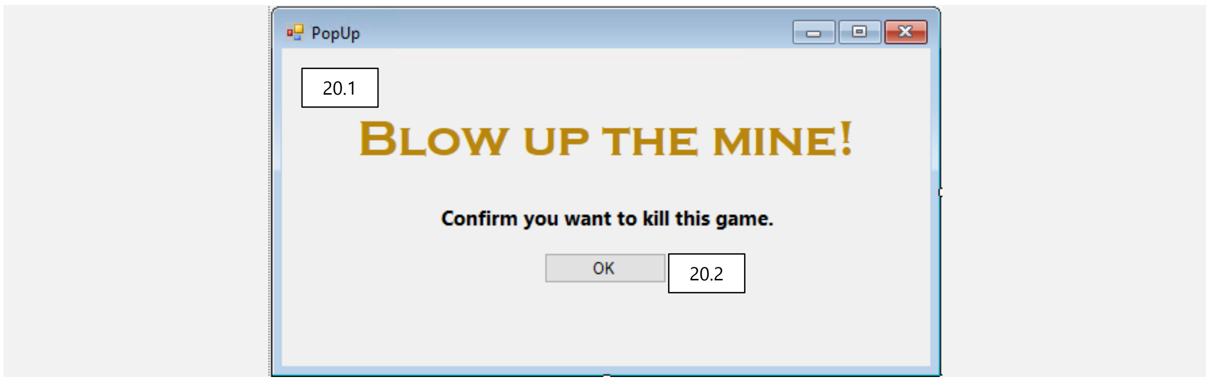
18.1 Dialog window appears if the player attempts to access the admin area and doesn't have admin privileges
18.2 OK button to go back to game home, move to storyboard [7]

Storyboard 19 – Admin home



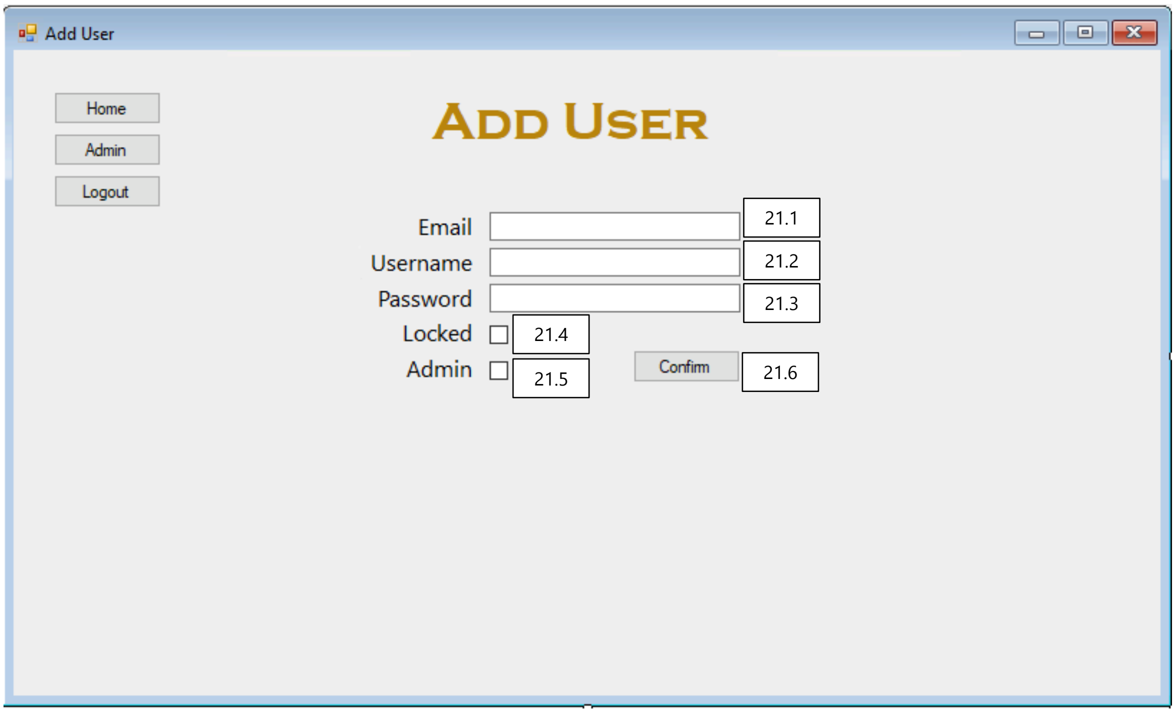
19.1 List box displaying all current games and the number of players playing, games can be selected, and a request made to kill the game
19.2 List box displaying all players and their high scores, players can be selected, and their details updated, or their account deleted
19.3 Kill game button, once a game is selected the kill game button end the game, move to storyboard [20]
19.4 Add player button, click to add a new player, move to storyboard [21]
19.5 Update player button, click to update a player's details, move to storyboard [22]
19.6 Remove player button, click to delete a player account, move to storyboard [23]

Storyboard 20 – Admin: Kill game warning



20.1 Dialog window appears if the administrator requests to kill a game
20.2 OK button kills the game and takes the user back to admin screen, move to storyboard [19]

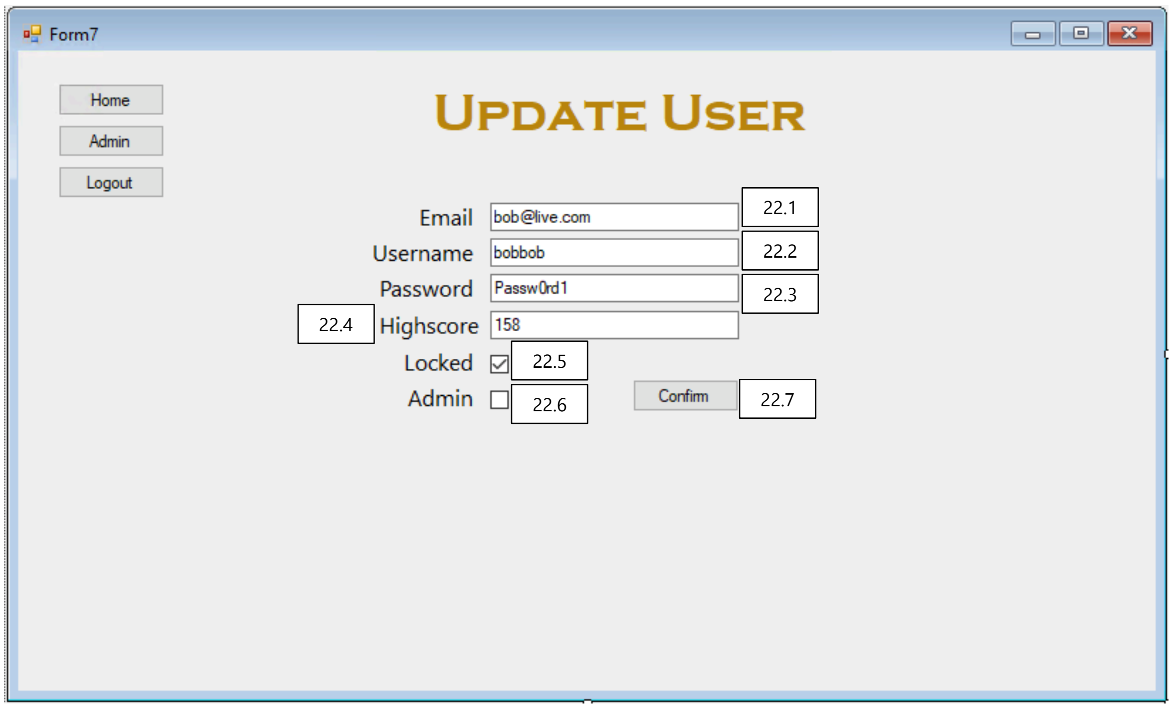
Storyboard 21 – Admin: Add user



Email	
Username	
Password	
Locked	☐
Admin	☐

21.1 Textbox to enter new player email, checked against the database
21.2 Textbox to enter a new player username, checked against the database
21.3 Textbox to enter new player password
21.4 Tick button to set the account as locked
21.5 Tick button to assign the account with admin privileges
21.6 Confirm button, checks with the database to ensure email and username are free, if not move to storyboard [6], if they don't exist go back to the admin page, move to storyboard [19]

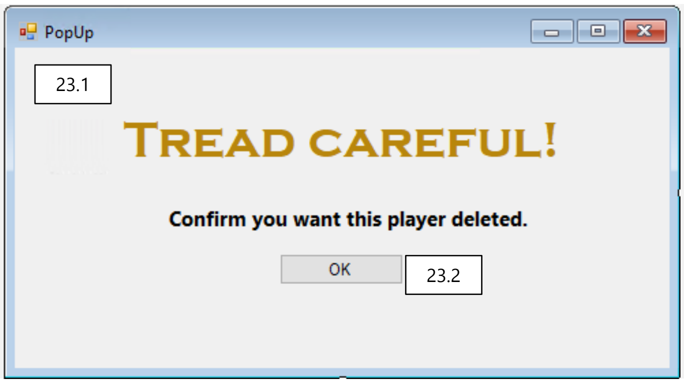
Storyboard 22 – Admin: Update user



Email	bob@live.com
Username	bobbob
Password	Passw0rd1
Highscore	158
Locked	☑
Admin	☐

22.1 Textbox displaying player email, checked against the database
22.2 Textbox displaying player username, checked against the database
22.3 Textbox displaying player password
22.4 Textbox displaying the high score
22.5 Tick button to set the account as locked
22.6 Tick button to assign the account with admin privileges
22.7 Confirm button, checks with the database to ensure email and username are free, if not move to storyboard [6], if they don't exist go back to the admin page, move to storyboard [19]

Storyboard 23 – Admin: Login Screen



TREAD CAREFUL!

Confirm you want this player deleted.

23.1 Dialog window appears if the administrator requests to remove a player
23.2 OK button deletes player and takes the user back to admin screen, move to storyboard [19]

1.2.4 Screen Design Rationale

The storyboards provide a general layout and basic screen structure for the prototype game in development. It is expected that after client feedback and prototype testing the screens will undergo changes and development to ensure requirements are met as closely as possible. Screens can often undergo further changes once a system has gone live, performance monitoring and useability metrics can inform the design and lead to improvements.

Each screen has a formal heading in the display surround, whilst the display windows have a whimsical heading befitting the fun and jovial nature of the game. The display window headings are specifically designed to relate to the seven dwarfs mining activities and famous song from the movie.

The storyboard screens have been designed using Visual Studio and have been designed utilising the tools available within the software package. These screens are prototype only and later incarnations of design would more closely align to the needs of the client.

1.2.4.1 Login & Registration

The login and registration screen both follow a similar layout and format. In this instance, the screens have been designed to be separate, which aligns with the industry standards experienced with most online systems. The login screen only contains information to allow for login to the system or to set up a new account. The registration page is similar in that only the required functions are displayed to set up a new user. Once again industry norms have been applied to this screen.

1.2.4.2 Game Home

The game home screen performs two main functions, acting as a basic secondary navigation area to allow users to move around the system and the main function comprising of the game information area.

Users can see the active games and choose to join games that have not hit the player limit, as well as current players active in gameplay. The user can choose, should they not want to join an active game, to start a new game that other players can join.

1.2.4.3 Game Play

The gameplay area is designed to focus the player on the tile board where the activity takes place, surrounding this is the navigation area and text boxes along each side displaying all the players currently active on the particular game instance.

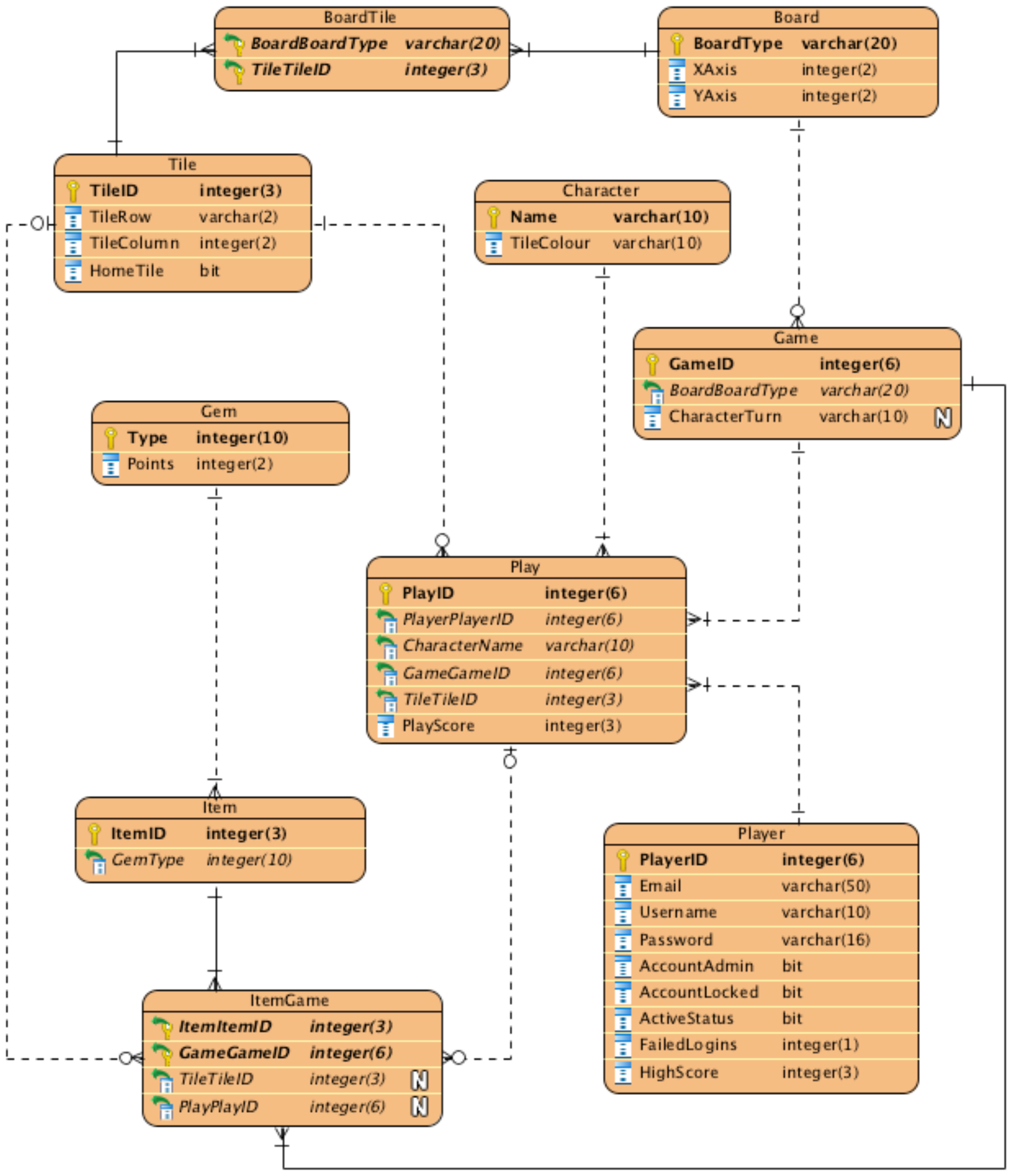
The board is comprised of a 9x9 tile grid which allows the central grid, 'E5', to be designated the home tile. The game ID is indicated at the top of the tile grid. At the bottom of the grid, there is a display indicating which player has the current turn. White space has been created to clearly define the areas and allow for changes and updates later in development.

1.2.4.4 Administrators

The admin screen was based on the game home screen to ensure consistency and allow for reduced development time. The administrator can view all current games and the number of players and all registered players and their high scores across two text boxes. This information is required to meet the business requirements and for administrators to kill games, and update and remove users.

Additional functionality allows the administrator to add a new player to the system. Some of this functionality can have a significant impact on the database, such as removing a player, so warning dialogue boxes are displayed when carrying out certain functions.

1.3 Entity Relationship Diagram (ERD)



1.3.1 ERD Rationale

The relational (logical) modelling diagram is developed to both improve the system and how it is visualised and communicated. The relational model details the events that take place and lists the data required. Further benefits from producing the relational diagram are that it builds the foundations for the creation of the physical diagram.

Logical diagrams are used to allow for a better understanding of the database by the non-technical business team. This is extremely beneficial to the development team as it allows them to involve the business users in the process. This is useful because the business team can be more capable of rooting out errors made in the assumptions, in addition to identifying any shortcomings within the database design from a business perspective.

1.3.4.1 Player, Play, Game

A player can play one or many games, and a game can be played by one or many players. To meet the rules of the relational model and normalisation, a join table establishes the relationship between them. The play table takes the player ID foreign key and game ID foreign key in the play join table. The play table stores data relating to each player or user of the system currently active in the gameplay.

1.3.4.2 Player, Play, Character

One player plays one or many characters, characters are played by one or many players meaning a join table is required in a relational database. The character and player foreign keys are entered into the play table.

1.3.4.3 Player, Play, Tile

At any point in the game, the player is located on a tile, one player can be located on one tile at any time, a tile can contain zero or one players. The relationship between the player and their tile on the particular game instance is stored in the play table. The join table already establishes a game and player relationship and adds the current tile location to the table. They ensure that the relationship between the player and tile location is stored against the correct game instance.

1.3.4.4 Tile, Item/Game, Play

Each item on the board must be located on a tile for it to be found by the players, the specific tile location is stored and is a nullable field. The field must be nullable as the item will not have a tile location when collected by a player. A tile can contain zero or many items and an item can be located on zero or one tile.

Once an item has been collected by a player the relationship between the item and the tile is removed from the database, in its place, a relationship with the player is established by updating the play ID field in the item table obtained from the play table. The play table is the instance of the player with a relationship with that game instance because a player can play one or many games.

1.3.4.5 Board, Board/Tile, Tile

One board has many tiles and tiles can be located on many boards, this many to many relationship requires a Board/Tile join table that establishes the relationship between board and tile. This join table satisfies normalisation in a relational database. Boards form the basis of each game instance. In this development stage for prototype development, there is only one board type established as a 9x9 grid.

1.3.4.6 Item, Item/Game, Game

Each item can exist as part of one or many games, each game can contain one or many items. To satisfy normalisation a join table is created between the game table and the item table. The join table takes the primary key of each table as foreign keys, together they make up the primary key for the Item/Game table. The item instance is related to many games and either located on a tile or associated with a play record.

1.4 CRUD Table

Entity/Attribute	Login Check Credentials	New User Registration	Failed Login Account Lock	Login Successful	New Game	Join Game	Player Moves	Player Finds Gem	End Game	Player Logout	Enter Admin Screen	Admin Kill Game	Admin Add Player	Admin Update Player	Admin Remove Player
Player	RU	CR	R	R	R	R	R	RU	RU	RU	R	RU	CR	RU	RD
PlayerID	R	C	R						R	R	R	R	C	R	RD
Email		CR											CR	RU	D
Username	R	CR		R	R	R	R	R	R		R	R	CR	RU	D
Password	R	C											C	RU	D
AccountAdmin		C		R							R		C	RU	D
AccountLocked	R	C	U										C	RU	D
ActiveStatus		C		U						U			C	RU	D
FailedLogins	RU	C	U	U									C	RU	D
HighScore		C		R				RU		RU	R		C	RU	D
Game				R	C	RU	RU	R	RU	RU	R	RU			RU
GameID				R	C	R	R	R	R	R	R	R			R
BoardBoardID					C	R									
CharacterTurn				U	C	RU	U		U	U		U			U
Board					R	R	R								
BoardType					R	R	R								
XAxis					R	R									
YAxis					R	R									
Tile					R	R	R	R							
TileID					R	R	R	R							
TileRow					R	R	R	R							
TileColumn					R	R	R	R							
HomeTile					R	R	R								
BoardTile					R	R	R								
BoardBoardID					R	R	R								
TileTileID					R	R	R								
Character					R	R	R	R							R
CharacterName					R	R	R	R							R
TileColour					R	R	R								
Gem					R	R		R							
Type					R	R		R							
Points					R	R		R							
Item					R	R		R							
ItemID					R	R		R							
GemType					R	R		R							
Play				R	C	C	RU	RU	R	R	R	R			RD
PlayID				R	C	C	R	R							D
PlayerPlayerID					C	C			R	R	R	R			RD
CharacterName					C	C									D
GameGameID				R	C	C			R	R	R	R			RD
TileTileID					C	C	U								D
PlayScore					C	C		U							D
ItemGame					C	R	R	RU							U
ItemItemID					C	R									
GameGameID					C	R									
TileTileID					R	R	R	RU							
PlayPlayID					R	R	R	U							U

1.4.4 CRUD Analysis

Effective development of persistent storage applications requires four functions, these are create, retrieve, update and delete (CRUD). CRUD is used to identify the events inherent to relational databases and the applications that use them, such as MySQL, Microsoft SQL Server, MariaDB and Oracle. The concept of CRUD functions works extremely well with a relational database, operating over attribute rows or fields within the table, or tuple.

The create function is used to create records in the database using INSERT, this adds a new row to a table and populates the columns within that row. The retrieve function can be viewed as a search function to retrieve specified information from the database, using SELECT, and can follow a set of customised criteria defined by the database user using the FROM and WHERE clause. The update function is used to modify specified data stored in the database using ALTER, for example, user information can change over time such as addresses or mobile telephone numbers. The delete function removes records from the database using DELETE. The decision to delete information should not be made lightly and some database administrators perform soft deletes that update the row status instead, depending on the circumstances this may be preferable to a hard delete with removes the record from the database entirely.

The CRUD table has been developed as a product of the logical ERD, the entities and attributes have been lifted from the current logical diagram version. The business and functional requirement in conjunction with the storyboards have been used to create the events listed on the table.

Login Check Credentials

This event occurs when a player attempts to log on to the system. The query needs to retrieve relevant fields from the player table to determine if the username matches an entry in the database, if found this is compared against the password to ensure it is correct.

If the query returns true, the user will be logged in. If there is no matching username record found the user would be directed to register as a new user, they can however attempt to log in again. If the password retrieved does not match against the username the system will return a message that notifies the user of the failed login attempt, the query will retrieve the number of failed login attempts to determine if the account should be locked.

The development of the prototype game and the project requirements does not include security considerations relating to the storing of passwords. A select query will be created that retrieves the record where the username and password match.

New User Registration

This event begins by checking the email and username input before creating a new record, if a conflict is found the user will be asked to input new records. If no conflict records are found an insert statement will input the new user's information.

Failed Login Account Lock

When an account is locked the player ID will be retrieved from the system and the player account lock field updated to true.

Login Successful

Where a username is found and the password retrieved match each other against the user input, the user will be logged in to the game home page. The information on the game home page is retrieved from several areas of the database:

- Retrieve a list of player usernames that are active for gameplay from the player table
- Retrieve the account admin status for the player account from the player table
- Update the account active, and failed login information in the player table
- Retrieve the high scores of all active players from the player table
- Retrieve the games currently in play from the game table
- Update the character turn for all active games the player is partaking in from the game table
- Retrieve play ID and game ID from the play table to display the number of players for each game

New Game

When a player chooses to start a new game the system must create many new fields across multiple tables and retrieve information from numerous fields. The game tables will need to create a new game and retrieve the board type that relates to that game, in the current planned prototype setup there is just one board type that will be retrieved. The character turn field is created listing the game creator as having the current turn.

The board type will need to retrieve the tiles that are assigned to that board and their location on the board from the tile table, this relationship is created in the Board/Tile join table. The final information that is retrieved is the characters, gems, and items that set the gameplay.

Lastly, the system will need to create the new play instance for the player that has created the game and update the relationship join table that assigned items to the game instance created.

Join Game

A player can choose to join an existing game where the player count is less than the number of maximum players. The system must retrieve the game information that has been chosen and update the character turn field to include the new player.

To display the game board and relevant components to carry out gameplay the system must retrieve the board type, tiles, board/tile, characters, gems and items. The system will create a new play instance that is assigned to the new player and retrieve the items that are associated with the game instance from the item/game join table.

Player Moves

When a player moves around the board the character turn field in the game table will need to be updated. The system will need to check which is the next character in the game list and then check that the player is active, if the player is not active the system will need to check the next character and so on. If no other players are active the turn will not revert to the player that has just completed a turn, instead, the field will revert to NULL. This field will not be updated again until one of the other players joins the game again after logging in.

The procedure will retrieve information from the tile table, the board and board/tile tables, the character table, and the gem and items associated with the tile. Finally, the procedure should update the tile ID for the play instance associated with the move.

Player Finds Gem

After a player has moved to a new tile the system will identify if the tile has items located on it, it will need to retrieve the items information from the items table and match any that are assigned to the same tile ID on the particular game instance. If no item is found the player is notified that the tile is empty of items.

If an item matches the tile ID the player is notified that the character has found a gem, the player can then choose the gem. There may be one or multiple gems and the system can update the tile ID and play ID field in the item/game table. A check will be made to ensure the character on the game instance matches the right play instance, then the play score can be updated.

The procedure compares the player ID in the play table to the player ID in the player table to calculate if the player high score needs to be updated.

End Game

A game ends when there are no more gem items left on the board. The player that has the highest points is declared the winner. The character turn table is updated to null, this condition is set at NULL permanently as a result of the game-ending.

Player Logout

If a player chooses to logout of the system their active status is updated to false, the system then retrieves all the games they are associated with and updates the character turn field so the system knows not to include them in the turn count. The procedure will determine the games for updating based on the information stored in the associated play instance for that player.

Enter Admin Screen

If a player clicks the admin button the admin screen will load only where certain conditions are met. The system checks that the player table has admin access permissions, if true the admin screen will load.

To load the relevant information the admin screen will retrieve player usernames and high scores as well as the games currently in play i.e. not finished, and the number of players in each game, which can be calculated from the retrieval of player ID and game ID matches from the play instance table.

Admin Kill Game

If an administrator kills a game in progress it will run the same sequence as the end game procedure, the only difference will be the notification sent to the active players, with no players being declared winners. Any high score already updated will also remain attached to player accounts.

Admin Add Player

If an administrator adds a new player the same information is created as with the new registration procedure, the system makes the same check to ensure the uniqueness of the entry.

Admin Update Player

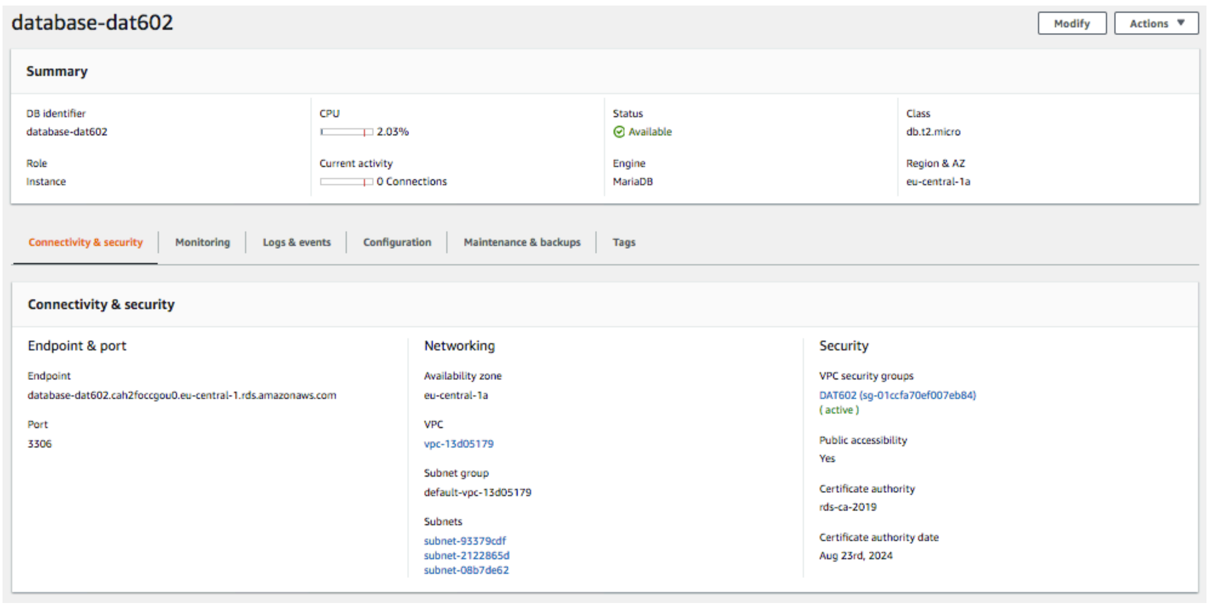
If an administrator updates a players information, the system retrieves the player ID and all other information stored in the player table. The administrator can update any or all the information stored in this table, except for the unique player ID field.

Admin Remove Player

If an administrator removes an existing player it deletes the entire player table record. It retrieves the player ID to determine which games the player is playing from the play instance table and deletes the records in the play tables. The games the player is associated with updates the character turn, any items the player has collected are updated so the play ID is set to NULL.

1.5 Structured Query Language (SQL)

The SQL file for the game is attached in a separate document and available through the authors GitHub account, access has been granted to the tutor and commits can be viewed. The database has been developed as a MariaDB hosted with AWS, which is backed up daily to ensure there is no loss of data. The data centre is located in Frankfurt, Germany to take advantage of the EU GDPR to offer the greatest level of privacy and security protection for users and their data. Details of the AWS MariaDB is displayed as follows:



database-dat602

Summary

DB identifier	CPU	Status	Class
database-dat602	2.03%	✓ Available	db.t2.micro
Role	Current activity	Engine	Region & AZ
Instance	0 Connections	MariaDB	eu-central-1a

Connectivity & security

Endpoint & port

Endpoint
database-dat602.cah2foccgou0.eu-central-1.rds.amazonaws.com

Port
3306

Networking

Availability zone
eu-central-1a

VPC
vpc-13d05179

Subnet group
default-vpc-13d05179

Subnets
subnet-93379cdf
subnet-2122865d
subnet-08b7de62

Security

VPC security groups
DAT602 (sg-01ccfa70ef007eb84)
(active)

Public accessibility
Yes

Certificate authority
rds-ca-2019

Certificate authority date
Aug 23rd, 2024

1.5.4 Data Definition Language (DDL)

1.5.4.1 Player Table

All player registration information is stored in the player table, this includes their current status and any account privileges. The table also stores the player's high score. The table consists of the following fields:

Player ID

The player ID is established as an auto-increment integer and set as the primary key and unique identifier. As this is the first table established the integer ID starts with a zero, the first test data entry is set to 000001. All other data entry fields in the table including username and email can be changed, which made an auto integer ID a logical choice and is then associated with other tables as a foreign key.

Email

The email field is set to varchar(50) because emails can be that long. The field is set to NOT NULL as this will be required for registration and contacting the player should the account become locked. The field is set to unique entries.

Username

The username field is set to varchar(10) to keep display names short and easy to read at a glance. The field is set to NOT NULL as this will be required for players joining games and registering their high scores. The field is set to unique entries.

Password

The password field is set to varchar(16) which is the recommended length to create a secure password. As the account must have a password to enable secure login the field is set to NOT NULL.

Account Admin, Account Locked, Active Status

These three fields use Boolean as their data types. The stored information is either set as true, i.e. it is true the account has admin privileges, or false, the account does not benefit from admin privileges.

Failed Logins

The failed login field will store numbers ranging from 0 to 5, as such the tinyint field type has been used. The default value will be set to 0 and each failed login attempt will add one to the current failed attempt number. The record is reset back to 0 once the player logs in successfully or an administrator unlocks the account.

High Scores

The high score field takes the highest score achieved by a player across all play instances of the games they have joined. The field is set as int which should easily allow for the high score maximums that can be achieved from a game instance. The field is set with a default value of 0.

1.5.4.2 Game Table

The game table stores each instance of games that have been set up by players, it includes an identifier, the board type being played and the character turn. The fields are as follows:

Game ID

The player ID is established as an auto-increment integer and set as the primary key and unique identifier. To make the auto number distinct from the player table, the starting ID is 100001.

Board Type

The board type is a foreign key field taken from the board table primary key. The field is set as varchar(20) which is the name given to the board type. The field is NOT NULL as a game requires a board type.

Character Turn

The character turn is a foreign key field taken from the character table primary key. The field is set as varchar(10) which is the name given to the character assigned to the player. The field can be nullable in instances where one player has joined the game, or only one player in the game is active.

1.5.4.3 Board Table

The board table is established as a parent table, the table assigns games to boards and boards to tiles that make up the board. The table consists of the following fields:

Board Type

The board type field is set as varchar(20), the field is used to give a simple clear description of the board type. No integer ID is required as there will be very few board types available. The field acts as the primary key for the table.

X-Axis

The field assigns the number of tiles in the grid along the X-axis. As this number is never likely to exceed 255 a tinyint field type is sufficient.

Y-Axis

The field assigns the number of tiles in the grid along the Y-axis. As this number is never likely to exceed 255 a tinyint field type is sufficient.

1.5.4.4 Tile Table

The tile table lists the separate tiles that make up the X and Y-axis of the board type. The tiles have a unique identifier that can be used to layout the board based on tile locations. The row and column numbers could have been used as a combine primary key, however, more flexibility is offered to stakeholders if they choose to create a more uniquely shaped board later in the business model life cycle. The fields are as follows:

Tile ID

The tile ID is established as an auto-increment integer and set as the primary key and unique identifier. To make the auto number distinct the starting ID is 001, the integer length is smaller than other tables as the tile count is unlikely to exceed 999.

Tile Row

The tile row is identified by a letter, to build in flexibility this is set as varchar(2) to allow the row sizes to go as high as AA, AB, AC and so on up to ZA, ZB, ZC and so on, to accommodate increasing board sizes. The field is set to NOT NULL as the tile instance requires a row location.

Tile Column

The tile column is identified by a tinyint which should be sufficient to accommodate increasing board sizes which are unlikely to go higher than 255 columns. The field is set to NOT NULL as the tile instance requires a column location.

Home Tile

The home tile field uses Boolean as the data type. The stored information is either set as true, i.e. it is true that this tile is the home tile, or false, this is not the home tile. The home tile has unique characters to comply with gameplay requirement so it must be identified.

1.5.4.5 Board/Tile Table

The board/tile table is used to link a tile record to the board type record in the relational database. This join table allows for different sized boards to created and potentially different shapes of boards.

1.5.4.6 Character Table

The character table establishes the seven dwarfs as characters, they are then assigned to players when they create or join a game. The fields are as follows:

Name

The character name field is set as varchar(10), the field is used for one of the seven dwarfs names. No integer ID is required as there will only be seven characters available. The field acts as the primary key for the table.

Tile Colour

The tile colour field assigns a colour to each character that identifies their position on the game board.

1.5.4.7 Gem Table

The gem table establishes the gem types that available for the player to collect when they land on a tile. The fields are as follows:

Type

The gem field is set as varchar(10), the field is used for one of 10 gem names. No integer ID is required as there will only be ten gems available, this could increase but it should never increase significantly to require an ID field. The field acts as the primary key for the table.

Points

The points field assigns a number to the gem, the points are the amount given to a player score as they collect gems. This field is a tinyint as the points score will never go higher than 255.

1.5.4.8 Item Table

The item table acts to assign gems to an item record, as there can be multiple instances of the same gem type on each board the items define instances of the gems. The fields are as follows:

Item ID

The item ID is established as an auto-increment integer and set as the primary key and unique identifier. To make the auto number distinct from tile ID the starting ID is 101, the integer length is smaller than other tables as the item count is unlikely to exceed 999.

Gem Type

This field is a foreign key entry of the gem table primary key. This assigns the item instance with the gem type name and the points related to it. The field is varchar(10) and set to NOT NULL as each item must be a gem type.

1.5.4.9 Play Table

The play table acts to assign players to specific games, playing a specific character, which is currently located on one tile. The fields are as follows:

PlayID

The play ID is established as an auto-increment integer and set as the primary key and unique identifier. To make the auto number distinct from other tables the starting ID is 500001, the integer length is the same as players because each player will play at least one game establishing a play instance.

Player ID

The player ID field is a foreign key of the player ID primary key established in the player table. This field is required to identify the player that has established this play instance record. The field is set to NOT NULL as a player creates the play instance when they join a game.

Character Name

The character name field is a foreign key of the character name primary key established in the character table. This field is required to identify the character that the player has been assigned. The field is set to NOT NULL as a player is assigned the next character when they join a game.

Game ID

The game ID field is a foreign key of the game ID primary key established in the game table. This field is required to link the player play instance to the game the player joined. The field is set to NOT NULL as a player is joining a game creating the play record.

Tile ID

The tile ID field is a foreign key of the tile ID primary key established in the tile table. This field is required to record where on the board of that game the particular player is playing. The field is set to NOT NULL as a player must be located on a tile.

Play Score

The field is set as int which should easily allow for the score count maximums that can be achieved from a game instance. The field is set with a default value of 0.

1.5.4.10 Item/Game Table

The item/game table is used to link an item record to the game record in the relational database. This join table allows for items to be associated with games so they can be found by players. Two additional fields are required by this table as follows:

Tile ID

The tile ID field is a foreign key of the tile ID primary key established in the tile table. This field is required to identify the location of the item. The field is nullable as the tile ID will be removed when the item is found by a player.

Play ID

The play ID field is a foreign key of the play ID primary key established in the play table. This field is required to identify that an item has been found by a character. The field is nullable as the play ID will only be inserted when the item is found by a player.

1.5.5 Test Data & Queries

Player Table

The table has been established with 8 entries, each player has been assigned different privileges and some have login failed attempts against their account. All are considered as being not active so all players must log in once the system is set up.

Administrator access has been assigned to several accounts as admins which will be required to carry out functions that are only available to them, such as unlocking accounts or making other players administrators. This combination should provide for adequate testing of the system.

Queries are to be established as defined in the CRUD table and the test data allows for these queries to be adequately tested. For example, not all players are assigned to games, these players can join a game with an associating procedure which will be created in milestone 2. The procedures created for milestone 1 include an update statement for a player username, select account status and delete a player.

Game Table

As part of the test data two games have been established in the database, the board has been defined and the character turn, 'Doc', is listed as the game is set up, but no activity has yet occurred.

The queries for milestone 2 will be to update the character turn once the current character has placed there move on the board. The query then looks for the next character player, to determine if they are active, if not it moves to the next player and so on. A delete statement will also be created that allows an admin to kill the game by deleting the game record and associated tables and fields.

Board Table

For testing purposes the board table includes just one instance, this can be increased as the prototype development proceeds. The board is 9 tiles by 9 tiles and is used as the base for all games in play. A query can be set up to add a new board type or edit an existing board type, but this is not necessary at this stage of the development. The procedures created for milestone 1 include an update statement for a new board size, select board axis data and delete a board.

Tile Table

The tile table includes test data for all tiles required to accommodate a 9 tile by 9 tile game board. The tiles row and column indicator are used to place them out over a square board grid, only one tile is assigned as being the home tile indicated by true. A query can be set up to add a new tile or edit the location of a tile, but this is not necessary at this stage of development. The procedures created for milestone 1 include an update statement for a tile location on the board, select the tile data for a specific tile and delete a tile.

Board/Tile Table

The Board/Tile table will list a data input for each tile instance and places it on the current board type that has been established. A separate query for the game would generate a new board type and possibly a new tile, the board/tile table would be updated to include the board and tiles associated with each other as part of that board. The procedures created for milestone 1 include an update statement for a tile associated with a board, select the board a tile is associated and delete a board/tile relationship.

Character Table

The character table lists all of the seven dwarf characters and an associate tile colour for the board. No specific character table queries are established as this table is unlikely to be updated. The procedures created for milestone 1 include an update statement for the character colour, select character name and delete a character.

Gem Table

The gem table list all of the gems that can be found by the players and associated points. There are currently ten gem types that are enough for testing purposes. A query can be set up to add a new gem or edit the gem type points, but this is not necessary at this stage of the development. The procedures created for milestone 1 include an update statement for the gem points, select the number of points associated with a gem type and delete a gem type.

Item Table

The item table test data lists all the instances of a gem type on a board, that can be found in each game instance. There are 10 gem types and 7 instances of each gem in each game making 70 records in the test data. No specific item table queries are established as this table is unlikely to be updated. The procedures created for milestone 1 include an update statement for an item to be associated with a different gem type, select the gem type for a tile instance and delete an item.

Play Table

The play table includes data that ensures the gameplay can function as intended. The test data includes play ID's which are linked to five of the players established in the player table, each player has a character in order of their established account creation order, the game they created or joined, their tile location, and game points which is 0, as no items have yet been collected. The procedures created for milestone 1 include an update statement for a character turn, select the character that has the next turn and delete a play instance.

Item/Game Table

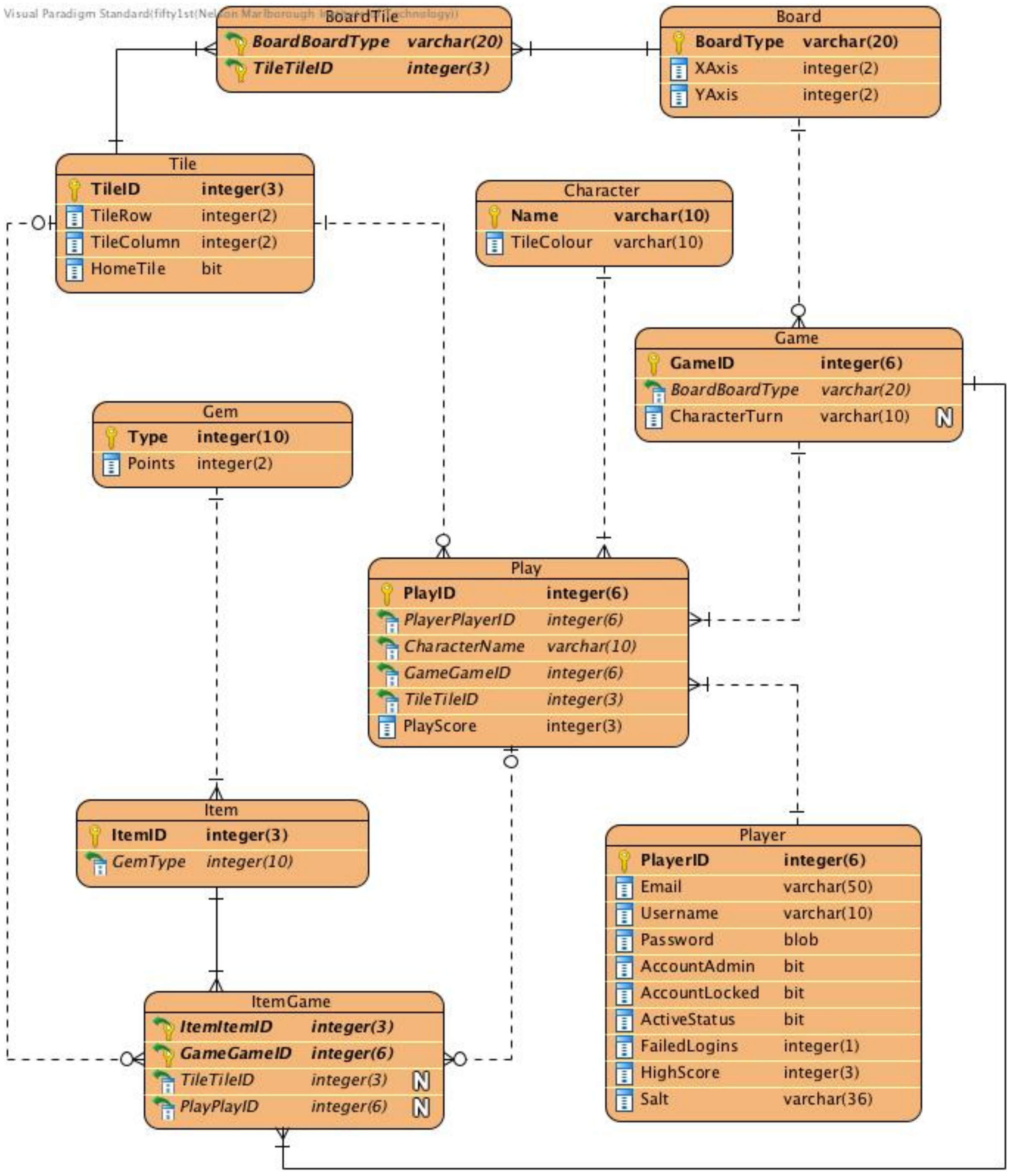
The item game table required the most test data input to accommodate the records established in the other tables. There are two games and each game needs all 70 items to allow for the gameplay, making 140 records in total. Each gem has a tile location meaning the play ID can be set as NULL. This table is updated as part of the query that establishes a new game instance.

A query is created that sets the tile location to NULL when a character finds the item and consequently updates the play ID field with the play ID of the players whose character found the gem. A query can be set up to add a new item or edit an item tile location or play ID, but this is not necessary at this stage of the development. The procedures created for milestone 1 include an update statement for a tile associated with an item associated with a game, select the location of an item on a game instance and delete the item/game records.

2.0 Milestone Two

2.1 Design Alterations

2.1.1 Logical Diagram



2.1.2 Rationale Logical & Physical Design

Alterations to the logical diagram have been minor during the transition from milestone 1 to milestone 2. The following alterations have been made to reflect changes implemented in the physical database design:

1. The tile row column in the tile table has been set to an integer to accommodate the player moves procedure, previously this had been string varchar. The change required the test data from milestone 1 to be updated replacing the string values with numbers.
2. Security has been considered for the database; this required some fundamental changes to ensure the player data at rest is secure. The first change was investigating password encryption, after researching the options the decision was made to use AES encryption as it offered the highest protection against most attack vectors. As such the player table column "password" was updated from varchar(16) to BLOB.
3. AES encryption only goes so far and can be subjected to some attacks such as brute force or rainbow attacks. To help mitigate the effects of such attacks a column was added for salt, to accommodate a unique identifier generated for each registration. This UUID is applied to the password passed by the player, this makes rainbow attacks far more difficult to achieve.

2.2.1 CRUD Table

Entity/Attribute	2.3.1 User Registration	2.3.2 Login Check Credentials	2.3.4 New Game	2.3.5 Join Game	2.3.6 Player Moves	2.3.7 Find Gem	2.3.8 Select Gem & Update Turn	2.3.9 Update High Score & End Game	2.3.10 Player Logout	2.3.11 Enter Admin Screen	2.3.12 Admin Kill Game	2.3.13 Admin Add Player	2.3.14 Admin Update Player	2.3.15 Admin Delete Player
Player	CR	RU	R	R	R			R	RU	R	R	CR	CRU	RD
PlayerID	C	R		R								C	R	D
Email	CR	R										CR	RU	D
Username	CR	R	R		R				R	R	R	CR	RU	RD
Password	C	R										C	RU	D
AccountAdmin	C	R								R	R	CR	RU	RD
AccountLocked	C	RU										C	RU	D
ActiveStatus	C	RU							U			C	RU	D
FailedLogins	C	RU										C	RU	D
HighScore	C	R						R		R		C	RU	D
Salt	C	R										C	CRU	D
Game		R	C	R	R		U	U		R	RD			
GameID		R	C	R	R			R		R	RD			
BoardBoardID			C								D			
CharacterTurn			C		R		U	U			D			
Board			R											
BoardType			R											
XAxis														
YAxis														
Tile			R		R									
TileID			R		R									
TileRow			R		R									
TileColumn			R		R									
HomeTile			R		R									
BoardTile			R											
BoardBoardID			R											
TileTileID			R											
Character			R	R	R									
CharacterName			R	R	R									
TileColour			R		R									
Gem			R			R	R							
Type			R			R								
Points			R			R	R							
Item			R			R								
ItemID			R			R								
GemType			R											
Play		R	C	CR	RU	R	R	R		R	RD			D
PlayID			C	C		R	R	R			D			D
PlayerPlayerID			C	CR	R	R					D			D
CharacterName			C	C	R		R	R			D			D
GameGameID		R	C	CR	R	R	R	R		R	RD			D
TileTileID			C	C	RU	R	R				D			D
PlayScore			C	C			RU	R			D			D
ItemGame			C			R	R	R			RD			D
ItemItemID			C			R	R				D			D
GameGameID			C					R			RD			D
TileTileID			R				U	R			D			D
PlayPlayID							U				D			D

2.2.2 CRUD Analysis

The CRUD table has been updated to reflect the changes made to the logical diagram, these have been represented in the CRUD table through the insertion of the salt column in the player table.

The main change has been to the headings and the location in which certain events occur, some of these changes are a result of a lack of understanding of the game process at the time of the milestone 1 report, other changes are a result of constraints within the physical database build.

The CRUD table headings have been changed to reflect the procedure names in the physical build, changes are represented as follows:

1. The login and check credentials can be performed in one procedure and this is reflected in the amended CRUD table. This procedure, login success, is also the home screen display procedure, made up of two select statements. Any updates now occur in the login check credentials procedure, which after a review is a more logical process.
2. As mentioned in point 1 above, procedure 2.3.3 of the CRUD has been removed as detailed in section 2.3.3 of this report, its data is retrieved in procedure 2.3.2 of the CRUD table.
3. The join game procedure has been amended to reflect the reduction in the quantity of information that must be retrieved by the procedure, meaning it performs only the necessary functionality.
4. The find gem procedure is now a select statement procedure that stores the necessary information in a temporary table, this must be performed first to allow the player to select the gem they want, should multiple gems be located on a tile.
5. A new procedure has been created that performs the update functions including the selecting of a gem and the updating of the character turn, this update only happens after the turn is completed, which is triggered by the selection of a gem. The play points are also updated in this procedure, acting as part of the turn ending.
6. The end game procedure has been amended to now include an update of the player's high score, this check is run at the end of each turn, however, the end game only occurs if there are no items left to collect. If the check confirms no items are left, then the game turn in the game table is updated to NULL, signifying the end of the game.
7. The update game turn functions, in most of the other procedures, have been removed, this was a game rules decision. It made little sense for players to continue playing and receiving turn whilst another player had left the game, this would give an unfair advantage and mean games could be won simply by staying in play for the longest amount of time.
8. Salt has been added as an input to all procedures that require it.

2.3 SQL Procedures & Transactions

The SQL file for the game is attached in a separate document and available through the authors GitHub account, access has been granted to the tutor and commits can be viewed.

2.3.1 User Registration

The database contains a constraint that only allows unique values to be allocated to Email and Username, should a new user attempt to register with either value existing in the database the procedure will not run. Otherwise, the procedure requires a user to enter an email, username and password with the remaining fields being created from the defaults, the procedure gives new players the ability to register an account.

2.3.2 Login Check Credentials

This procedure allows a user to log in to the game, it retrieves the users salt record to ensure the password is passed correctly in addition to the user’s active status to ensure they are not already logged in. If the user is logged in their details are displayed and a message is passed stating their active status. If an incorrect username or password is entered and an error message is returned.

Two select statements make up the procedure and have been designed with thought given to the end GUI. Should a login attempt be successful, which is further check by selecting the active status of the user, then the relevant information as described in the storyboarding is displayed.

2.3.3 Home Screen Display

This did include the two select statements listed in the login check credentials procedure, this has been removed as it makes more sense to be included in that procedure.

2.3.4 New Game

The new game procedure creates a new game in the game, which includes an autoincrement ID, a new play instance for the player that creates the new game, and a new item list associated with the game in the item/game table. The play table and item/game table retrieves the newly created game ID, this is achieved by declaring the new game ID and passing the LAST_INSERT_ID() function. Finally, the items are allocated to tiles within the game, this is done randomly so each game is different.

2.3.5 Join Game

When a player joins a new game, the next available character is selected, a play instance is created that is assigned to the game with the character and the player ID. If all seven dwarf characters are playing in the game, then an error message is displayed.

2.3.6 Player Moves

This procedure moves a player to a new tile if the tile is plus or minus one from the player's current tile position in a game instance, the play table records the current player's position. If a player is already on the tile, with an exception made for the home tile, then the player cannot move to it and an error message is displayed. Likewise, if a player selects a tile that is not plus or minus one adjacent from the current tile an error message is displayed. If a player is on a tile but has logged out of their account then the game allows another player to move to this tile, when the first player logs back in their turn instruct them to move to a new tile as normal.

2.3.7 Find Gem

When a player lands on a tile this procedure will run, it displays all the items on that tile so that one gem can be selected by the player.

2.3.8 Select Gem & Update Turn

The player can select one of the items from the temporary table relating to the game instance, this selection will update the play ID of that game to the players play instance and removes the item from the tile by updating the column to NULL. The next turn is updated in the game table instance and the points are added to the players play instance total.

2.3.9 Update High Score & End Game

This procedure leads on from the preceding procedure and checks if the added points to the play instance total is now higher than the player's high score, if it is higher, the player's high score is updated. The procedure then looks to determine if that item selection ends the game by identifying if there are any more items left to collect, if true then it selects the player with the highest score as the winner and updates character turn in the game table to NULL, effectively ending the game as no more turns can occur.

2.3.10 Player Logout

The logout procedure updates the players active status to false, meaning that if they want to access the game again the login procedure will check the active status and request login credentials.

2.3.11 Enter Admin Screen

When admin access is requested the procedure checks if the player has admin privileges, if successful the admin screen displays the relevant information that an admin would require. If the player is not an admin an error message is returned.

2.3.12 Admin Kill Game

The procedure carries out an additional check to ensure the user has admin privilege, then deletes a game and all the play instances and item/game instances associated with that game.

2.3.13 Admin Add Player

An admin can use this procedure to add a player, many of the inputs are set to default and as such, there is no need to alter these manually for this procedure, as is the case with the new registration procedure. The only feature included is for the manual input of admin status, it may be useful for admin to allocate admins at this stage, further changes can be made in the update player procedure.

2.3.14 Admin Update Player

The procedure allows an admin user to update all information pertaining to an existing player, except for the player ID.

2.3.15 Admin Delete Player

The procedure allows an admin user to delete all information about an existing player. The procedure deletes the player record, the play instances associated with the player and any association with item records in the item/game table.

2.4 ACID

ACID is an important database design model that describes four separate aims that database management systems should incorporate. These aim to guarantee the reliability of transactions processed in the database. To this end, the ACID model concerns itself with how a database can recover from a failure that occurs whilst processing a given transaction, to maintain accuracy and consistency despite the failure.

For example, in a bank database, financial transactions will occur to move a given amount of money from one account to another, in this situation the transaction should either complete accurately, the correct monetary value is removed from the senders account and the same monetary value is received by the receivers account, with a separate log being recorded of the transaction. Should the transaction fail at a set point, such as before the money is added to the receivers account but after it has been removed from the sender's account, then the entire procedure should fail, so the data is restored to its correct state. This means it remains accurate and consistent despite the failure.

ACID provides a mechanism that seeks to ensure the correctness of a database, ensuring that transactions group operations into a single procedural unit to produce results that are consistent and act in isolation from other updates or operations. The principles of ACID are adhered to by all major relational database management systems including InnoDB as part of MySQL, with features that support these four aims. The four aims of ACID that form the acronym, stand for atomicity, consistency, isolation and durability, and these are explained as follows to support the importance of the model in situations like that described above.

2.4.1 Atomicity

Atomicity is a rule that states any modifications to a database act on an ‘all or nothing’ principle, meaning that transactions are atomic, so if one part of the transaction fails then the entire transaction fails. The nature of the atomic principle is critical to ensure that transactions guarantee the data stored, in the event of a database, systems of hardware failure.

In the game development, atomicity is crucial when dealing with the integrity of items assigned to game instances and located on tiles. When a player selects an item from a tile the item must be removed from a tile in the item/game table, setting the tile record to NULL and assigning the item to the player.

This transaction means that a game can end when all items have been collected from tiles, meaning points are correctly allocated to players, which then updates their play score in the play table. If the tile in the item/game table was not being set to NULL when the item was selected by a player, then the game would never end as the procedure would register items still on tiles, despite them being allocated to players play instance. Likewise, if the item was not updated with the play instance of the player collecting the item, then the points would not be allocated to the play score when it increases.

At the same time, this occurs the player turn must update to ensure that the next player can make their move, these transaction form part of the same procedure to ensure atomicity. To ensure these changes, InnoDB is set to 'autocommit', meaning that changes to a table will take effect immediately.

2.4.2 Consistency

Consistency ensures that all data is guaranteed according to a set of defined rules established in the database, these rules are constraints, cascades and triggers. The rules enforce database consistency by rolling back transactions that violate these rules, meaning that the database is restored to a state consistent with those rules. If the rules are met then the database is taken from one state that complies with these rules, to another state that also complies with these rules.

There are several consistency rules applied to the game database that help to guarantee the data stored, foreign keys across tables is a prime example of a consistency constraint. Additional constraints include data types and length that can help to enforce consistency, in addition to unique values such as username and password in the player table and NOT NULL values, which are used across the database tables.

2.4.3 Isolation

Isolation is a guarantee that transactions are not affected by other transactions that are yet to complete, in effect, they occur independently of each other. This allows for multiple transactions to occur concurrently without interference.

In the game database, this allows for player A to create a new game whilst player B is waiting to join a game, player B will not be able to join the new game created by player A until the transaction is written to memory by being called. This means the join game transaction cannot read data from the new game until the new game transaction is completed.

Additionally, isolation means that player A can join a game at the same time as player B, one transaction will be performed in its entirety before the next transaction is performed, this prevents one transaction from reading intermediate data produced as a side effect of the other transaction. Isolation does not state which transaction is performed first, just that they are performed in isolation of each other.

The local version of the game database uses InnoDB which, by default, uses repeatable read as the isolation level. This ensures that transactions are read consistently as a snapshot. The isolation of the procedures and transactions means that players should be able to operate across games, with each game instance and play instance acting on the procedures in isolation of one another. This is further helped by the game rules being that players on a game instance play in turn.

Transaction isolation is vital for database processing, without it the game development for this course could not be achieved. Isolation forms part of the ACID principle and allows multiple transactions to be performed, and to make changes to the database, at the same time. This principle and functionality mean a database is consistent and reliable when these procedures are processed. There are several isolation levels that a database analyst can choose from to meet requirements:

2.4.3.1 Repeatable Read

This isolation level performs consistent reads within a transaction read, meaning select statements are consistent with each other. This is the default isolation level for InnoDB, which is being used for the local version of the game database development. In this isolation level changes made by other transactions are not taken into account.

2.4.3.2 Read Committed

This isolation level produces a separate snapshot for each consistent read, even when using the same transaction. This means that uncommitted changes are not visible to any other transaction that is being performed. The default isolation level in InnoDB of repeatable read has been changed to the read committed isolation level. This decision has been made due to the balance trade-off for this type of multi-player role-based game.

2.4.3.3 Read Uncommitted

In this isolation level transactions can view data changes made by other transactions before they are committed. This is the lowest level of isolation available and can lead to the reading of data that may never get committed, leading to what is known as a 'dirty-read'.

2.4.3.4 Serializable

This isolation level will isolate transactions from one another, meaning one cannot impact another. One transaction must finish before another can be completed, these combined make this the strongest possible isolation level.

2.4.3.5 Consistent Nonlocking Reads

A consistent read presents a query snapshot of the database at a specific point in time, meaning that any transaction that has committed changes to the database, before the specific point in time, is returned. Changes contained within uncommitted transactions are not returned in the query.

Furthermore, unlike the default InnoDB setting 'repeatable read' where consistent reads contained in the same transaction, read the snapshot that is established by the first read, 'read committed' sets and reads a fresh snapshot for each consistent read in the transaction.

2.4.3.6 Locking Reads

Locking reads include:

- SELECT with FOR UPDATE or LOCK IN SHARE MODE
- UPDATE statements
- DELETE statements

For these instances, only index records are locked, this allows for new records to be inserted next to locked records. With UPDATE and DELETE statements, 'read committed' holds locks for the rows that are being updated or deleted, rows that are not included in the statement are released after the WHERE condition has been evaluated. This is important for reducing the chance of deadlocks occurring, but it doesn't entirely eliminate them.

If a row is locked and forms part of an UPDATE statement, a 'semi-consistent' read is performed in InnoDB. In these cases, the latest committed version is returned and the row is checked to determine if it matched the WHERE condition. If the match is confirmed then the row is read again and locked, or MySQL waits for it to be locked.

2.4.3.7 Transaction Activity

Activity performed on the database by its users occurs inside the transactions. This activity takes the form of:

- Autocommit
- Commit
- Rollback

Where a database has 'autocommit' enabled, all statements will form separate, single transactions. ‘autocommit’ is enabled by default when each session is started, which means that each statement is committed unless an error is found, at which point the statement may be rolled back.

With ‘autocommit’ enabled on a session, multiple-statement transactions can be performed. When COMMIT or ROLLBACK is used to end a transaction, a new transaction can then be started. If ‘autocommit’ is disabled on a session and the transaction does not end with COMMIT, ROLLBACK will be performed on the transaction.

COMMIT makes the changes made in a transaction permanent and available to other sessions, ROLLBACK cancels any modifications made in a transaction

All of the locks that are set in a transaction are released when COMMIT or ROLLBACK are run at the end of the transaction.

2.4.4 Durability

After a transaction has been called and committed to the database, durability states that the data should not be lost, even in the event of a system crash. The changed state of the data resulting from the transaction should remain in the system permanently, if a user has received a message that a transaction was a success, then it must have been successful and must remain that way.

Durability can be maintained through the use of regular database backups and logs of the transactions made. Performing these operations allows for the restoration of the database should a significant data loss or breach occur, either through software or hardware failure, or a malicious attack targeted at the database.

When stored in the AWS environment the game database was backed up daily to ensure that any data loss was minimal, mitigating the effects. The game needs to ensure the durability of many game instances, locations of players on tile, current play scores and login credentials, to prevent disruption of the service.

2.5 Test Data

The test data produced in milestone one is sufficient to successfully complete the procedures that makeup milestone two. Any additional functionality such as password encryption and salt can be carried out through calling the procedures that relate to these instances.

3.0 Milestone Three

3.1 CRUD Table

Entity/Attribute	2.3.1 User Registration	2.3.2 Login Check Credentials	2.3.3 Home Screen Display	2.3.4 New Game	2.3.5 Join Game	2.3.6 Player Moves	2.3.7 Find Gem	2.3.7.1 Gem Display	2.3.8 Select Gem & Update Turn	2.3.9 Update High Score & End Game	2.3.10 Player Logout	2.3.11 Enter Admin Screen	2.3.12 Admin Kill Game	2.3.13 Admin Add Player	2.3.14 Admin Update Player	2.3.15 Admin Delete Player
Player	CR	RU	R	R	R	R				R	RU	R	R	CR	CRU	RD
PlayerID	C													C	R	D
Email	CR													CR	RU	D
Username	CR	R	R	R	R	R					R	R	R	CR	RU	RD
Password	C	R												C	RU	D
AccountAdmin	C											R		C	RU	D
AccountLocked	C	U												C	RU	D
ActiveStatus	C	U									U			C	RU	D
FailedLogins	C	U												C	RU	D
HighScore	C		R							R		R		C	RU	D
Salt	C	R												C	CRU	D
Game			R	C	R	R			U	U		R	RD			
GameID			R	C	R	R				R		R	RD			
BoardBoardID				C									D			
CharacterTurn				C		R			U	U			D			
Board				R												
BoardType				R												
XAxis																
YAxis																
Tile				R		R										
TileID				R		R										
TileRow				R		R										
TileColumn				R		R										
HomeTile				R		R										
BoardTile				R												
BoardBoardID				R												
TileTileID				R												
Character				R	R	R										
CharacterName				R	R	R										
TileColour				R		R										
Gem				R				R	R							
Type				R				R								
Points				R				R	R							
Item				R				R								
ItemID				R				R								
GemType				R												
Play			R	C	CR	RU		R	R	R		R	RD			D
PlayID				C	C			R	R	R			D			D
PlayerPlayerID				C	CR	R		R					D			D
CharacterName				C	C	R			R	R			D			D
GameGameID			R	C	CR	R		R	R	R		R	RD			D
TileTileID				C	C	RU		R	R				D			D
PlayScore				C	C				RU	R			D			D
ItemGame				C			R	R	R	R			RD			D
ItemItemID				C			R	R	R				D			D
GameGameID				C			R			R			RD			D
TileTileID				R			R		U	R			D			D
PlayPlayID									U				D			D

3.2 CRUD Analysis

The most significant change to note is the reversion of the CRUD table to reintroduce the home display procedure implemented in the early iterations of milestone two. This was removed at a late stage in milestone two so that the list generated from the procedure would display automatically in the list view when the user logged in, this meant the retrieval of many of the fields in the procedure were no longer required. Likewise, due to the nature of the GUI implementation, the user active status is no longer required as part of the procedure.

Another significant change is the separation of the 2.3.7 Find Gem procedure to add the 2.3.7.1 Gem Display procedure that has been created in SQL. As a result, the find gem procedure can deal with the logic that determines if a gem is on the tile selected and if not, a separate display message can be generated in the GUI. If there are gems on the tile, the gem display procedure can then pass the select statement into the data grid view, a player can then select one of the listed gems and the points are added to their game score.

The implementation of the GUI to communicate with the database has led to several additional changes that weren't anticipated in the earlier milestones. All of the procedures that are performed by the admin user, for instance, do not need to check that the user is an admin, this need only occur once when the admin user first enters the admin area, non-admins are stopped at this point and prevented from proceeding. This change is reflected in the CRUD table.

Lastly, additional changes include the information passed for the procedures. Originally the game called the player IDs, the thinking behind this was that these are a constant in the database and as such would be a more stable input. In reality, the game doesn't operate this way and it is the player's username that is the more logical input call through the procedures. These changes might have been better anticipated had I had more experience with the GUI element of the build before designing the database, however, the iterative nature of the milestones allows for development to occur alongside the learning process.

3.3 GUI Components

Development work for milestone three of the game development has required updates of both the SQL and data access class in C#, ensuring it can function with the GUI forms created for the storyboards in milestone one. The interaction between the GUI through the data access class to the SQL meant that further consideration had to be made to the exit handlers within the procedures, this was to ensure the game could operate effectively and not encounter errors, forcing it out of the GUI and back to the raw code in Visual Studio.

The effort spent to create the forms in Visual Studio for the storyboards in milestone one proved invaluable, it required extra time learning a package that I had no prior experience with but allowed consideration to how the game would operate. As a result, there have been limited changes to these forms other than the inclusion of the data grid views, an integration feature that I was unaware of in milestone one.

The same setup process was followed as with milestone two, Visual Studio required the installation of MySQL connector through the NuGet tool. This allowed the data access class developed in milestone two to connect with the database running in MSQL Workbench. The data access class required additional changes to reflect the updated CRUD table, however much of the heavy lifting had already been achieved in milestone two, particularly understanding and applying the differences in data types across the two coding languages and platforms.

The functionality incorporated into the GUI as part of the development includes:

- Register as a new player without admin privileges
- Login credential check to access the game area
- Create a new game that other players can then join
- Join an existing game where the player count is less than 7
- Move a player from one tile grid to an adjacent tile grid
- Access the admin area where a user is authorised
- Select a game and kill it, removing it from the database
- Add a new player and set all the various privileges
- Update a player, allows for all the player fields to be updated
- Delete a player from the database
- Logout of the game

Creating a working GUI that interacted with the database was a far more rewarding task when compared with the list formula deployed in milestone two. Witnessing a fully functioning GUI input, whether that was an on click button, or passing information in text fields, which could be viewed in the database, gave a great deal of achievement, particularly as this type of coding is new to me.

Changes made to the CRUD table as a result of a lack of GUI experience were also required with the GUI functionality itself, several of the navigation buttons were deemed to no longer be necessary. For example, if an admin user is seeking to update a player, the update player form does not need to have functionality for the user to move back to the home page or to log out, logically they would finish the process with part of that completion taking the player back to the main admin screen automatically. It makes more sense that such functionality is located on the primary admin screen.

The biggest challenge was passing data into the data grid view, once this was accomplished I started to understand what the code was performing, and why. A lack of confidence led to a belief that the implementation of selecting the current logged in user, to then pass through the procedure, would be an insurmountable task without support, however, the logic became clear and I achieved this independently. The biggest personal benefit has been my ability to now better understand the requirements of procedures in SQL when these will ultimately be used to pass through GUIs.

References

ACID Properties in DBMS. (2016, August 7). *GeeksforGeeks*. https://www.geeksforgeeks.org/acid-properties-in-dbms/

Bichot, G. (2015, October 14). Storing UUID Values in MySQL Tables. *MySQL Server Blog*. https://mysqlserverteam.com/storing-uuid-values-in-mysql-tables/

C# Windows Forms Application Tutorial with Example. (n.d.). Retrieved March 6, 2021, from https://www.guru99.com/c-sharp-windows-forms-application.html

CREATE. (n.d.). MariaDB KnowledgeBase. Retrieved March 17, 2021, from https://mariadb.com/kb/en/create/

Finding records in one table not present in another table. (2007, September 20). https://weblogs.sqlteam.com/peterl/2007/09/20/finding-records-in-one-table-not-present-in-another-table/

Frank. (2006, August 21). MySQL Consulting and NoSQL Consulting: MySQL DBA: Storing Passwords in MySQL. *MySQL Consulting and NoSQL Consulting*. http://mysqldatabaseadministration.blogspot.com/2006/08/storing-passwords-in-mysql.html

Gould, H. (2016). Systems Analysis and Design (1st Edition). bookboon.

How do you securely store a user's password and salt in MySQL? (n.d.). Stack Overflow. Retrieved April 24, 2021, from https://stackoverflow.com/questions/7270526/how-do-you-securely-store-a-users-password-and-salt-in-mysql

LAST_INSERT_ID. (n.d.). MariaDB KnowledgeBase. Retrieved April 27, 2021, from https://mariadb.com/kb/en/last_insert_id/

MySQL | AES_DECRYPT ( ) Function. (2019, November 6). *GeeksforGeeks*. https://www.geeksforgeeks.org/mysql-aes_decrypt-function/

MySQL Error Handling using the Signal and Resignal Statements—DatabaseJournal.com. (n.d.). Retrieved April 27, 2021, from https://www.databasejournal.com/features/mysql/mysql-error-handling-using-the-signal-and-resignal-statements.html

MySQL first()—Javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved April 27, 2021, from https://www.javatpoint.com/mysql-first

MySQL LAST_INSERT_ID Function By Practical Examples. (n.d.). *MySQL Tutorial*. Retrieved April 27, 2021, from https://www.mysqltutorial.org/mysql-last_insert_id.aspx

MySQL :: MySQL 5.7 Reference Manual: 14.7.2.1 Transaction Isolation Levels. (n.d.). Retrieved May 12, 2021, from https://dev.mysql.com/doc/refman/5.7/en/innodb-transaction-isolation-levels.html

MySQL :: MySQL 8.0 Reference Manual: 5.1.8 Server System Variables. (n.d.). Retrieved April 27, 2021, from https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html#sysvar_autocommit

MySQL :: MySQL 8.0 Reference Manual: 12.24 Miscellaneous Functions. (n.d.). Retrieved April 24, 2021, from https://dev.mysql.com/doc/refman/8.0/en/miscellaneous-functions.html#function_uuid

MySQL :: MySQL 8.0 Reference Manual: 13.2.13 UPDATE Statement. (n.d.). Retrieved March 17, 2021, from https://dev.mysql.com/doc/refman/8.0/en/update.html

MySQL :: MySQL 8.0 Reference Manual: 13.6.5.8 WHILE Statement. (n.d.). Retrieved April 27, 2021, from https://dev.mysql.com/doc/refman/8.0/en/while.html

MySQL :: MySQL 8.0 Reference Manual: 15.1 Introduction to InnoDB. (n.d.). Retrieved April 27, 2021, from https://dev.mysql.com/doc/refman/8.0/en/innodb-introduction.html

MySQL :: MySQL 8.0 Reference Manual: 15.2 InnoDB and the ACID Model. (n.d.). Retrieved April 27, 2021, from https://dev.mysql.com/doc/refman/8.0/en/mysql-acid.html

MySQL :: MySQL 8.0 Reference Manual: 15.7.2.1 Transaction Isolation Levels. (n.d.). Retrieved April 27, 2021, from https://dev.mysql.com/doc/refman/8.0/en/innodb-transaction-isolation-levels.html#isolevel_repeatable-read

MySQL :: MySQL 8.0 Reference Manual: 15.7.2.2 autocommit, Commit, and Rollback. (n.d.). Retrieved May 12, 2021, from https://dev.mysql.com/doc/refman/8.0/en/innodb-autocommit-commit-rollback.html

MySQL :: MySQL 8.0 Reference Manual: 15.7.2.3 Consistent Nonlocking Reads. (n.d.). Retrieved May 12, 2021, from https://dev.mysql.com/doc/refman/8.0/en/innodb-consistent-read.html

MySQL DELETE - Deleting Data from a Table. (n.d.). *MySQL Tutorial*. Retrieved March 17, 2021, from https://www.mysqltutorial.org/mysql-delete-statement.aspx

MySQL Stored Procedure Parameters. (n.d.). *MySQL Tutorial*. Retrieved March 17, 2021, from https://www.mysqltutorial.org/stored-procedures-parameters.aspx

MySQL Stored Procedure Tutorial. (n.d.). *MySQL Tutorial*. Retrieved March 17, 2021, from https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx/

Salted, M. G. U., Hashed, February 09, L. was published on, & 2015. (n.d.). *MySQL: Generate Unique Salted, Hashed, Logins*. Boone Putney. Retrieved April 24, 2021, from /development/mysql-generate-salted-hashed-logins/

Salting and hashing passwords in MySQL. (n.d.). Stack Overflow. Retrieved April 24, 2021, from https://stackoverflow.com/questions/23278565/salting-and-hashing-passwords-in-mysql

SQL query to find second highest salary? - GeeksforGeeks. (n.d.). Retrieved April 27, 2021, from https://www.geeksforgeeks.org/sql-query-to-find-second-largest-salary/

SQL Tutorial. (n.d.). Retrieved March 17, 2021, from https://www.w3schools.com/sql/default.asp

Storing passwords in a secure way in a SQL Server database. (n.d.). Retrieved April 24, 2021, from https://www.mssqltips.com/sqlservertip/4037/storing-passwords-in-a-secure-way-in-a-sql-server-database/

Storing passwords securely with MySQL encryption. (n.d.). Zino UI. Retrieved April 24, 2021, from https://zinoui.com/blog/storing-passwords-securely

The HAVING and GROUP BY SQL clauses—DatabaseJournal.com. (n.d.). Retrieved April 27, 2021, from https://www.databasejournal.com/features/mysql/article.php/3469351/The-HAVING-and-GROUP-BY-SQL-clauses.htm

Twitter. (n.d.). *How the ACID Model Ensures Your Data Is Safe*. Lifewire. Retrieved April 27, 2021, from https://www.lifewire.com/the-acid-model-1019731

Use MySQL Control Flow Functions—CASE, IF, IFNULL, and NULLIF. (n.d.). Retrieved April 27, 2021, from https://www.geeksengine.com/database/single-row-functions/control-flow-functions.php

Using MySQL UNIQUE Index To Prevent Duplicates. (n.d.). *MySQL Tutorial*. Retrieved March 17, 2021, from https://www.mysqltutorial.org/mysql-unique/

What does ACID mean in Database Systems? | Database.Guide. (n.d.). Retrieved April 27, 2021, from https://database.guide/what-is-acid-in-databases/

What is CRUD? Explaining CRUD Operations. (n.d.). Sumo Logic. Retrieved March 7, 2021, from https://www.sumologic.com/glossary/crud/