Project

*DAT602 – Database Application Development*

Jackson Hayes

NMIT ID - 13521772

S2 2024

Table of Contents

[Milestone One 4](#_Toc180402207)

[Project Introduction 4](#_Toc180402208)

[Storyboards 6](#_Toc180402209)

[Home Screen 7](#_Toc180402210)

[register Screen 7](#_Toc180402211)

[Login Screen 9](#_Toc180402212)

[Main Lobby 10](#_Toc180402213)

[Game Board 11](#_Toc180402214)

[Admin Console 12](#_Toc180402215)

[Profile Screen 13](#_Toc180402216)

[Entity Relationship Diagram Rationale 13](#_Toc180402217)

[Player 13](#_Toc180402218)

[Game 14](#_Toc180402219)

[Tile 14](#_Toc180402220)

[Item 14](#_Toc180402221)

[Inventory 14](#_Toc180402222)

[Chat Session 15](#_Toc180402223)

[Chat 15](#_Toc180402224)

[SQL Queries and Procedures 15](#_Toc180402225)

[Login and Registration 15](#_Toc180402226)

[Gameplay 15](#_Toc180402227)

[Administration 16](#_Toc180402228)

[Milestone Two 16](#_Toc180402229)

[Implementation 16](#_Toc180402230)

[1. Player login, including lock out 16](#_Toc180402231)

[2. Player registration 17](#_Toc180402232)

[3. Laying out tiles on a game board 17](#_Toc180402233)

[4. Placing an item on a tile 17](#_Toc180402234)

[5. Player Movement 17](#_Toc180402235)

[6. Game play scoring 17](#_Toc180402236)

[7. Player game play acquiring inventory 17](#_Toc180402237)

[8. Move an Item (NPC effect) 17](#_Toc180402238)

[9. Kill running games 17](#_Toc180402239)

[10. Add new players 17](#_Toc180402240)

[11. Update data of a player 17](#_Toc180402241)

[12. Delete a player 17](#_Toc180402242)

# Milestone One

## Project Introduction

The game ‘BerryBlitz’ is a 2D berry picking multiplayer game that combines strategy and competition in a simple environment. The game is designed to allow 2 players to compete against each other with the goal of collecting the most berries located around the game board or reaching a predetermined score before the other person. Players will navigate a tiled map on a turn based system, moving from one tile to another, collecting berries to increase their score.

To make the game more engaging, there will also be “Poisonous Berries” scattered around the board that will move position every 2-3 turns. The poisonous berries will appear different to the regular berry tile and if collected by a player will deduct points from the player’s score.

A further analysis of the game mechanics and interactions include:

1. **Player interaction:**
   * Players interact with the game board through a point and click interface, allowing the player to move across a grid-based map. Each move should be a strategic decision as players attempt to collect the good berry tiles and avoid the poisonous ones.
   * The game begins with players starting on designated “Home Tiles”, from here they can move out further onto the board by clicking on adjacent tiles during their turn. Movement is restricted by the rule that only one player can occupy a tile at a time, which adds a small amount of strategy to the game.
2. **Map and tiles:**
   * The game map consists of a grid of tiles, each representing a specific tile type. Tiles can hold items (Berries), be occupied by players or represent a home/starting tile
3. **Item collection and inventory:**
   * Placed across the map are different items represented by tile type. This includes:
     1. a home tile, where the player will start the game
     2. a berry tile, where the player will collect a berry and increase their score
     3. a poisonous berry tile, where if collected by the player, they will lose score. These will move around the map every 2-3 turns
   * The players inventory can be represented by the amount of score/berries the player has in that specific game. This will be needed to end the game when a predetermined win condition is added.
4. **Game state:**
   * The game maintains a persistent state in the database, ensuring that all player actions or item collections are updated in real time. This allows the game to support being played by multiple people
   * When a player exits a game, their current state, including their position and score for that game are saved. Upon returning, the player can resume where they left off.
5. **Multiplayer:**
   * The game will support multiplayer interactions, with players competing against each other on the same game map. Communication between these 2 players is available through the in-game chat system.
   * The main lobby will display a list of online players and their high score.
6. **Game objective:**
   * The main objective of the game is to collect a predetermined number of berries before the other player. The first to reach this target score wins the game
   * In addition, there will be “Poisonous berries” located around the game board which will move around intermittently. Collecting one of these berries will reduce the players score and require them to collect an additional berry to win the game

## Storyboards

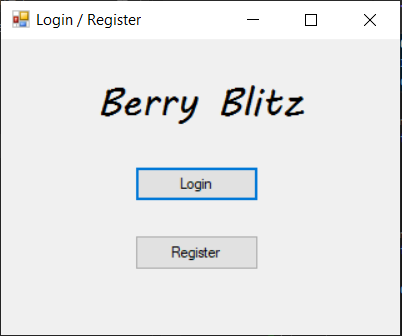
A screenshot of a computer

Description automatically generated

The general flow of the application follows the screenshot above, starting with the home screen on the far left of the screenshot, where the user has the option to either login or register an account. Each button opens up their respective form (Login button opens login page, register button opens register page) Once logged into an account the user is then moved to the main lobby where they can choose to create a new game of join an existing one. If the user account has administrator privileges, then the option to open up the admin console in the main lobby will show, otherwise it won’t be displayed to regular users.

A more detailed look into each storyboard and their components are included below:

### Home Screen



**Components:**

1. **Login Button:** Opens up the login screen window
2. **Register Button:** Open up the registration screen window

Once the application is run, the home screen (Login and Register options) will be displayed as the first screen of the application. The user has the option to either login or register a user account which will also open the respective form for whatever button is pressed.

### register Screen

A screenshot of a login form

Description automatically generated

**Components:**

1. **Register Button:** When clicked and the entered credentials are valid (don’t link to an existing account) the account will be created in the database.
2. **Cancel Button:** Closes the registration screen and returns the user to the home screen.

If the user chooses to register a user account from the home screen, they will be brought to the register page shown above. Here the user will need to enter some basic information such as a username, password and email address in order to register. If successful, the user will be prompted by a pop up window saying their account was created. They can then proceed to close this screen, opening up the home screen again and login as usual.

The potential outcomes for the register screen include:

* Entered user credentials are valid and aren’t associated with an existing account. The account is successfully created.
* Information entered is not valid and/or links to an existing account (same email or username). User account is not created.
* Information is entered into the form but cancel button is pressed before registering. User account is not created and the screen closes without saving any information.

### Login Screen

A screenshot of a computer

Description automatically generated

**Components:**

1. **Login Button:** When clicked and if login credentials are valid, the user will be redirected to the main lobby screen for the application.
2. **Cancel Button:** Returns the user to the home screen.

If the user chooses to login to an existing account from the home screen, they will be brought to login screen shown above. Here the user must enter a valid username and password, then click the “Login” button to proceed.

The potential outcomes for the login screen include:

* Entered user credentials are valid and link to an existing account. The user will be successfully logged in and redirected to the main lobby of the application
* Invalid credentials or the account doesn’t exist. A pop up window displaying an error message will be displayed and the user will need to try login again.
* The user can exit out of the login screen via the “Cancel” button, returning them to the home screen.

### Main Lobby

A screenshot of a computer

Description automatically generated

Components:

1. **Players List box** – A list of players currently registered and their high score.
2. **Active games List box** – A list of the active games that a player can join. Shows the owner of the game and the amount of people currently in that game.
3. **New Game** – When clicked, a new game will be created, and the user will be redirected to the game window.
4. **Join Game** – When clicked, the user will join the game that is currently selected in the ‘active games’ list box. If nothing is selected, the user will be prompted to select an active. If the game is full (already has 2 players) and error will be displayed telling the user that the game is full.
5. **Log Out** – When clicked, the application will exit and reopen the home screen. This will log the user out of their account and require they log in to continue.
6. **Administrator** – This button will only appear to accounts that have admin privileges. When clicked another window for the admin console will open, where the user can manage the application.

Shown above is the main lobby for the application and will appear once the user has successfully logged into an account. The lobby works as a hub for the application, giving the user an overview of other users, current games, and the option to create a new game or join an existing one. The user has the ability to log out of their account once they are finished via the “Log Out” button in the bottom left corner. If the user account has administrator privileges, the button for the admin console will appear in the bottom right corner, otherwise, the admin console button will not show for regular users.

### Game Board

A screenshot of a computer

Description automatically generated

**Components:**

1. **Game board** – Where the game will be played. Currently displayed by a placeholder.
2. **Game specific information** – Information relevant to that specific game is displayed around the screen (top left and bottom left corners). This includes the owner of the game, the total time the game has been running for and both players scores.
3. **Chat system** – Here both players have the option to chat between each other if they wish. Messages should stay consistent when loading or closing the game.
4. **Leave Game button** – When clicked, players can leave the game and return to the main lobby.

This is the general layout and expected components for each game board created by a user with the only difference between the board which is currently displayed by a large placeholder in the centre of the screen.

Each game will have its own chat session, which is shown on the right hand side, where the people playing in that game will have the option of chatting with each other.

Basic information such as who’s game it and time in game are shown at the top of the screen and scores displayed at the bottom. The user will have the option to leave the game which will return them to the main lobby of the application.

All information is expected to save and load in case users wish to continue with the game at a later date. This includes the current game state, chat messages, score and time.

### Admin Console

A screenshot of a computer

Description automatically generated

**Components:**

* **Players List** – A list of all the player accounts registered in the application. The admin can select a record by clicking on its row.
* **Current Games List** – A list of all the games currently active or created in the application. The admin can select a record by clicking on its row.
* **Add player** – The admin can add a new user account via the admin console. This will open the profile screen, where they can enter user credentials similar to the register screen process.
* **Edit player** – when clicked, the profile screen will be displayed, allowing the administrator to edit the details of an existing account.
* **Delete** – when clicked, the selected row will be deleted. This can be either an existing account or active game. The admin will be prompted for confirmation before deleting/terminating.

The admin console can only be accessed by user accounts with administrator privileges and gives them the ability to manage the application by adding player accounts or editing existing ones. The admin can also choose to delete existing accounts or terminate active games.

### Profile Screen

A screenshot of a computer

Description automatically generated

**Components:**

1. **Text boxes** – Text boxes for the administrator to add or edit account details such as the username, password and email address linked to the account
2. **Submit button** – when clicked, will save the information currently in the text boxes to the database
3. **Cancel button** - when clicked, will close the profile management window and return the user to the admin console.

The Profile screen is used as a way for the administrator to manage user account details. Whether this be creating a new user account or editing the details of an existing one. Once the admin is finished the “Submit” button will save the details currently entered in each text box. The cancel button will close the profile screen and return the user to the admin console.

## Entity Relationship Diagram Rationale

### Player

The Player table stores all information related to the players who use the application. This includes login information (username, password and email), gameplay statistics (score, total berries collected, games played and games won) and their account privilege level (admin or not). Each player is uniquely identified by ‘player\_id’.

The Player and Game tables within the database are linked together through a join table due to the many to many relationship. This is because a player can join and has statistics related to many games and a game needs many players to function correctly.

### Game

The Game table tracks individual game sessions. It contains information about the game’s status, when it started, who’s turn in the game it currently is and the link to the associated chat session.

As mentioned above, the Game and Player table share a many-to-many relationship, besides this, the Game table shares relationships to both Tile and Chat Session tables. Because each game have their own separate chat sessions and a chat session cannot belong to another game, they share a one to one relationship. There is a one-to-many relationship between Game and Tile due to the game having many tiles but the tile only belonging to one game.

### Tile

The Tile table represents the individual tiles that make up the game board. Each tile is associated with a specific game and can be occupied by only 1 player. Tiles can be of different type or essentially contain an item the player can interact with (item, home tile, trap/NPC).

The Tile table is related to 3 other tables, Game, Player and Item. An Item can belong to many Tiles but a Tile cannot have many items, but only 1 at a time (be of only a single tile type at a time). The Player and Tile tables are linked through the Tile needing to know what player is or is not currently occupying it.

### Item

The Item table stores information about the different items that the players can collect in the game such as berries that increase or decrease player score. Each item has a type (berry or poisonous) and a point value that contributes to the player’s score

Other than the relationship with the Tile table mentioned above, the Item and Inventory Tables and connected through a join table due to the fact that a player will have a different inventory in each game, meaning an Item can belong to many inventory’s and an inventory can have many different items in it.

### Inventory

The inventory table is used to represent a player’s inventory which acts as a container for storing items the player collects during the game. Each player has one inventory per game they play which can also contain multiple items at once.

The player can have many inventories, but an inventory can only belong to one player.

### Chat Session

The Chat Session table represents a session of chat that is linked with a game. The chat is used for communication between players with each game having a separate chat session.

### Chat

The Chat table stores information related to individual chat messages that are sent by players during the game. The stored information includes a time stamp of when the message was sent and the text message that it contains.

The chat table is related to chat session table with a chat session containing many different messages but a message only belonging to one session. The player can also have many different chat messages, but the message can only belong to one player.

## SQL Queries and Procedures

Each table within the database was populated with a very small amount of test data to validate how the database functions and to link it to windows forms application which visually displays the game and interface. There are also some simple queries used to further validate the database creation.

### Login and Registration

To demonstrate the login and registration process and ensure it has a valid connection to the database, I have linked both the registration and login screens in the application. This means users can open the application and register an account where they should get a message back from database starting the user has been created.

Once an account has been created, the user can navigate back to the login screen where they can login to the application using the same details, they just registered an account with. Additionally, the user could log into the game using one of the test login’s that were made during database creation.

Once logged into an account successfully, the user will be redirected to the game lobby window.

### Gameplay

To demonstrate the gameplay form and connection to the database, the user, once logged into an account and on the main lobby window, can click the ‘New Game’ button to create a new game record in the database. If successful a message will be displayed stating that the game was created.

### Administration

To demonstrate the administration aspects and connection to the database, once on the main lobby window, the user can open up the administrator console using the button in the bottom right corner which redirects them to another window.

The admin console then displays a list of both registered accounts and active games. These lists change if new accounts or games are created will further validate the account registration connection by listing any newly created accounts. As far as functionality is concerned, there is no direct administrative features at this stage, only a display of the registered accounts and active games to show the connection to the database records.

The admin console feature is currently accessible by any user but will only be usable by accounts with administrator privileges in the future.

# Milestone Two

The goal of the second milestone within this project is to continue with developing a prototype application, where logic, game management and its database is implemented using TSQL procedures, C# data access objects and a GUI using Windows Forms components. This stage/milestone builds on the CRUD analysis conducted in the previous phase and focuses on developing and implementing core functionality that is required for the game. The system aims to have user management (login, registration), game play management (board layout, item placement, scoring, movement, etc.) and administrative features (adding/editing/deleting user accounts and games).

The database plays an important role in controlling the applications logic with all essential operations being held within a procedure. These TSQL procedures are then invoked from the applications DAO to interact with the database. Windows forms GUI components are then linked to the connected DAO, enabling users to perform various tasks and actions such as logging in, registering and gameplay.

## Implementation

### 1. Player login, including lock out

The **loginUser** procedure is used to validate the user’s credentials and check their status (locked, banned, etc.). After each failed login attempt, the login attempts counter is incremented, and if after a preset threshold is reached, the user account is locked, requiring the application admin to manually reset their account locked\_out status.

The application uses a corresponding DAO method to connect to the database and call the loginUser procedure, returning whether the login was successful or not. The GUI component within windows forms consists of a login form to capture the players, username and password input. When the user submits the login form, the GUI triggers the previously described DAO method. If login is successful, the application allows the user to access the game, otherwise it will display an error message or inform the user the account is locked.

### 2. Player registration

The **registerUser** procedure is used to handle the creation and insert of new user accounts into the tblPlayer table within the database. It ensures that all required fields such as username, email and password are provided and filled out by the user to be accepted.

The registerUser DAO method then encapsulates the connection to the database and called the previously mentioned TSQL procedure. It takes user input as parameters and manages inserting the new player into the system. The GUI component consists of a registration form which collected the user’s data (username, email and password) and once submitted, triggers the registerUser DAO method. Feedback is given in the form of a success/error message if registration fails (account already exists) or if required fields are left empty.

### 3. Laying out tiles on a game board

The **makeBoard** procedure generates the initial layout of the game board by creating and organizing tiles based on a preset board size within the tblGame table. The procedure creates individual and unique tiles depending on the board size specified (10x10 board size will make 100 tiles for that game id). The procedure has an internal call to the getTileType function which returns a tile type (item) to be randomly placed on the tiles during board generation. This gives each game a unique and random placement of items, bringing in a changing dynamic to the application.

The makeBoard DAO method calls the previously mentioned TSQL procedure to initialize the game board. It sets parameters like the number of rows and columns and manages the boards structure in the database. In this early iteration of the game, the user currently has no ability to change the board size within the application/GUI. The GUI sends the preset data when the Create Game button on the Lobby form is clicked, which uses the makeBoard DAO method to generate a new board/game.

### 4. Placing an item on a tile

Due to the makeBoard procedure handling the items/tile types during board generation, a procedure for placing an item on a tile seemed a little irrelevant, although a procedure was created to meet the assessment criteria. The **placeItemOnTile** procedure allows a user to set a specific item on a tile. It takes in a item id and tile id as parameters and simply replaces the existing item on that tile with whichever the user gives in the parameter.

The placeItemOnTile DAO method is responsible for calling the SQL logic that places an item on a specific tile. It handles updates to the game state where items (berries, thorns, null) are placed on tiles. The current iteration of the game has a very basic and probably unnecessary GUI component to implement this TSQL procedure in the way of a button within the game form which triggers the placeItemOnTile method to update the item on a tile.

### 5. Player Movement

The **movePlayer** procedure handles player movement and checks for if the player is able to move to the new tile or not (if adjacent to the player’s existing location). The player can move to any new tile within a 1 tile radius around there current location and will receive an error if trying to move out of bounds or to a tile that is not adjacent. The procedure takes in the game id, player id and new tile location (row and column) as parameters.

The movePlayer DAO method handles updating the players position in the database. It checks if the move is valid and then updates the player’s current tile position in the given game. Since there is no game board GUI in the current application iteration, the games GUI captures player movement using a button which triggers the movePlayer DAO method and if successful, the players position is updated and a success message is displayed, otherwise an error.

### 6. Game play scoring

Gameplay scoring is tied into player’s moving to a tile that has an item on it and acquiring this item into their Inventory, removing it from the tile. The scoring procedure **updatePlayerScore** is internally called within the player movement procedure which gets checked each time a player moves to a tile that is not empty. The updatePlayerScore procedure adjusts the players score based on which item is acquired during gameplay.

The updatePlayerScore DAO method calls this TSQL procedure to update the players score in the database based on actions that occur during the game (collecting berries, running over thorns, etc.). Since the scoring is handled within another procedure, to demonstrate its functionality separately to the rest of the application, the GUI uses a button to update the player’s score in real time. The player and item that is acquired is currently preset in the application to demonstrate this TSQL procedure.

### 7. Player Acquiring Inventory

The player gameplay acquiring inventory functionality is tied into the movePlayer procedure and gets internally called once a player moves onto a tile with an item on it, but for the criteria of the assessment a separate procedure with the name of **acquireItem** was implemented to demonstrate the item acquisition independently to other application procedures. The acquireItem procedure takes in a player id and tile id as parameters and essentially retrieves the item associated with the tile id and adds it to the inventory of the player matching the parameterized player id. The acquired item is then removed from the tile and item id of the tile set to null.

The acquireItem DAO method adds items to that players inventory and updates both the tile which the item was on and the players inventory record in the database. As mentioned previously, the gameplay acquiring inventory is intended to be called within the movePlayer procedure but for the demonstration of the assessment, a button was added to the GUI and when clicked, interacts with the acquireItem method. The item is removed from the preset tile id and added to the preset player’s inventory.

### 8. Move an Item (NPC effect)

The **moveThorns** TSQL procedure is used to move items (NPCs) on a given game board. The logic affects the game dynamics by changing the game board layout and flow after there have been 5 player moves (5 calls of the movePlayer procedure). The procedure follows similar structure to randomly placing items on a tile but instead, retrieves all tiles with a thorn item and replaces it with another item id and then moves the previous thorns to another tile within the board.

The moveThorns DAO method calls the same TSQL procedure to move the items (NPCs) on the board. It updates the position of most but not all items on the board. The GUI component for this is reflected as a button, and when clicked, the thorns for a predefined game id are moved around the board to a random location. In the final implementation this procedure will likely be called during the movePlayer procedure to streamline efficiency and keep the game running smoothly.

### 9. Kill running games

The **deleteGame** procedure is used to kill a game whether it be active or inactive and removes it from the tblGame table within the database. While this can be seen as not ‘killing’ a running game but deleting, I thought it fit the structure of the application better, as I want games to either be joinable or finished (ready for deletion).

The deleteGame DAO method handles removing the selected game record from the database, effectively ending the game and deleting any associated data from the database. The GUI component for this can be seen in the admin console form and shown as a ‘Delete’ button which removes the selected row from the games list box when clicked. A confirmation message is displayed and needs to be accepted for finally deleting the record.

### 10. Add new players

The **addUser** procedure follows almost identical structure and SQL code as the registerUser procedure. The difference between the two is one is for the user of the application and one is for the administrator. The addUser procedure inserts a new player/user account into the tblPlayer table, adding relevant details such as username, email and password.

The addUser DAO method calls the procedure to add a new user account to the system. The GUI within the admin console provides an interface for admins to add new players. The TSQL procedure is then called once the new user account details have been filled out and submitted.

### 11. Update data of a player

The **updatePlayerProfile** procedure involves running an UPDATE query to modify an existing user account’s details in the tblPlayer table of the database. The procedure is used for admins of the application to edit user information without having to register a new account.

The updatePlayerProfile DAO method updates any changed player details in the database, such as username, password and whether their account is banned or locked. The GUI provides the user a form for editing account details and when submitted, the updatePlayerProfile procedure is triggered to apply the new changes to the database record.

### 12. Delete a player

The **deletePlayer** procedure follows identical structure and SQL to the deleteGame procedure. The difference being the deletePlayer procedure takes in a player id parameter and requires the admin has an existing user selected in the players list box of the admin console.

The deletePlayer DAO method calls the necessary SQL to remove the selected player from the database. The GUI component allows administrators to select a record from the players list box which was populated from all entries in the tblPlayer database table. Once the delete button is clicked, the method is called and the user is removed from the system.