Contents

[Introduction 1](#_Toc175906407)

[Game Description 1](#_Toc175906408)

[Gameplay 2](#_Toc175906409)

[Objective 2](#_Toc175906410)

[Authentication 2](#_Toc175906411)

[Lobby System 2](#_Toc175906412)

[Administration 2](#_Toc175906413)

[Requirements: 2](#_Toc175906414)

[Story Boards 4](#_Toc175906415)

[Login Form 4](#_Toc175906416)

[Signup Form 5](#_Toc175906417)

[Main Menu (Normal User) Form 5](#_Toc175906418)

[Main Menu (Admin User) Form 6](#_Toc175906419)

[Browser Form 6](#_Toc175906420)

[Chat Form 7](#_Toc175906421)

[Modifying Player Form 7](#_Toc175906422)

[Administrator Form 8](#_Toc175906423)

[Gameplay Form 8](#_Toc175906424)

[Logical ERD 9](#_Toc175906425)

[CRUD Table 9](#_Toc175906426)

[GitHub Repo 9](#_Toc175906427)

# Introduction

## Game Description

The inspiration for this game idea is derived from the snakes and ladders board game. To ensure the requirements are met changes will be made to the original game.

The board will be randomly generated spawning various tile types randomly. These include tiles that boost the player forward, move the player back, or give a random item to the player.

Items will be described through the following categories: Buff, de-buff, effect

Buff – will provide positive effects to the player e.g. next roll is doubled

De-buff – will provide negative effects to the player e.g. next roll is divided by 2

These will be able to be applied to other players on the board along with the player who has them in their inventory.

### Gameplay

Players start on the home tile and are to roll a 6-sided dice. The result of their roll is how many tiles they will move forward. There will be varying pathways which the player will be allowed to choose between when they arrive at them.

If a player will land on a tile with a player already on it, they will notified they cannot move there and will have to roll again. This game does not follow a turn based game and is essentially a race to the end.

### Objective

The goal of the game is to make it to the end of the board first.

## Authentication

Users will be prompted with a login screen upon the opening of the application. Users must enter pre-existing login details to continue. If a valid username with an incorrect password is provided the user will have up to 5 times to retry until the associated account is locked. If an invalid username is provided the user will be able to create an account through the sign-up screen.

## Lobby System

Once a user has been authenticated, they will be able to either create or join a lobby. Creating a lobby will generate a unique code and in which settings can be configured surrounding the game rule by the lobby owner (The user who created the lobby). A lobby can be joined by entering the code of an existing lobby into the join lobby menu.

## Administration

An administrative user will be given extra permissions surrounding the backend management of the game. An administrative user will be able to kill running games, add new players, update data of existing players, remove existing players. This will all be accessible through the admin user interface.

## Requirements:

Playing the game on a screen.

1. The game is to be played on a live/real time 2D “point-and-click” tiled map as follows:
2. Players move around the map from one tile to another, collecting items - these are put into their
3. inventory (for example jewels) and competing with other players who are playing on the same tiled
4. map. Note they are not running the same install of the App on the same device.
5. A player is logged in and registered as described in detail below.
6. Players move from one tile to another by clicking on neighbouring tiles.
7. Players start on the “Home Tile”. Apart from the “Home tile” only one player can be placed on each
8. tile at time. The player who achieves the first “click” on an empty tile moves to that tile. Once on a tile
9. the player can click on the items on that tile to gain or lose points. When a player moves from the tile,
10. the tile becomes empty, but any items they have left behind remain on the tile.
11. When a player leaves the game, their current state is kept in the database.
12. When the player returns to the game if the tile, they were on is currently empty, they return to the
13. tile on the map they were on when they left the game, otherwise the player must choose a different
14. tile to continue playing the game.
15. Players accumulate items in an inventory that may be used in the game play. The database keeps
16. track of player inventory, where the items are on the board, and the state of the player.
17. As the player moves, the position of the player is to be stored in the database.
18. Some game items can move; the position of the item is to be stored in the database.
19. Players can communicate with other characters through text chat
20. A player can “delete” their account.
21. A player can be an Administrator – see details below.

Login and Registration – to support multi-user play.

1. Your game is required to keep track of players and their scores on the “remote database”. Players
2. are not required to register; however, the system automatically registers players when they first try
3. to log in as follows:
4. When a new player name is entered, your system is to detect the new name and prompt the
5. new user to register as a player.
6. On accepting a new registration, the player’s name and details are added to the list of
7. players currently online, i.e., successful registration is treated as a successful log in.
8. If the player’s name is an existing name, the player is prompted for a password and the
9. password they submit is checked to see if that password is correct.
10. After five tries the player is locked out, and an administrator email is presented.
11. Page 12 of 28
12. If an existing player succeeds at logging in, the player’s name and details are added to the list
13. of players online.

Administration Interface

An administrator interface is to be created that includes an interface for managing locked out

players. The administrator can:

1. kill running games,
2. add new players
3. update data of existing players or
4. remove existing players.

# Story Boards

## Login Form

A screenshot of a login screen

Description automatically generated

This is where players login, if they exceed the defined amount of tries in the login procedure they will be locked out of their account.

## Signup Form

A login screen with black text

Description automatically generated

This is where players can register providing a username and password, if the username already exists in the database they will be unable to register and will have to use a different username.

## Main Menu (Normal User) Form

A screenshot of a computer screen

Description automatically generated

This is the main menu for normal accounts, this allows players to navigate to the various other forms, for browsing, exiting the game, or deleting their account.

## Main Menu (Admin User) Form

A screenshot of a video game

Description automatically generated

This is the main menu for normal accounts, this allows players to navigate to the various other forms, for browsing, the admin interface, exiting the game, or deleting their account.

## Browser Form

Screens screenshot of a game

Description automatically generated

This allows players to either join a game using the provided code, find an game that still needs players, or create a game.

## Chat Form

A screenshot of a message

Description automatically generated

This is the chat form where players can communicate with other players. The chat is one per game, so all players in each game can communicate.

## Modifying Player Form

A screenshot of a computer

Description automatically generated

This is the player editing form where administrator users can modify player data.

## Administrator Form

Screens screenshot of a game

Description automatically generated

This is the administrator form where administrators can manage existing games and players. They can kill running games, and modify and delete accounts.

## Gameplay Form

A screenshot of a computer screen

Description automatically generated

This is the gameplayer form where all the tiles are laid out. The main game area is display in the top right of the screen, where the different tiles are displayed.

The green tile is the home tile where all players start, gray tiles have no special features, pink tiles randomly give the player an item, red tiles send the player back a random number of tiles, and the blue tiles send the player forward a random number of tiles.

The player can roll the dice at the bottom of the screen at any time, this will calculate how far they have to move, validate if the player can move there, and then move the player.

The player can access the chat form using the button on the bottom right.

The player can leave the game using the leave game button on the bottom right.

The player can view and use what is in their inventory shown on the right side of the form.

# Logical ERD

A diagram of a computer

Description automatically generated

# CRUD Table

Google Sheets Link: [CRUD TABLE](https://docs.google.com/spreadsheets/d/1k3YG2LgZCModdobcrpfYeA7nqDoKKkm-/edit?usp=sharing&ouid=106832273171546172298&rtpof=true&sd=true)

The CRUD table defines what procedures will be required for the application to work correctly, each of these can be broken down into what tables and columns they will modify and with what CRUD operation.

# Data Access Object

A data access object is the method in which the C# Application will contact the database to perform the CRUD operations outlined earlier. A data access object handles the client-side error handling, translation between SQL datatypes to .NET datatypes and as an abstraction layer between the database and the client application.

Before the implementing the final GUI system, each of the procedures outlined in the CRUD table had a respective method created in a data access object.

These were separated into three categories: authentication, admin, gameplay.

Due to the ever-changing nature of the development of this project the exact state of the data access method during this time have been lost, however the GUI to test these methods remained intact, shown below:

A screenshot of a computer

Description automatically generated

The test data provided to each of these procedures is shown below:

A screen shot of a computer program

Description automatically generated

# Error Management

Error management within MySQL is done through various methods, including exit handlers, if statements, and try and catch. The primary method used during the development of this project was if else statements, ensuring that all modifications to the database were being correctly checked before committal.

To further upon this in future iterations ensuring that specific edge cases are being handled by exit handlers and try catch blocks would be crucial to catching errors that cannot be prevented by simple if else conditions.

# Concurrency Management

Concurrency management is crucial to take into consideration during the development of database such as this one. High levels of multi-user activity require careful management of the data integrity and error handling.

Different approaches to concurrency management can include optimistic and pessimistic. These different approaches are suited for different scenarios and situations, that can require high levels of data consistency and concurrency, or a lack of interference between data operations. These factors can change how the system should be designed and how it will eventually perform.

The optimistic approach is suited for situations where conflicts between transactions is minimal, as it allows concurrently without locking tables, or rows. This allows for a more simple and performant design to concurrency but can result in conflicts occasionally occurring.

In contrast the pessimistic approach is suited for high concurrent applications that require high levels of data integrity and prevents any conflicts from arising in the system. It achieves this through locking tables and isolating transactions from one another, this ensures that transactions will not conflict with one another. This can result in a more complex and overall, less performant design but is often natively supported by many DBMS’s.

## Concurrency Management in MySQL:

Concurrency management in MySQL consists of a transactional behaviour and the different isolation levels that can be used surrounding a given transaction. The isolation level of a transaction describes how isolated the data is between different transactions, for example two transactions read and update from the same table, should they be able to update at the same time, or read data that might get updated?

Isolation levels can be described as:

Read Uncommitted: Allows reads of all uncommitted changes within between transactions, offering no isolation between transactions. This results in dirty, non-repeatable and phantom reads.

Read Committed: Ensures that all data read has been committed. Results in no dirty reads, some non-repeatable reads and phantom reads.

Repeatable Read: Ensures that all referenced rows are locked, ensuring that no other transactions can interfere. This ensures that there are no dirty reads, non-repeatable read but may result in some phantom reads.

Serializable: The highest level of isolation, ensuring that all transactions are fully isolated from one another resulting in no dirty, non-repeatable, and phantom reads.

Benefits to a high isolation level include, improved concurrency, and more consistency across the database, reducing conflicts and data anomalies.

In the case of this project the Repeatable Read isolation level was chosen due to the application not requiring 100% perfectly accurate information at all times, phantom reads resulted from this are quickly, discarded in later transactions and prioritising performance over 100% data accuracy and consistency was desired.

## Concurrency Management in .NET:

Concurrency Management in the .NET environment can be done through two main methods the direct database connection and the through the built-in Entity Framework.

Direct Connection: The direct connection involves creating data access objects that execute SQL commands on the database, and then translating these through .NET’s LINQ to a .NET datatype able to be worked with in the client application.

Entity framework: The entity framework offers and entity data model relationship between the database and the application. .NET offers a GUI interface to aid with the setup process of these models and how they connect to the database.

The direct connection method was chosen for this project as it offers a deeper control of the connection between the database and the client application and how each query and transaction is translated into .NET datatypes and objects. It also follows a pessimistic approach to concurrency management as it will call a transactional procedure defined in the database. LINQ was used to translate between the database datatypes and the .NET client application datatypes.

Comparing this to the entity framework which follows the optimistic approach, which is not really suited for this project and due to limited customizability of the entity framework the direct connection was chosen.

# Reflection

In all honesty I do not believe the handed in project is to a very high quality, there are various missing, unpolished and buggy components to the hand in. In a future iteration or if a more adequate time management strategy had been used the following features would have been added:

* Effective classes and encapsulation (Player, Tile, etc) with required methods and properties, ensuring synchronization between database and client application.
* More verbose and effective error handling, ensuring a more stable connection to the database would have helped with this, through the step described above. This includes error handling on both the client and the database, including try catch and exit handlers.
* More consistent data access methods and translation between MySQL datatypes to .NET datatypes.
* Lobby system, allowing for players to join before starting a given game with a count down
* Cool down on rolling the dice.
* Animations for player movement and not just immediate teleporting.
* A more simplistic approach to map generation and traversal, allowing for a more simplistic UI generation on the client application.
* Overall, more polished and refined object oriented approach and codebase.

## GitHub Repo

GitHub Repo: [Ollie-Moss-DAT602](https://github.com/Ollie-Moss/DAT602)