7/24/2024

Frederick Laroche

NMIT

Game Project

DATA-602

Table of Contents

[Project Mow-Tivated 2](#_Toc176164424)

[Overview of the game: 2](#_Toc176164425)

[In Depth - Description 2](#_Toc176164426)

[Gameplay mechanics: 2](#_Toc176164427)

[Scoring and Leaderboard 3](#_Toc176164428)

[Scoring System: 3](#_Toc176164429)

[Screen Design & Storyboarding 3](#_Toc176164430)

[Player Login, Player Registration, Locked-Out, Player Deletes Own Account 3](#_Toc176164431)

[Laying out tiles on a game board and player movement 5](#_Toc176164432)

[NPC (tile\_type : gnome) 7](#_Toc176164433)

[Player Game Scoring 8](#_Toc176164434)

[Player inventory, item location and placing item on a tile(tile\_type) 9](#_Toc176164435)

[Admin Interface 10](#_Toc176164436)

[Database Design 11](#_Toc176164437)

[Entity Relationship Diagram (ERD): 11](#_Toc176164438)

[Table Descriptions: 12](#_Toc176164439)

[CRUD Analysis 16](#_Toc176164440)

[References : 19](#_Toc176164441)

Project Mow-Tivated

## Overview of the game:

The game is about lawnmowing where players mow patterns on a tile-based map. The player needs to completely mow a specific pattern (rectangle, square, or diamond) as quickly as possible while avoiding obstacles like rocks and moving garden gnomes.

The players will earn points based on completion speed, remaining health point minus penalties for obstacles hit.

In single-player mode, the goal is to get the highest score possible. Same thing in multiplayer mode, whereas players compete against each other, and the one with the highest score is the winner.

There will be a global chat feature to allow players to communicate during the game or while waiting in the “lobby”.

There will also be a leaderboard to display the top player’s scores.

*(\*See Scoring system below)*

## In Depth - Description

### Gameplay mechanics:

Players move on a grid one tile at a time, either by clicking on an adjacent tile of their lawnmower or using keyboard arrows(I hope this is possible).

The tile has many types :

|  |  |
| --- | --- |
| Tile\_type | Description |
| pattern\_tiles | Mow these to gain points by finishing the pattern. |
| rock\_tiles | Hit these and the player lose 5 points and 1 health point. |
| gnome\_tile | Hit these and the player lose 5 points and 1 health point. |
| heart\_tiles: | Restores health. If health is full, the heart is stored in the inventory. +9\*/  If health is not full , get stored as empty heart in inventory. |
| big\_blade\_Tiles | Allows the players to mow three tiles at once instead of one. It also gets stored as an item in the inventory under the name “Bigger Blade”. |
| home\_tile | Where both player starts the game and is the only tile that both player can be on at the same time. |
| Mowed\_tiles |  |

## Scoring and Leaderboard

### Scoring System:

Scores is based on player performance: time taken, health remaining minus penalties for hitting obstacles.

Scores are recorded in the score table, linked to specific players id and game id. This data is used to create the leaderboard.

#### Time :

|  |  |
| --- | --- |
| Time | Point |
| 10 sec | 100 points |
| 20 sec | 90 points |
| 30 sec | 80 points |
| 40 sec | 70 points |
| 50 sec | 50 points |
| 60 sec | 30 points |
| +60 sec | 1 points |

#### Obstacles :

|  |  |  |
| --- | --- | --- |
| Obstacle | Score | Health Point |
| Gnome | -5 | -1 |
| Rock | -5 | -1 |

#### Bonus Point for each health point left when finished :

+5 points for each health point

## Screen Design & Storyboarding

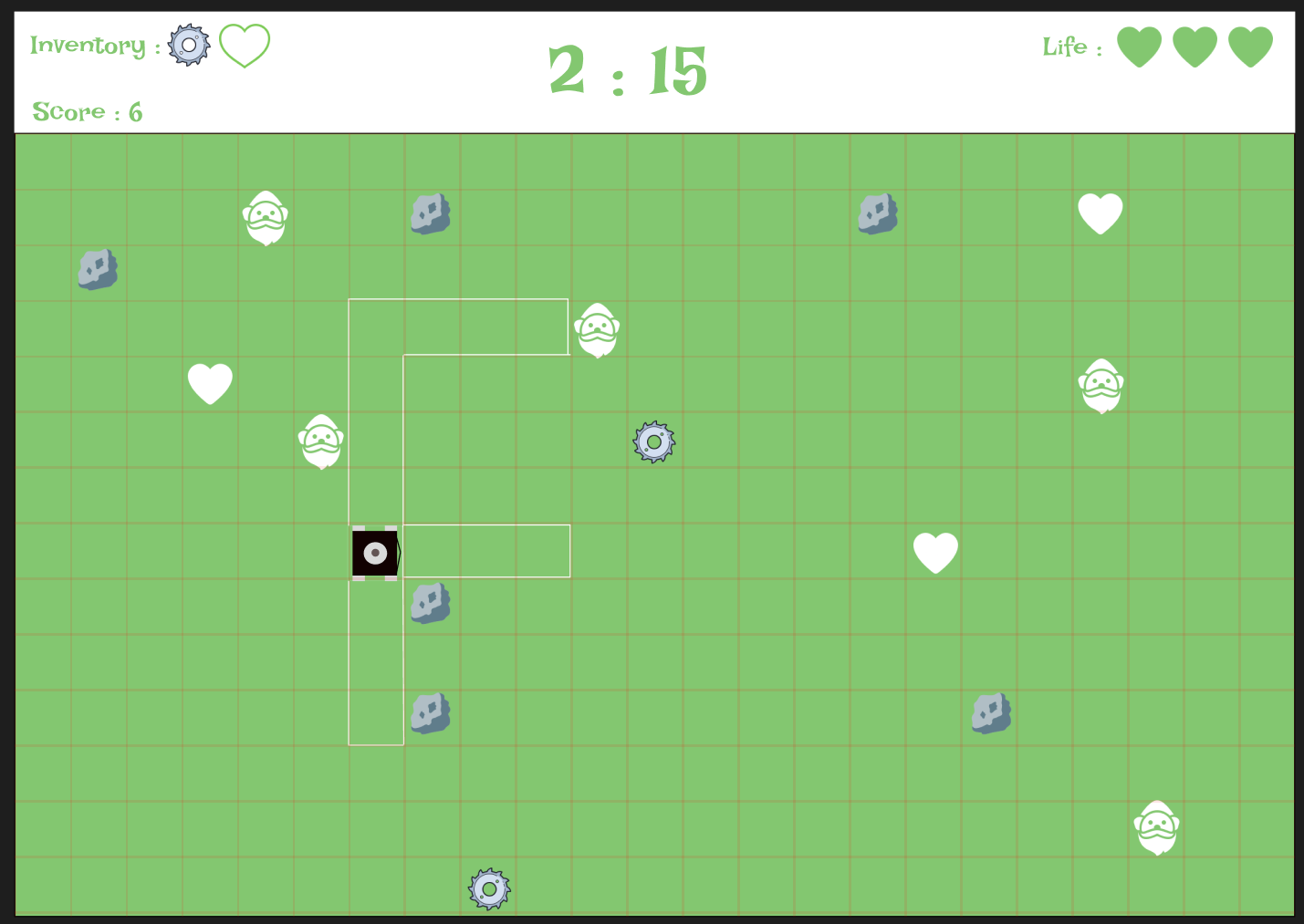
### Player Login, Player Registration, Locked-Out, Player Deletes Own Account

A screenshot of a login form

Description automatically generated

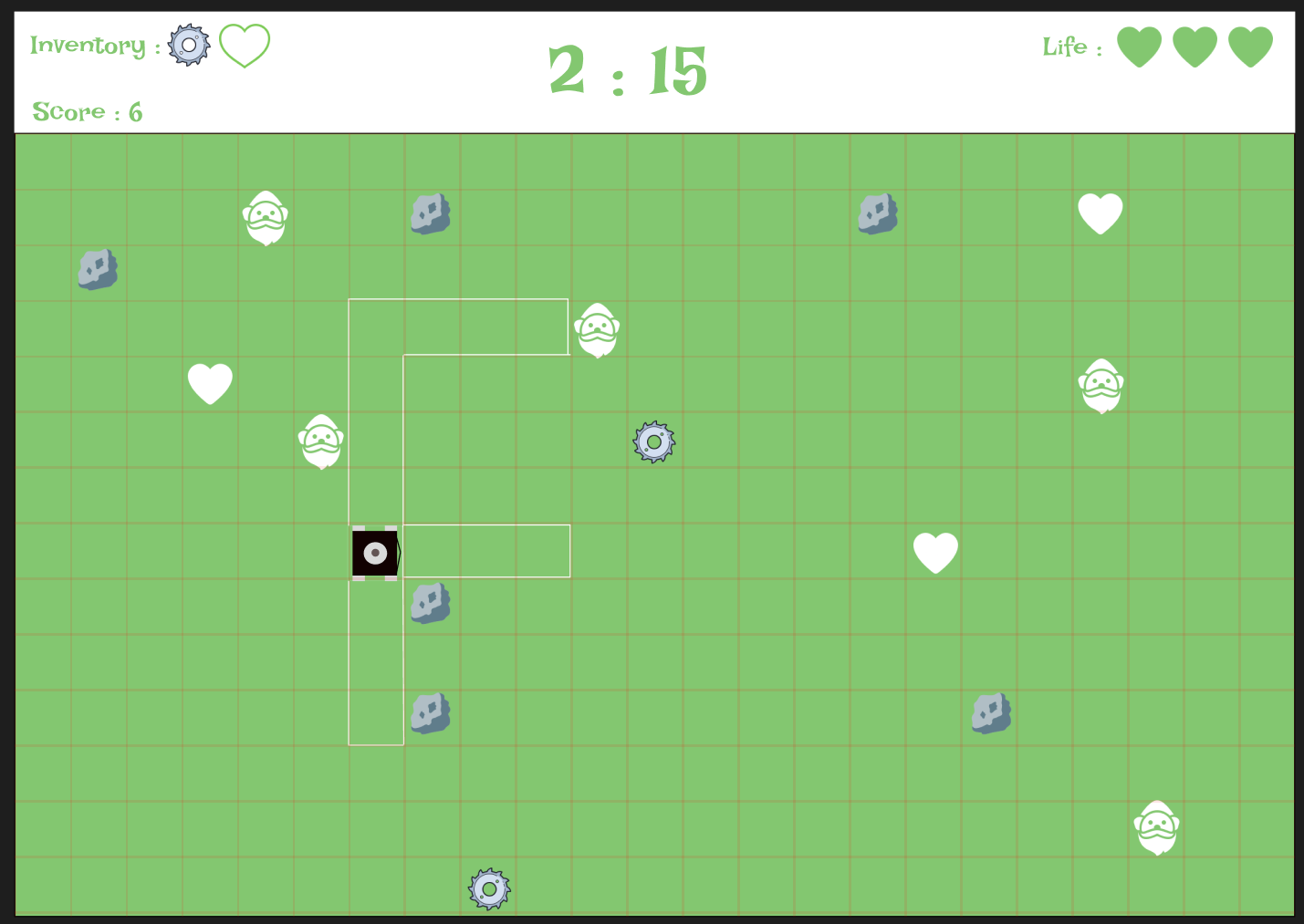
|  |  |  |
| --- | --- | --- |
| Player login | | |
| Description | **User stories** | **Operations** |
| Players need to enter his username and password to log into the game.  Steps:  The system checks if the username exists in the database.  If the username exists ,ask the user to enter their password. If the password is correct, they are logged in.  If the username does not exist already, give the player a "Creating a new account?" message with the options Yes or No.  If the player fails 5 consecutive login attempts, they are locked out.  The Yes or No options is to check if the player have made a mistake in their username as they could only have miss type. | *"As a player, I want to be able to log in so that I can access my player account and play the game."* | **SELECT:** Check the player table if the username exists.  **SELECT:** Check the player table for the player's status to make sure he isn’t ban or lock-out.  **SELECT:** If the username exists, check if the password matches the data recorded in the player table.  **UPDATE:** If login fails, add a login\_attempt for the player in the log table.  **UPDATE:** If the login is successful and before the player is locked out, reset the login\_attempt to 0.  **UPDATE:** Modify the player status to "online" in the player table on login.  **CREATE**: player\_session\_start timestamps on login in the log table.  **CREATE:** Player\_session\_end timestamp when player log out or quit the game.  **Lock-out**  **UPDATE:** When the player hits the maximum consecutive failed login attempts (5 times), update the status in the player table to "locked-out.".  **SELECT:** Get the administrator's email and display to the player on lockout. |
| Player registration | | |
| Description: | **User stories** | **Operations** |
| New players can register by entering their new username in the login form. The system will then check if the username already exists.  If the username doesn’t exist, the player will get a "Creating a new player?" message with Yes or No option.  After clicking Yes, the system will ask for a password to create the new player account. | *"As a new player, I want to register so that I can create an account to tie my data to and start playing the game."* | **CREATE:** Add the player's details into the player table (username, password).  **UPDATE:** Update the player's status to "online" in the player table.  **CREATE:** Add a login\_timestamp in the log table for the new player. |
| Player lock-out: | | |
| Description: | **User stories** | **Operations** |
| After 5 consecutive failed login attempts, the player's account will get locked-out.  The system will then send an email to the player on how to reset their account status.  Also, the player will have the option to click an "Ask for Help" button to contact an admin for assistance. | *"As a player, I want to understand why I can’t log in and how I can unlock my account"*  *“As an admin , I want to be able to change a player status to active.”* | **SELECT:** Get the administrator's email and display to the player on lockout.  **UPDATE:** In the admin email , there will be a link (I guess if possible)that can reset the player's status to "active" .  **OR**  **SELECT:** An admin can check the player's status in player table to verify if the player is either banned or locked-out.  **UPDATE:** When an admin reset the player's status to "active" in the player table . |
| Deletes own account | | |
| Description: | **User stories** | **Operations** |
| Players can delete their own account from the profile form. | *"As a player, I want to be able to delete my account and all my data if I no longer want to play this game."* | **DELETE:** Delete in cascade all the records associated with the player such as : inventory, score, tile, game, player table, log , tile location ,map and player\_chat. |

### Laying out tiles on a game board and player movement



|  |  |  |
| --- | --- | --- |
| Laying out tiles on a game board | | |
| Description | **User stories** | **Operations** |
| The game board is a tile-based grid where each tile represents different type such as grass, mowed tile, rocks, gnomes, hearts and saws.  The tile\_type of each tile will create the game environment and pattern to mow. | *"As a player, I want to play the game and see the different tiles to mow the pattern as fast as possible."* | **CREATE**: Insert data into the tile\_location table, to assign each tile a tile\_id, tile\_location and a tile\_type (ex: grass, rock, gnome, home).  **SELECT**: Get all the records above to initialize the game.  **UPDATE**: As the game advance, update the tile\_location and tile\_type changes (ex: grass is mowed, a rock is destroyed, or gnome is moving). |
| Player Movement | | |
| Description | **User Stories** | **Operations** |
| The player controls a lawnmower that moves one tile at a time on the game board.  The player can click on a tile adjacent to the lawnmower or use the arrow keys on the keyboard to move(*if it is possible*).  The game will then check the tile\_type to affect or not the player appropriately. | *“As a player, I want to move my lawnmower on the game board”*  *“As a player, I want to get affected by the type of tile I move on.”*  *“As a player, I want the game to stop iif my health drops to zero.”* | **SELECT:** When the player moves, the system checks the tile\_type of the tile and the tile\_location.  **If tile\_type is rock or gnome:**  **UPDATE:** Move the player to the tile and update the rock/gnome tile\_type to mowed.  **UPDATE:** Reduce the player's health by 1 in the player table.  **UPDATE:** Reduce 5 points from the player's score in the score table. If the score is negative, set it to 0.  **UPDATE:** Change the tile type to “mowed.”  **If tile\_type is heart:**  **UPDATE:** Increase the player's health by 1 (if not full).  **UPDATE:** Change the tile\_type “heart” to “mowed” from the tile\_type table.  **INSERT:** Add the heart to the player's inventory(if full health, add a full heart. If not full health , add an empty heart to the inventory.  **If tile\_type is grass\_pattern:**  **UPDATE:** Move the player to the tile and update the tile type to “mowed.”  **SELECT/UPDATE:** After each move, check the player's health points. If health is greater than 0, continue the game . If player’s health = 0, create the end\_time of this game.  **If tile\_type is bigger\_blade:**  **UPDATE:** Equip the player with a big blade, increasing mowing area(From 1 to 3 horizontally).  **UPDATE:** Change the big\_blade tile\_type to mowed.  **CREATE**: Add it to the player's inventory. |

### NPC (tile\_type : gnome)



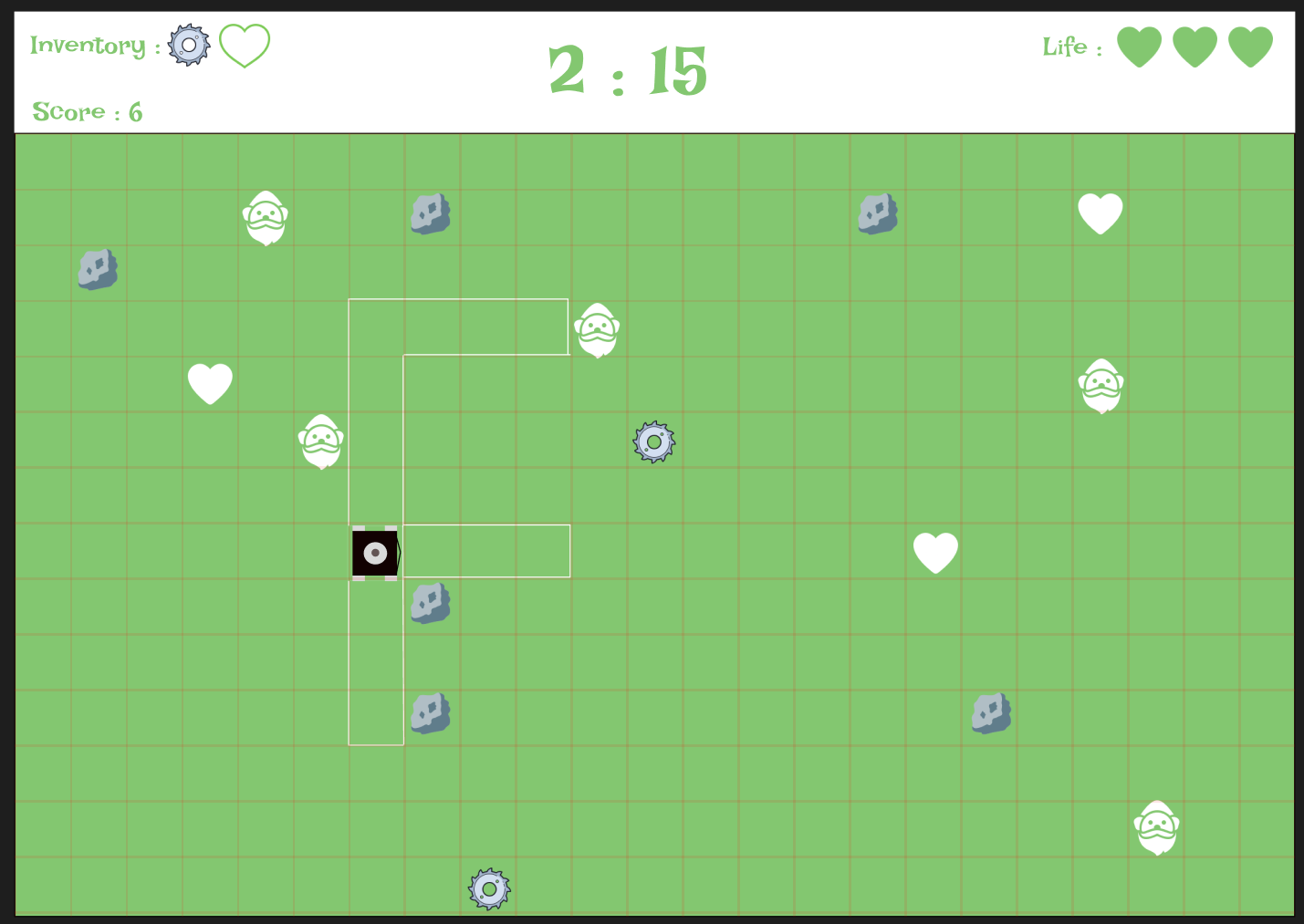
|  |  |  |
| --- | --- | --- |
| Gnome NPC | | |
| Description | **User stories** | **Operations** |
| Gnomes are a tile type. They are moving obstacles, to add a layer of difficulty.  If the player’s lawnmower moves on gnome tile, the player loses health and points.  Gnomes add a layer of difficulty. | *“As a player, I want to gnome to affect me when I come in contact with my lawnmower.”*  *“As a player, I want the gnome to move around the game board to feel some difficulty.”* | **SELECT:** Get the current position and tile\_id of each tile\_type gnome from the tile\_location table.  **UPDATE:** Move the gnomes by updating their location in the tile\_location table with their id.  **UPDATE:** As gnomes move, update the tile\_type they leave to its previous type(ex: tile\_type pattern or grass). |

### Player Game Scoring



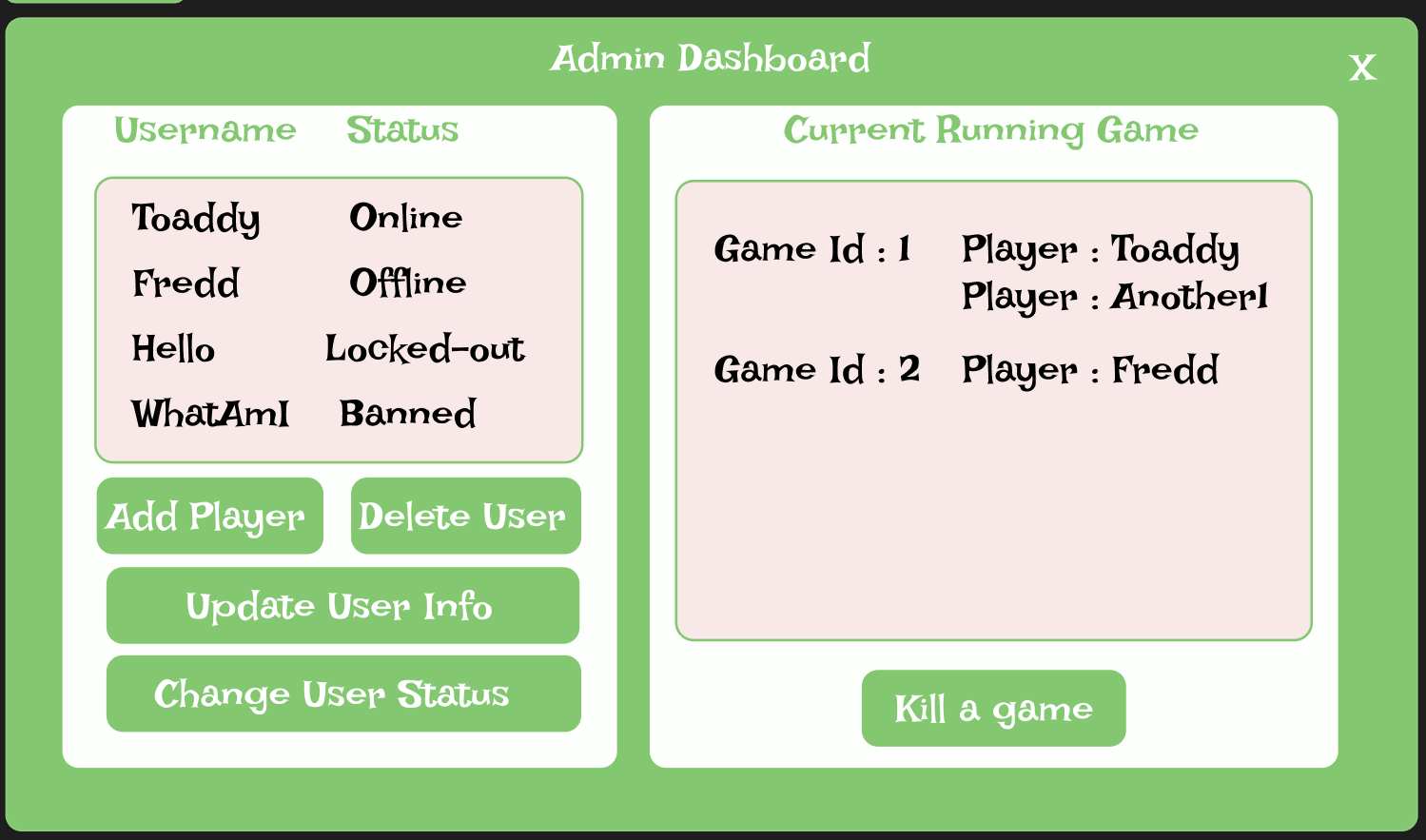
|  |  |  |
| --- | --- | --- |
| Leaderboard | | |
| Description | **User stories** | **Operations** |
| The leaderboard displays the top 5 players and their scores, sorted in desc. order.  Anyone(player and admin) can refresh the leaderboard to look at the latest highest scores. | “As a player, I want to see my score and compare it with others on the leaderboard.”  “As a user, I want to be able to refresh the leaderboard to get the latest top score.” | **SELECT:** Get the highest score, ordered by score\_value in desc. order from the score table to display the leaderboard.  **SELECT:** Get the score\_value and the player username from the player table to display player names and their scores. (Join)  **SELECT:** When the user clicks the "refresh" button, the system re-selects the info to updates the leaderboard. |
| Scoring mechanism | | |
| Description | **User stories** | **Operations** |
| Players will earn points based on their performance, such as time of completion , health remaining, and penalties for hitting obstacles like rocks or gnomes. | “As a player, I want to see my score to be updated in-game so that I know where I stand.” | **Update:** During gameplay, the player's score is updated constantly based on actions (ex: mowing grass in the pattern area, hitting obstacles) by adding or subtracting from the score\_value in the score table.  **Insert:** At the end of the game, the final score is stored in the score table linked to the player's player\_id and the game\_id. |

### Player inventory, item location and placing item on a tile(tile\_type)



|  |  |  |
| --- | --- | --- |
| Player inventory | | |
| Description | **User stories** | **Operations** |
| The inventory is showed at the top left. It shows the items collected during the game.  Only the full heart item can be used on delay (when health drop by one, it is automatically used in the inventory).  All the other items are use immediately after being picked up(Bigger saw and heart when the player has not all his health point already). | “As a player, I want to see the items I have collected in my current game and keep track of them in my inventory.” | **CREATE:** When a player step on an item(tile\_type: heart or bigger blade), it is added to the inventory table and linked to the player\_id.  **SELECT:** Get the inventory table and item table with the player\_id.  **UPDATE:** When a player uses an item, update the item table. |
| Item location | | |
| Description | **User stories** | **Operations** |
| Items like hearts, rocks, bigger blade, and gnomes are placed on specific tiles on the game board to add a layer of strategy and difficulty to the gameplay. | “As a player, I want to see where the items are located on the map so that I put a strategy in place.” | **SELECT:** Get the location of the tile\_type heart and bigger blade on the game board by searching the tile\_location table(row and column) for each tile\_id and tile\_type.  **UPDATE:** When a player moves on a tile\_type heart or bigger blade, change the tile\_type to mowed. |
| Item on a tile (tile\_type) | | |
| Description | **User stories** | **Operations** |
| Each item on the board is a specific tile\_type. They all affect the player differently . | “As a player, I want the items on the tiles to interact with my lawnmower based on the type of tile they are on, affecting my score and health.” | **SELECT:** Set the game board by getting the tile\_location ,tile id and tile\_type.  **UPDATE:** As the player moves on different tiles, the player can trigger some effects. Will need to update either health, inventory, tile\_type, mowing area or score . |

### Admin Interface



|  |  |  |
| --- | --- | --- |
| Kill running games | | |
| Description | **User Stories** | **Operations** |
| Function to kill running games when an admin user selects a game and click the "Kill Game" button in the admin interface. | “As an admin, I want to have the possibility to terminate running games to manage the system.” | **SELECT:** Get the game table where end\_time is null to get a list of current running games.  **UPDATE:** When the admin selects a game and clicks "Kill Game," update the end\_time in the game table to now. |
| Add new players | | |
| Description | **User stories** | **Operations** |
| Admin can add new players by filling out a form(username and password). | “As an admin, I want to add new players to the game easily in my interface.” | **CREATE:** Insert a new record in the player table (username, password).  **SELECT:** Check if the username already exists in the player table to prevent duplicates. |
| Update players data | | |
| Description | **User stories** | **Operations** |
| Function to update existing players' data by the admin. | “As an admin, I want to be able to update player information to manage user accounts in my interface.” | **SELECT:** Get the player details from the player table when the admin selects a player in the admin interface.  **UPDATE:** Modify the player’s details (username, password, status) and update the record in the player table. |
| Remove existing players | | |
| Description | **User stories** | **Operations** |
| Function for admin to remove existing players by deleting their account. | “As an admin, I want to be able to change a player status to manage user’s access to the game.” | **SELECT:** Get the player's status and details from the player table when the admin selects a player in the admin interface.  **DELETE:** Completely remove the player’s record from the player table and delete in cascade in the other related tables. |

# Database Design

### Entity Relationship Diagram (ERD):

### Table Descriptions:

|  |  |  |
| --- | --- | --- |
| Player | | |
| Purpose | **Attributes** | **Relationships** |
| Stores all relevant information about the players.  A player can only have one character as the player is the character and their player username is the character name. | * **player\_id:**   Primary key to identify each unique player.   * **game\_id:**   Foreign key linking the game table to identify the current game to the associated player.   * **tile\_id:**   Foreign key linking the tile table to identify the current player position on the game board.   * **username:**   The player’s username used for login and in the game.   * **player\_password:**   The player’s password for login and in the game.   * **is\_admin:**   Boolean(true or false) if the player has administrative privileges or not.   * **status:**   Current status of the player (ex: active, locked-out, banned).   * **healthpoint:**   Current player’s health points during the game. | **Relationship with game**:  A player is associated with a game via game\_id. A player can only have one game running at a time .  A game can have many player as it is needed to play multiplayer.  **Relationship with tile**:  A player’s position on the game board is tracked by their tile\_id.  A tile can have many player(Only the home tile on multiplayer mode) but a player can only have 1 tile. |
| Game | | |
| Purpose | **Attributes** | **Relationships** |
| Tracks the details of each game session. | * **game\_id:**   Primary key to identify each unique game session.   * **game\_type:**   Type of game (single-player or multiplayer).   * **start\_time:**   Timestamp of when the game started.   * **end\_time:**   Timestamp of when the game ended (null for ongoing games). | **Relationship with player**:  Multiple players can participate in a game, linked through game\_id.  A player can only one game at a time.  **Relationship with map**:  Each game has one map.  Each map can only have one game. |

|  |  |  |
| --- | --- | --- |
| Score | | |
| Purpose | **Attributes** | **Relationships** |
| Records the scores for each player in each game session. | * **score\_id:**   Primary key to identify each unique score.   * **player\_id:**   Foreign key linking the player table to indicate which player got this specific score.   * **game\_id:**   Foreign key linking the game table to indicate in which game the score was made.   * **score\_timestamp:**   Timestamp of when the score was made.   * **score\_value:**   The actual score value made by the player. | **Relationship with player**:  A score is linked to a player by the player\_id.  **Relationship with game**:  A score is linked with a specific game by the game\_id. |
| Log | | |
| Purpose | **Attributes** | **Relationships** |
| To keep track of player login attempts and log time. | * **log\_id:**   Primary key to identify each unique log entry.   * **player\_id:**   Foreign key linking the player table to indicate which player the log is from.   * **login\_attempt:**   To count the number of login attempt made by the player.   * **login\_timestamp:**   Timestamp of each login attempt. | **Relationship with player**:  Log data is linked with a specific player by a player\_id. |
| Player chat | | |
| Purpose | **Attributes** | **Relationships** |
| To store every chat messages sent by players in the global chat. | * **player\_id:**   Foreign key linking the player table to indicate which player sent the message.   * **chat\_id:**   Foreign key linking the chat\_session table, indicating the chat session the message belongs to.   * **timestamp:**   Timestamp of when the message was sent.   * **message:**   The message content. | **Relationship with player**:  A chat message is linked to a player by a player\_id.  **Relationship with chat\_session**:  A message is linked with a specific chat session. |

|  |  |  |
| --- | --- | --- |
| Chat session | | |
| Purpose | **Attributes** | **Relationships** |
| To track chat sessions that happen during game and lobby sessions. | * **chat\_id:**   Primary key to identify each unique chat session.   * **session\_start:**   Timestamp of when the chat session started.   * **session\_end:**   Timestamp of when the chat session ended (null for ongoing chat sessions). | **Relationship with player\_chat**:  Many messages can be associated with one chat session by chat\_id. |
| Map | | |
| Purpose | **Attributes** | **Relationships** |
| To define the map and the tiles linked with it for each game . | * **map\_id:**   Primary key to identify each unique map.   * **game\_id:**   Foreign key to link the game table to indicate which game this map has. | **Relationship with game**:  Each map is linked with a game session by a game\_id.  **Relationship with tile**:  The map has multiple tiles linked by the tile\_id. |
| Tile | | |
| Purpose | **Attributes** | **Relationships** |
| To store each tile and details on the game board. | * **tile\_id:**   Primary key to identify each unique tile.   * **map\_id:**   Foreign key to link the map table to which map has the specific tile . | **Relationship with map**:  Each tile is part of a map\_id.  **Relationship with tile\_location**:  The placement of a tile on the game board is made by the tile\_location. |
| Tile location | | |
| Purpose | **Attributes** | **Relationships** |
| To set the position of each tile on the board. | * **tile\_id:**   Foreign key to link the tile table to identify the specific tile.   * **tile\_type\_id:**   Foreign key to link the tile\_type table to a type of tile (ex: pattern, rock, gnome).   * **row:**   The row position of the tile on the game board.   * **column:**   The column position of the tile on the game board. | **Relationship with tile**:  The tile location is tracked by a tile\_id.  **Relationship with tile\_type**:  The type of each tile is tracked by a tile\_type\_id. |

|  |  |  |
| --- | --- | --- |
| Tile type | | |
| Purpose | **Attributes** | **Relationships** |
| To define the types of tiles that are on the game board. | * **tile\_type\_id:**   Primary key to identify each unique tile type.   * **name:**   The name of the tile type (ex: pattern, rock, gnome).     * **effect:**   The effect that the tile type has on the player (ex: +/- health, +/- score).   * **score\_value:**   The score value of the tile type if there is any (ex: pattern= 0 point). | **Relationship with tile\_location**:  Each tile type is linked to a tile on the board by the tile\_type\_id.  **Relationship with item**: Items are linked with the tile types by the tile\_type\_id. |
| Inventory | | |
| Purpose | **Attributes** | **Relationships** |
| To track the items that players have collected during the current game. | * **inventory\_id:**   Primary key to identify each unique inventory.   * **player\_id:**   Foreign key to link the player table to which player has this inventory.   * **game\_id:**   Foreign key to link the game table to the game the item is collected.   * **quantity:**   The quantity of the item held by the player. | **Relationship with player**:  The inventory is linked to a specific player by the player\_id.  **Relationship with game**:  The items in the inventory are linked to a specific game session by the game\_id. |
| Item | | |
| Purpose | **Attributes** | **Relationships** |
| To store all the item’s details that can be collected by a players. | * **item\_id:**   Primary key to identify each unique item.   * **inventory\_id:**   Foreign key to link the inventory table to the inventory the item belongs to.   * **tile\_type\_id:**   Foreign key to link the tile\_type table to a type of tile the item is associated with.   * **item\_name:**   The name of the item.   * **item\_description:**   Description of the item and his effects. | **Relationship with inventory**:  Each item is linked to a player's inventory by the inventory\_id.  **Relationship with tile\_type**:  The items are linked to a specific tile type by a tile\_type\_id. |

# CRUD Analysis

**C= Insert R= SELECT U= UPDATE D= DELETE**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Table | Attributes affected | CRUD | Description |
| Player login/log out  Including lockout | Player  log | Username  player\_password  status  login\_attempt  player\_id  player\_session\_start  player\_session\_end | **R** | Check if the username exists |
| **R** | Check the player’s status |
| **R** | Check the password if the username exists |
| **U** | Add a login attempt in the log table |
| **U** | Reset to 0 the login attempts of player on login |
| **U** | Update the player status to : "online" ,“offline” or ”locked-out” |
| **C** | Add a player\_session\_start time |
| **C** | Add a player\_session\_end time on logout |
| **U** | Update the player status to "locked-out" after 5 consecutive failed attempts |
| **C** | Add a login time in the log table |
| **R** | Get and display the admin email to the player on lockout |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Player registration | Player  log | Username  player\_password  status    player\_id  login\_timestamp | **C** | Insert the new player info in the player table |
| **R** | Check if the username exists |
| **U** | Update the player status to "online" when the account is created |
| **C** | Add a login\_timestamp in the log table |
| **C** | Add a player\_session\_start time |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Laying out tiles on a game board | tile  tile\_location  tile\_type | tile\_id    tile\_type\_id    row    column | **C** | Insert data to define each tile\_type and tile\_location on the board |
| **R** | Get the tile data to initialize the game board |
| **U** | Update the tile\_type as the game progresses (ex: tile\_type pattern to mowed, rock to mowed etc) |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Placing an item on a tile | tile\_location  item | tile\_id  tile\_type\_id  item\_id  row  column | **C** | Insert an item on a tile in the tile\_location table |
| **R** | Get the item and tile details(type and location) |
| **U** | Update the item state and location |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Table | Attributes affected | CRUD | Description |
| Player movement | Player  tile\_location  tile\_type  score  inventory  Game | tile\_id  tile\_type\_id  player\_id  healthpoint  score\_value  inventory\_id  end\_time | **R** | Check the tile\_type and tile\_location before moving to determine what will interaction is with the player. |
| **U** | Move the player to the new tile, update healthpoint if necessary, and change the tile\_type to "mowed." |
| **U** | Update the score\_value according to the tile\_type and effect. |
| **C** | Insert an end\_time for the game if the player mows the last tile\_type pattern. |
| **U** | Update inventory if necessary (ex: add heart to inventory). |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Gameplay scoring | score  player | score\_value  player\_id  game\_id  score\_timestamp | **U** | Update the score\_value continuously in the score table during the game based on the player/players interactions. |
| **R** | Get the score\_value and the score\_timestamp from the score table to create the leaderboard. |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Player Gameplay Acquiring Inventory  (ex: picking up items off a tile and putting them in an inventory) | Inventory  item | inventory\_id  player\_id  item\_id  quantity | **C** | Add an item to the player’s inventory table. |
| **U** | Update the quantity in the inventory table for the specific player. |
| **R** | Get the player's id and inventory by selecting the inventory table to show the inventory on screen. |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Move an item (NPC effect) | tile\_location  item  tile\_type | tile\_id  tile\_type\_id  item\_id | **R/U** | Get and update the row and column of the tile\_type gnome to move the NPC to a new tile. |
| **R/U** | Get the previous tile\_type of the tile and update the type by the old value when the gnome moves. |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Kill running games | game  player | game\_id  end\_time  status | **R** | Get the ongoing games from the game table where end\_time is null. |
| **U** | Update the status in the player table to set player/players as offline. |
| **U** | Update the end\_time in the game table to set the game as ended. |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Add new players | player | username  player\_password  status  player\_id | **C** | Insert new player record in the player table(username, password). |
| **R** | To make sure the username does not exist before creating the account. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Process | Table | Attributes affected | CRUD | Description |
| Update data of a player | player  score  log | username  player\_password  status  player\_id | **R** | Get the player info from the player table to do the update. |
| **U** | Modify and update the player info from the player table. |
| Process | **Table** | **Attributes affected** | **CRUD** | **Description** |
| Delete a player | player  score  inventory  log  player\_chat    game | player\_id  username  status  score\_value  score\_timestamp  login attempt  quantity  message  start\_time  end\_time | **R** | Retrieve player data before deletion. |
| **Delete in cascade** | |
| **D** | Delete the player's score\_value and score\_timestamp in the score table where the player\_id matches the selected player. |
| **D** | Delete all inventory items associated with the player from the inventory table where the player\_id matches. |
| **D** | Remove all login attempt records related to the player from the log table where the player\_id matches the selected player. |
| **D** | Delete all chat messages associated with the player from the player\_chat table where the player\_id matches the selected player. |
| **D** | Remove all the game session data(start\_time and end\_time) of the player from the game table where the player\_id matches the selected player. |

# Script

*\*I don’t know to put a MySQLl script inside a repository*

DROP DATABASE IF EXISTS dat\_602\_game;

CREATE DATABASE dat\_602\_game;

USE dat\_602\_game;

-- Note: INT in MySQL does not need a defined width like INT(10) as it will be deprecated in the next version

## -- Store procedure to create table

DELIMITER //

CREATE PROCEDURE createAllTables()

BEGIN

-- Drop and create the game table// stores all the game sessions, game type (single-player or multiplayer), start time, and end time of a game(end time is null when the game is still running).

DROP TABLE IF EXISTS game;

CREATE TABLE game (

game\_id INT AUTO\_INCREMENT PRIMARY KEY,

game\_type VARCHAR(50) NOT NULL,

start\_time TIMESTAMP NOT NULL,

end\_time TIMESTAMP

);

-- Drop and create the map table

DROP TABLE IF EXISTS map;

CREATE TABLE map (

map\_id INT AUTO\_INCREMENT PRIMARY KEY,

game\_id INT,

FOREIGN KEY (game\_id) REFERENCES game(game\_id)

);

-- Drop and create the tile table// represents each tiles on the board linked to a specific map to define the game layout.

DROP TABLE IF EXISTS tile;

CREATE TABLE tile (

tile\_id INT AUTO\_INCREMENT PRIMARY KEY,

map\_id INT,

FOREIGN KEY (map\_id) REFERENCES map(map\_id)

);

-- Drop and create the tile\_type table// define the diffrent type of tiles available in the game and the effects on gameplay or player(score,+/- health,pattern can end game when all gone)).

DROP TABLE IF EXISTS tile\_type;

CREATE TABLE tile\_type (

tile\_type\_id INT AUTO\_INCREMENT PRIMARY KEY,

`name` VARCHAR(50) NOT NULL,

effect VARCHAR(50) NOT NULL,

score\_value INT NOT NULL

);

-- Drop and create the player table // store the player data ,the game they are playing, their position on the board,status(banned,offline,online,lock-out) and login info(username,password).

DROP TABLE IF EXISTS player;

CREATE TABLE player (

player\_id INT AUTO\_INCREMENT PRIMARY KEY,

game\_id INT,

tile\_id INT,

username VARCHAR(50) NOT NULL UNIQUE,

player\_password VARCHAR(255) NOT NULL,

is\_admin BOOLEAN NOT NULL DEFAULT FALSE,

`status` VARCHAR(20) NOT NULL DEFAULT 'offline',

healthpoint INT NOT NULL DEFAULT 3,

FOREIGN KEY (game\_id) REFERENCES game(game\_id),

FOREIGN KEY (tile\_id) REFERENCES tile(tile\_id)

);

-- Drop and create the score table // store the score of each player in each game session , their time and score .

DROP TABLE IF EXISTS score;

CREATE TABLE score (

score\_id INT AUTO\_INCREMENT PRIMARY KEY,

player\_id INT,

game\_id INT,

score\_timestamp DATETIME NOT NULL,

score\_value INT NOT NULL,

FOREIGN KEY (player\_id) REFERENCES player(player\_id),

FOREIGN KEY (game\_id) REFERENCES game(game\_id)

);

-- Drop and create the log table // store the log info such as player login attempts and time to track the consecutive failed attempts(act as a counter and need to be reseted on successful login).

DROP TABLE IF EXISTS log;

CREATE TABLE log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

player\_id INT,

login\_attempt INT NOT NULL DEFAULT 0,

login\_timestamp DATETIME NOT NULL,

FOREIGN KEY (player\_id) REFERENCES player(player\_id)

);

-- Drop and create the chat\_session table // store the start and end times for each session.

DROP TABLE IF EXISTS chat\_session;

CREATE TABLE chat\_session (

chat\_id INT AUTO\_INCREMENT PRIMARY KEY,

session\_start DATETIME NOT NULL,

session\_end DATETIME

);

-- Drop and create the player\_chat table // to store all the messages sent by players and link each message to a player id and the chat session for communication during a game.

DROP TABLE IF EXISTS player\_chat;

CREATE TABLE player\_chat (

player\_id INT,

chat\_id INT,

`timestamp` DATETIME NOT NULL,

message VARCHAR(255) NOT NULL,

PRIMARY KEY (player\_id, chat\_id, timestamp),

FOREIGN KEY (player\_id) REFERENCES player(player\_id),

FOREIGN KEY (chat\_id) REFERENCES chat\_session(chat\_id)

);

-- Drop and create the tile\_location table // to track the position(row,colum) of each tile on the game board,the type of tile .

DROP TABLE IF EXISTS tile\_location;

CREATE TABLE tile\_location (

tile\_id INT,

tile\_type\_id INT,

`row` INT NOT NULL,

`column` INT NOT NULL,

PRIMARY KEY (tile\_id, tile\_type\_id), -- Composite Primary Key because of the joint many to many relationship

FOREIGN KEY (tile\_id) REFERENCES tile(tile\_id),

FOREIGN KEY (tile\_type\_id) REFERENCES tile\_type(tile\_type\_id)

);

-- Drop and create the inventory table // to track the items collected by the players during the game, the player and the game session with it.

DROP TABLE IF EXISTS inventory;

CREATE TABLE inventory (

inventory\_id INT AUTO\_INCREMENT PRIMARY KEY,

player\_id INT,

game\_id INT,

quantity INT NOT NULL,

FOREIGN KEY (player\_id) REFERENCES player(player\_id),

FOREIGN KEY (game\_id) REFERENCES game(game\_id)

);

-- Drop and create the item table // to store the item and details that player can get during the game.

DROP TABLE IF EXISTS item;

CREATE TABLE item (

item\_id INT AUTO\_INCREMENT PRIMARY KEY,

inventory\_id INT,

tile\_type\_id INT,

item\_name VARCHAR(50) NOT NULL,

item\_description VARCHAR(100) NOT NULL,

FOREIGN KEY (inventory\_id) REFERENCES inventory(inventory\_id),

FOREIGN KEY (tile\_type\_id) REFERENCES tile\_type(tile\_type\_id)

);

END //

DELIMITER ;

## -- Store procedure to insert test data

-- Now the test data in a store procedure

DELIMITER //

CREATE PROCEDURE insertTestData()

BEGIN

-- test data in the player table // players with usernames, if they are admin and their initial healthpoint.

INSERT INTO player (username, player\_password, is\_admin, healthpoint, `status`)

VALUES ('Toaddy', 'password123', FALSE, 3, 'online'),

('Fredd', 'password1234', TRUE, 3, 'online'),

('Hello', 'password12345', FALSE, 3, 'locked-out'),

('WhatAmI', 'password123456', FALSE, 3, 'banned');

-- test data in the game table // create the initial game sessions info with game type.

INSERT INTO game (game\_type, start\_time)

VALUES ('single-player', NOW());

-- test data in the map table // to linkthe map to a game session.

INSERT INTO map (game\_id)

VALUES (1);

-- test data in the tile\_type table // define all the tile type ,their effect and the score associated with this type of tile.

-- (?does my heart have a score value if only remaining heart at the end of the game should be worth 5 point each?)

INSERT INTO tile\_type (`name`, effect, score\_value)

VALUES ('Pattern', 'When all mowed, end the game', 0),

('Rock', '-1 healht point', -5),

('Gnome', 'health point', -5),

('Heart', '+1 healht point', 5),

('Biggerblade', 'Increase the mowing area', 0);

-- test data in the tile table // link the tiles with a specific map.(is this correct,need to check again?)

INSERT INTO tile (map\_id)

VALUES (1),

(1),

(1),

(1),

(1),

(1),

(1);

-- test data in the tile\_location table // place the tiles on the board with specific location to create the layout.

INSERT INTO tile\_location (tile\_id, tile\_type\_id, `row`, `column`)

VALUES (1, 1, 1, 1), -- Pattern tile

(2, 2, 1, 2), -- Rock tile

(3, 3, 2, 1), -- Gnome tile

(4, 4, 2, 2), -- Heart tile

(5, 5, 2, 3), -- Bigger Blade tile

(6, 1, 3, 1), -- Pattern tile

(7, 1, 3, 2); -- Pattern tile

-- test data in the inventory table // set the quantity of item in a specific player inventory.

INSERT INTO inventory (player\_id, game\_id, quantity)

VALUES (1, 1, 1), -- Toaddy has 1 item

(2, 1, 2); -- Fredd has 2 items

-- test data in the item table // put items collected by the players ,the effects in their inventory table.

INSERT INTO item (inventory\_id, tile\_type\_id, item\_name, item\_description)

VALUES (1, 4, 'Heart', '+1 healht point'), -- Toaddy item

(2, 5, 'Biggerblade', 'increase the mowing area'); -- Fredd item

-- test data in the score table // set some score stored in different game sessions.

INSERT INTO score (player\_id, game\_id, score\_timestamp, score\_value)

VALUES (1, 1, '2024-08-29 08:24:00', 42), -- Toaddy score

(2, 1, '2024-08-29 08:25:00', 36), -- Fredd score

(3, 1, '2024-08-29 08:26:00', 34), -- Hello score

(4, 1, '2024-08-29 08:27:00', 30); -- WhatAmI score

-- test data in the log table / the login attempts and timestamps test data for all situation(locked-out,banned and a regular successful login.

INSERT INTO log (player\_id, login\_attempt, login\_timestamp)

VALUES (1, 1, '2024-08-29 08:28:00'), -- Toaddy regular login attempt

(2, 2, '2024-08-29 08:29:00'), -- Fredd login attempt #2

(3, 1, '2024-08-29 08:30:00'), -- Hello login attempt #1

(3, 2, '2024-08-29 08:31:00'), -- Hello login attempt #2

(3, 3, '2024-08-29 08:32:00'), -- Hello login attempt #3

(3, 4, '2024-08-29 08:33:00'), -- Hello login attempt #4

(3, 5, '2024-08-29 08:34:00'), -- Hello's login attempt #5 (get locked out)

(4, 4, '2024-08-29 08:35:00'); -- WhatAmI's login attempt (player is banned)

-- test data in the chat\_session table: Creating a chat session for players.

INSERT INTO chat\_session (session\_start)

VALUES ('2024-08-29 08:24:00');

-- test data in the player\_chat table: Storing player chat messages in the session.

INSERT INTO player\_chat (player\_id, chat\_id, `timestamp`, message)

VALUES (1, 1, '2024-08-29 08:37:00', 'Hello world'),

(2, 1, '2024-08-29 08:36:00', 'Hi Toaddy!');

END //

DELIMITER ;

CALL createAllTables();

CALL insertTestData();

# References:

**Lawn Mower Game Image:**

Vecteezy. (2024) Lawn mowing vectors. *Vecteezy*.

<https://www.vecteezy.com/free-vector/lawn-mowing>