# MILESTONE ONE

## GAME DESCRIPTION

### 1.1 Base Idea

The idea for my 2D multiplayer real-time competitive game is online **Treasure Hunt Adventure**. This game is a one turn-based point-and-click game. The player will battle against another player or an NPC. Each game lasts a maximum of 5 mins. If the time runs out, the player with more marks, or the first player who reach 20 marks first will be the victor.

### 1.2 LOGGING IN

When first opened user will see a login interface. User will be required to enter their username. If this username already exists in database, the user will be prompted for their password. Users have 5 attempts to enter password before account locked. If the username was not found in the database, user will be asked if they would like to register a new account. When finish registration, user will be required to enter email address and password.

### GAME LOPPY

图形用户界面, 应用程序

描述已自动生成

### 1.4 GAME PLAY

When user start game, they will see two panel. The left one is current player’s panel. The right one is rivals. These panels are the “game area”, player will execute game operation on the panel. At the same time, player can communicate with other player through text chat.

### 1.4.1 ONE ON ONE GAMES

图片包含 图表

描述已自动生成

### 1.4.2 TWO ON TWO GAMES

### 1.4.3 OBJECTIVE AND SCORING

The goal of the game is simple, collect all the gems you can find to get more points! Or as many as you can.

Each Jewel will get 10 point, each bomb will lose 20 point.

### 1.5 ADMINISTRATOR

Some players in this game will be given administrator privileges, these players can do something different with normal players. Such as kill running game, create new player, edit details of exist players, delete user’s account, lock or unlock account, or give other user administrator privilege.

**Rules:**

1. User must click the neighboring tiles to move.
2. When the leaving user back to the game, they will return on the tile that is currently empty. If the tile already not empty, user have to choose another tile to continue the game.

## STORYBOARDS

|  |
| --- |
| STORYBOARDS 1 – REGISTRATION SCREEN |
| 1.1  1.2  1.3  1.4  1.5 |
| 1. Textbox for user to enter username 2. Textbox for user to enter email address 3. Textbox for user to enter and create a password 4. Textbox for user to enter re-enter password 5. Button for user to complete registration upon complete, move to storyboard 2 |
| **User Story:** As a user, I want to customize information and create account for playing game. |
| **Indication:**   1. Verify all required fields, if any field is empty, system will return prompt to user. User will be required to enter all fields 2. Verify that the email is valid. If there are any errors, user will be required to enter valid email. 3. Verify that the password in Password and Re-enter Password is the same. User will be required enter same password when any error occurs. 4. User clicks the Register button when entered all required field, system will try to create a new account and redirect user to storyboard 2. If the email already in the database, the prompt will be appeared and redirect user to storyboard 2 |
| STORYBOARDS 2 – LOGIN SCREEN |
| 2.3  2.2  2.1 |
| 1. Textbox for emaill address to be entered. 2. Textbox for password 3. Login button, if emaill address and password are valid, move to storyboard 9. If email is invalid, move to storyboard 3. If password is invalid , move to storyboard 4. If enter invalid password five times, the account will be locked (Move to storyboard 5) |
| **User Story:** As a user, I want to log in my account to play game. |
| **Indication:**   1. Verify all required fields, if any field is empty, system will return prompt to user. User will be required to enter all fields 2. Verify that the email is valid. If there are any errors, user will be required to enter valid email. 3. Verify that the password is valid. System will check the password in the database, if any errors, user will be required to re-enter the valid password. (Move to storyboard 4) 4. User clicks the Continue button to log in account. System redirect user to storyboard 6. If the email doesn’t exist, user will be redirected to storyboard 3. |
| STORYBOARDS 3 – ACCOUNT NOT FOUND |
| 3.22222222  3.1  3.3 |
| 1. Prompt pops up if email is not found in database 2. If Yes, move to storyboard 1 3. If No, move to storyboard 2 |
| **User Story:** As a user, I want to be redirected to register interface when I enter invalid email on login screen. |
| **Indication:**   1. When user enter invalid email, the prompt will be displayed. If click Yes, user will be redirected to storyboard 1. If click No, user will stay on login screen. |
| STORYBOARDS 4 – INVALID PASSWORD |
| 4.2  4.1 |
| 1. Warning prompt pops up if entered invalid password 2. Confirm button for user to close this prompt |
| **User Story:**  As a user, I want a way to remind them when I enter invalid password. |
| **Indication:**   1. When user enter the valid email, but enter invalid password, the prompt will be appeared. User can Click Confirm to close this prompt. |
| STORYBOARDS 5 – ACCOUNT LOCKED |
| 5.2  5.1 |
| 1. Warning prompt pops up if entered invalid password 5 times 2. Click confirm button to close this prompt |
| **User Story:**  As a user, I want a way to protect my account. |
| **Indication:**   1. When user enter the valid email, but enter invalid password 5 times, the prompt will be appeared, and this account will be locked. |
| STORYBOARDS 6 – ADMINISTRATOR INTERFACE |
| 6.2  6.1  6.6  6.5  6.4  6.3 |
| 1. List box for displaying current game 2. List box for displaying all registered players 3. Button for administrator to kill current existing game (Move to storyboard 17) 4. Button for administrator to add new player (Move to storyboard 7) 5. Button for administrator to edit existing player (Move to storyboard 7) 6. Button for administrator to delete existing player (Move to storyboard 8) |
| **User Story:**  As a administrator, I want a specific interface to manage game and user. |
| **Indication:**   1. Admin select the game or player in the groupbox 2. For game, admin can click kill button to kill the existing game 3. For player, admin can choose add, delete, or edit button to manage player |
| STORYBOARDS 7 – ADMIN MANAGE INTERFACE |
| 7.7  7.6  7.4  7.5  7.2  7.3  7.1 |
| 1. Textbox for administrator to add or edit player username 2. Textbox for administrator to add or edit player email address 3. Textbox for administrator to add or edit player password 4. Checkbox to indicate whether the player account is locked or not 5. Checkbox to indicate whether the player account is administrator 6. Button for administrator to delete existing player (Move to storyboard 9) 7. Button for administrator to confirm change (Move to storyboard 6) |
| **User Story:**  As an administrator, I want to have a specific interface to add/edit specific player. |
| **Indication:**   1. Admin can enter required information to add/edit player 2. When all required fields have been entered with valid data, admin can click button to finish their operations |
| STORYBOARDS 8 – DELETE CONFIRMTATION |
| 8.1  8.3  8.2 |
| 1. Warning Prompt pops up 2. Button for administrator to confirm delete player (Move to storyboard 7) 3. Button for administrator to cancel delete player (Move to storyboard 7) |
| **User Story:**  As a admin, I want a confirmation to in case accidental delete |
| **Indication:**   1. When admin click delete player, this prompt will be pops up |
| STORYBOARDS 9 – MAIN GAME LOBBY |
| 9.8  9.7  9.6  9.5  9.4  9.3  9.2  9.1 |
| 1. List box displaying a list of current game and the number of players in the game 2. List box display online players and their highest scores 3. Button for player to delete their own account (Move to storyboard xxx) 4. Button for player to start a new 1v1 Game (Move to storyboard 10) 5. Button for player to start a new 2v2 Game (Move to storyboard 10) 6. Button for player to join the selected currenting game (Move to storyboard 10) 7. Button only for administrator to open admin interface (Move to storyboard 6) 8. Button for player to sign out (Move to storyboard 2) |
| **User Story:**  As a player, I want a game lobby to select and play the game |
| **Indication:**   1. Player can click New 1v1 Game button to create a new game 2. Player can click New 2v2 Game button to create a new game 3. Player can select an existing game in current game and click Join Game button to join this game 4. The player who is administrator can click Admin Console to open the admin interface 5. Player can logout when they end the game |
| STORYBOARDS 10 – ONE VS ONE GAME SCREEN |
| 10.10  10.8  10.9  10.7  10.6  10.5  10.4  10.2  10.3  10.1 |
| 1. The green tile is the “hometile”, it represents the start location of the game 2. Tiles will turn gray it means that player click the “Empty” Tile, player can click the neighbor tile and arrive to the empty tile 3. Tiles will turn red it means that player click the “Jewel” Tile, click this item can get scores 4. Tiles will turn black it means that player click the “Bomb” Tile, when player clicks this tile, their score will be deducted 5. Display the next player to make a move 6. Display the time of game running 7. Display player’s nickname in the game 8. Display the number of Jewel in the player’s inventory 9. Display the number of Bomb in the player’s inventory 10. Display the score the player gets while playing |
| **User Story:**  As a player, I want a game board to play the game |
| **Indication:**   1. When player click the new game button, they will be redirected to game board |
| STORYBOARDS 11 – INVALID LOCATION ERROR |
| 11.2  11.1 |
| 1. The invalid location message prompt pops up 2. Button for user to close the prompt (Move to storyboard 10) |
| **User Story:**  As a player, I want a prompt to remind me when I place on the invalid tile |
| **Indication:**   1. When player places on the invalid tile, this prompt will be displayed |
| STORYBOARDS 12 – JEWEL SCREEN |
| 12.1  12.2 |
| 1. The score message prompt pops up 2. Button for user to close the prompt (Move to storyboard 10) |
| **User Story:**  As a player, I want a prompt to remind when I get Jewel |
| **Indication:**   1. The prompt will appear when player get Jewel |
| STORYBOARDS 13 – BOMB SCREEN |
| 13.2  13.1 |
| 1. The deduction of score message prompt pops up 2. Button for user to close the prompt (Move to storyboard 10) |
| **User Story:**  As a player, I want a prompt to remind when I get Jewel |
| **Indication:**   1. The prompt will appear when player get Jewel |
| STORYBOARDS 14 – WINNING SCREEN |
| 14.2  14.1 |
| 1. The wining message prompt pops up 2. Button for user to close the prompt (Move to storyboard 9) |
| **User Story:**  As a player, I want a prompt to remind when I be the winner |
| **Indication:**   1. When player win the game, this prompt will appear |
| STORYBOARDS 15 – LOST SCREEN |
| 15.2  15.1 |
| 1. The game over message prompt pops up 2. Button for user to close the prompt (Move to storyboard 9) |
| **User Story:**  As a player, I want a prompt to remind when I lost |
| **Indication:**   1. When player lost the game, this prompt will appear |
| STORYBOARDS 16 – PLACING ITEM SCREEN |
| 16.2  16.1  16.3 |
| 1. The item on this tile (3,0) has been picked up (refer to storyboard 10) 2. The item has been placed on the tile (4,0) 3. The number of Jewel in the player’s inventory has updated |
| **User Story:**  As a player, I want a game screen to play the game |
| STORYBOARDS 17 – KILL GAME SCREEN |
| 17.1 |
| 1. Button for user to close this prompt |
| **User Story:** As a admin, I want a prompt when I try to kill the game |

## Entity Relationship Diagram

图示

描述已自动生成

### Entity Relationship Diagram Description

#### Game table

The Game table is used to store instances of a game. Each game has a unique ID, Start time, End time, and State. This table includes relevant information about the game.

**Attributes**:

* **GameID integer(10) (Primary Key)**: This is the unique identifier for each game.
* **StartTime** **date:** This attribute indicates the start time of the game.
* **EndTime** **date:** This attribute indicates the end time of the game.
* **State varchar(32):** This attribute indicates the state of the game, such as running, paused, or over.

**Relationships**:

* **Game and Map**: One-to-one relationship. It means that each game has only one game map.
* **Game and Player**: One-to-many relationship. It means that each game can have many players, and multiple players can join the same game.

#### Game\_Player table

The Game\_Player table is a join table, used to associate many-to-many relationship between Game and Player table. This table stores player detail information in a specific game. Such as Score, the last location.

**Attributes:**

* **GameID integer(10) (Composite Primary Key, Foreign Key) :** This is unique identifier for each game and as a foreign key to link Game Table
* **PlayerID integer(10) (Composite Primary Key, Foreign Key):** This is unique identifier for each player and as a foreign key to link Player Table
* **Score integer(10):** This attribute stores the score the player gets while playing.
* **TileID integer(10) (Foreign Key):** This attribute is a foreign key to link Game\_Player table and Tile table

**Relationships:**

* **Game\_Player and Game:** Many-to-one relationship. It means that each game can allow multiple players to play at a time
* **Game\_Player and Player:** Many-to-one relationship. It means that each player can join multiple game at the same time
* **Game\_Player and Tile:** One-to-one relationship. It means that each player can only stay in one tile at a time.

#### For Player table

This table is used to store plays’ basic information and game state. Each player has the unique key in this table. Such as PlayerID, Name, Account, Password and so on.

**Attributes:**

* **PlayerID(Primary Key):** This is the unique identifier for each player.
* **Name varchat(255):** This attribute stores the nickname of player
* **Email varchat(255):** This attribute stores the player’s email credentials, which are used to log in to the game
* **Password varchat(255):** This attribute stores the credentials of password, ensuring that player can login security
* **LockState** **bit:** This attribute indicates whether the player’s account is locked.
* **LoginState bit:** This attribute indicates whether the player’s account is signed.
* **GameState bit:** This attribute stores the current game state of player.
* **IsAdministrator bit:** This attribute indicates whether the player is administrator

**Relationship:**

* **Player and Game\_Player:** One-to-many relationship. It means that each player can join multiple game at the same time
* **Player and inventory\_Item:** One-to-many relationship. It means that each player can have multiple inventories to store different types of items
* **Player and Chat\_Player:** One-to-many relationship. It means that each player can join multiple chat sessions at a time

#### Map table

This map used to store the information about the game board. Each map has unique identifier. This table defines the size of the game board. This is a vital part of the game, players will play on the map.

**Attributes:**

* **MapID** **integer(10) (Primary Key):** This is the unique identifier for each map.
* **GameID integer(10) (Foreign Key):** This attribute links the Map to a specific Game. It establishes relationship between Map table and Game Table
* **MaxRow integer(10):** This attribute indicates the maximum number of the rows on the map.
* **MaxColumn integer(10):** This attribute indicates the maximum number of the columns on the map.

**Relationships:**

* **Map and Game:** one-to-one relationship. It means that each map only belongs to one game.
* **Map and Tile:** one-to-many relationship. It means that a map can have many tiles. The player will play game on the tile of the map.

#### Tile table

This table used to store the information about the tile. Each tile has x and y coordinates, the occupied state and item state of the tile.

**Attributes:**

* **TileID integer(10) (Primary Key)**: This is
* **MapID integer(10) (Foreign Key):** This attribute links the Tile to a specific Map. It establishes a relationship between Tile table and Map table.
* **Row** **integer(10):** This attribute stores the row number of Tile on the specific map
* **Column** **integer(10):** This attribute stores the column number of Tile on the specific map
* **IsOccupied (bit):** This attribute indicates whether the tile is occupied by player.
* **IsEmpty (bit):** This attribute indicates whether the tile has any item.

**Relationships:**

* **Map and Tile:** one-to-many relationship. It means that each tile belongs to one specific map.
* **Tile and Tile\_Item:** one-to-many relationship. It means that each tile can include one or more items
* **Item and Game\_Player:** One-to-one relationship. It means that each player only place on one single tile at a time

#### Tile\_Item table

The Tile\_Item table is an join table, used to establish many-to-many relationship between Tile and Item table. This table tracks the location of item

**Attributes:**

* **TileID: integer(10) (Primary Key, Foreign Key):** This is the unique identifier for each and as a foreign key to link Tile table.
* **ItemID: integer(10) (Primary Key, Foreign Key):** This is the unique identifier for each item and as a foreign key to link Item table.

**Relationships:**

* **Tile\_Item and Tile:** many-to-one relationship. It means that each single tile can have one or more item at a time
* **Tile\_Item and Item:** one-to-one relationship. It means that each item can only be stored in one single tile at a time

#### Item table

This table is used to store the item information. Such as ID of each item

**Attributes:**

* **ItemID(Primary Key):** This is the unique identifier for each item.
* **ItemTypeID integer(10) (Foreign Key):** This attribute links the item to a specific ItemType. It establishes a relationship between Item table and ItemType table.

**Relationships:**

* **Item and Inventory\_Item:** One-to-one relationship. It means that each Item only can be stored in one inventory
* **Item and ItemType:** Many-to-one relationship. It means that each item has their specific type. Each type can have multiple items
* **Item and Tile\_Item:** One-to-one relationship. It means that each item only can store in a specific tile.

#### Item\_Type table

The Item\_Type table used to store the specific type of item. Each item type defines some common features, such as harm and health.

**Attributes:**

* **ItemTypeID: integer(10) (Primary Key):** This is the unique identifier for each Item Type
* **Type: varchar(255):** This attribute stores the type of item
* **Harm: integer(10):** This attribute stores the numerical value of harm
* **Heath: integer(10):** This attribute stores the numerical value of health

**Relationships:**

* **ItemType and Item:** one-to-many relationship. It means that each Item Type can have one or more items

#### Inventory\_Item table

The Inventory-Item table is a join table, used to establish many-to-many relationship between Player and Item table. This table tracks player’s inventory and number of each type of item in the inventory

**Attributes:**

* **PlayerID integer(10) (Primary Key, Foreign Key):** This is unique identifier for each player and as a foreign key to link Player table.
* **ItemID integer(10) (Primary Key, Foreign Key):** This is unique identifier for each item and as a foreign key to link Item table.
* **Quantity integer(10):** This attribute stores the number of item in the player’s inventory

**Relationships:**

* **Inventory\_Item and Player:** Many-to-one relationship. It means that each player can have one or more inventories
* **Inventory\_Item and Item:** One-to-one relationship. It means that each item can only be stored corresponding inventory

#### Chat\_Session table

The Chat\_Session table used to store the unique identifier of each chat session.

**Attributes:**

* **ChatID integer(10) (Primary Key):** This is unique identifier for each chat.

**Relationships:**

* **Chat\_Session and Chat\_Player:** One-to-many relationship. It means that each chat can be seen by multiple players

#### Chat\_Player table

The Chat\_Player table is a associated table used to associate Player and Chat\_Session table and establish many-to-many relationship between Player and Chat\_Session table. This table used to store message content and messages sent time. This table also link player to specific chat session.

**Attributes:**

* **ChatID integer(10) (Primary Key, Foreign Key):** This is unique identifier for each chat and as a foreign key link the Chat\_Session table
* **PlayerID integer(10) (Primary Key, Foreign Key):** This is unique identifier for each chat and as a foreign key link the Player table
* **MessageContent varchar(255):** This attribute stores the message content of chat.
* **TimeStamp timestamp:** This attribute stores the time when message is sent.

**Relationships:**

* **Chat\_Player and Player:** Many-to-one relationship. it means that multiple records in Chat\_Player table can be associated with a record in the Player table
* **Chat\_Player and Chat\_Session:** Many-to-one relationship. It means that multiple records in the Chat\_Player table can be associated with a record in the Chat\_Session table

## SQL

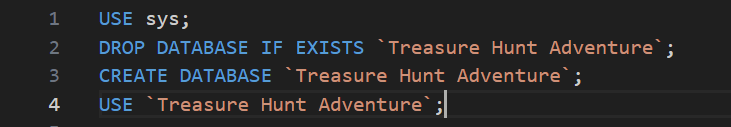
**Description:**

This script creates a ‘Treasure Hunt Adventure’ database consisting of 11 tables. This database is designed to support the game that can be played 1v1 or multiple players, and the database is a basic structure for support game features and player interact with the game. Such as registration, login, manage account, player movement, collect item, and chat with each player real-time.

**Explanation:**

**DATABASE SETUP**

For this part, I wrote 'USE sys' statement to specify database that named ‘sys’. Then I drop the ‘TreasureHuntAdventure’, this script is for avoid errors occurred by the database already exists on the computer. Afterward, this script creates a new database ‘Treasure Hunt Adventure’ and select this database as target database to prepare for subsequent operations.



**TSQL PROCEDURE SETUP**

First, I use ‘DELIMITER //’ to change the statement terminator from default ‘;’ to //. This is to prevent interrupting code execution within procedure. The ‘BEGIN’ indicates the start of ‘CreateTables’. In the ‘CreateTables’ procedure, I create 11 necessary tables and their relationships to support my game. All based on my logical ERD. For each table that includes table name, attribute, data type, primary key and foreign key. In the next part of the procedure includes insert statements. I used fake data generate tool (<https://www.mockaroo.com/>) to create some test data and inserted them to test my database. Finally, I use ‘END //’ to end the script that in the procedure and use ‘UNLIMITER ;’ to change statement terminator from ‘//’ to ‘;’, then I used ‘CALL CreateTables();’ to call this procedure.

## CRUD Table

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Entity/Attribute** | **Player login, including lock out** | **Player registration** | **Laying out tiles on a game board** | **Placing an item on a tile** | **Player game play movement** | **Game play scoring** | **Player game play acquiring inventory** | **Move an Item (NPC effect)** | **Kill running games** | **Add new players** | **Update data of a player** | **Delete a player** |
| **Player** | **RU** | **C** |  |  | **R** | **R** | **R** |  |  | **C** | **RU** | **D** |
| **PlayerID** | **R** | **C** |  |  | **R** | **R** | **R** |  |  | **C** | **R** | **D** |
| **Name** |  | **C** |  |  |  |  |  |  |  | **C** | **RU** | **D** |
| **Email** | **R** | **C** |  |  |  |  |  |  |  | **C** | **RU** | **D** |
| **Password** | **R** | **C** |  |  |  |  |  |  |  | **C** | **RU** | **D** |
| **LockState** | **R** | **C** |  |  | **R** | **R** | **R** |  |  | **C** | **RU** | **D** |
| **LoginState** | **U** | **C** |  |  | **R** | **R** | **R** |  |  | **C** | **RU** | **D** |
| **GameState** |  | **C** |  |  | **R** | **R** | **R** |  |  | **C** | **RU** | **D** |
| **IsAdministrator** |  | **C** |  |  |  |  |  |  |  | **C** | **RU** | **D** |
| **Game** |  |  | **R** |  | **R** | **R** | **R** | **R** | **RU** |  |  |  |
| **GameID** |  |  | **R** |  | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **StartTime** |  |  |  |  |  |  |  |  |  |  |  |  |
| **EndTime** |  |  |  |  |  |  |  |  | **U** |  |  |  |
| **State** |  |  | **R** |  | **R** | **R** | **R** | **R** | **U** |  |  |  |
| **Map** |  |  | **R** |  | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **MapID** |  |  | **R** |  | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **GameID** |  |  | **R** |  | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **MaxRow** |  |  | **R** |  |  |  |  | **R** |  |  |  |  |
| **MaxColumn** |  |  | **R** |  |  |  |  | **R** |  |  |  |  |
| **Tile** |  |  | **CR** | **RU** | **R** | **R** | **R** | **RU** | **R** |  |  |  |
| **TileID** |  |  | **C** | **R** | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **MapID** |  |  | **R** | **R** | **R** | **R** | **R** | **R** | **R** |  |  |  |
| **Row** |  |  | **C** | **R** | **R** | **R** | **R** | **R** |  |  |  |  |
| **Column** |  |  | **C** | **R** | **R** | **R** | **R** | **R** |  |  |  |  |
| **IsOccupied** |  |  | **C** | **R** | **RU** | **RU** | **RU** | **R** |  |  |  |  |
| **IsEmpty** |  |  | **C** | **U** | **RU** | **RU** | **RU** | **U** |  |  |  |  |
| **Item** |  |  |  | **R** |  | **R** | **R** | **R** |  |  |  |  |
| **ItemID** |  |  |  | **R** |  | **R** | **R** | **R** |  |  |  |  |
| **ItemTypeID** |  |  |  | **R** |  | **R** | **R** | **R** |  |  |  |  |
| **ItemType** |  |  |  |  |  | **R** | **R** | **R** |  |  |  |  |
| **ItemTypeID** |  |  |  |  |  | **R** | **R** | **R** |  |  |  |  |
| **Type** |  |  |  |  |  | **R** | **R** | **R** |  |  |  |  |
| **Harm** |  |  |  |  |  | **R** | **R** | **R** |  |  |  |  |
| **Heath** |  |  |  |  |  | **R** | **R** | **R** |  |  |  |  |
| **Tile\_Item** |  |  |  | **RU** |  |  |  | **RU** |  |  |  |  |
| **TileID** |  |  |  | **U** |  |  |  | **U** |  |  |  |  |
| **ItemID** |  |  |  | **R** |  |  |  | **R** |  |  |  |  |
| **Game\_Player** |  |  |  |  | **RU** | **RU** |  |  |  |  |  |  |
| **GameID** |  |  |  |  | **R** | **R** |  |  |  |  |  |  |
| **PlayerID** |  |  |  |  | **R** | **R** |  |  |  |  |  |  |
| **TileID** |  |  |  |  | **RU** | **R** |  |  |  |  |  |  |
| **Score** |  |  |  |  |  | **U** |  |  |  |  |  |  |
| **Inventory\_Item** |  |  |  |  |  |  | **RU** |  |  |  |  |  |
| **PlayerID** |  |  |  |  |  |  | **R** |  |  |  |  |  |
| **ItemID** |  |  |  |  |  |  | **R** |  |  |  |  |  |
| **Quantity** |  |  |  |  |  |  | **U** |  |  |  |  |  |
| **ChatSession** |  |  |  |  |  |  |  |  |  |  |  |  |
| **ChatID** |  |  |  |  |  |  |  |  |  |  |  |  |
| **Chat\_Player** |  |  |  |  |  |  |  |  |  |  |  |  |
| **ChatID** |  |  |  |  |  |  |  |  |  |  |  |  |
| **PlayerID** |  |  |  |  |  |  |  |  |  |  |  |  |
| **MessageContent** |  |  |  |  |  |  |  |  |  |  |  |  |
| **TimeStamp** |  |  |  |  |  |  |  |  |  |  |  |  |

### Description

1. **Player login, including lock out**

Before a player logs in, the system first reads the player's credential information and account status from the Player table, such as PlayerID, Email, Password, and LockState. If the login is successful, the system will update the player's LoginState to '1'. After five incorrect password inputs, the system will update LockState to '1'

1. **Player registration**

When a player registers an account, the system creates a new record in the Player table, including PlayerID, Username, Email, Password, LockState, LoginState, GameState, IsAdministrator. The default values ​​of the LockState, LoginState, GameState, IsAdministrator fields are 0

1. **Laying out tiles on a game board**

At the beginning of the game, the system will create Tiles according to the maximum Row and Column specified in the 'Map' table. The information of each Tile is recorded in the Tile table, and the IsOccupied record of the Tile table is '0' in the initial state

1. **Placing an item on a tile**

When a player or NPC places an item on a Tile on the map, the system creates a new record in the Tile\_Item table, associates the Tile with the item ItemID and updates the 'IsEmpty' record in the Tile table to 1, indicating that the Tile now contains an item.

1. **Player game play movement**

When the player moves to another legal Tile, the system first checks the 'IsOccupied' record in 'Tile'. Only when the record is '0' can the player successfully move to the Tile. When the player moves, the system updates the 'TileID' record in the 'Game\_Player' table to indicate that the player has moved successfully.

1. **Game play scoring**

When a player picks up a Jewel, they will get the corresponding score. The system will first query the Type, Harm, Heath records given in the 'Item' and 'ItemType' tables. When the player picks up the item and puts it in the inventory, the system will change the 'Score' value in the 'Game\_Player' table to indicate that the player has successfully obtained the score.

1. **Player game play acquiring inventory**

When a player picks up an item from a Tile on the map, the system adds a new record to the 'Inventory\_Item' table to indicate that the item has been successfully stored in the inventory. At the same time, the system modifies the 'IsEmpty' record in the 'Tile' table to indicate that the item has been picked up and that the Tile has no items (note that this only applies if the Tile has one and only one Item).

1. **Move an Item (NPC effect)**

When the game's NPC moves an item, the system reads and modifies the 'Tile-Item' table, indicating that the item is moved from one Tile to another. At the same time, the IsEmpty attribute of the original 'Tile' table that originally held the item is modified to '1', and the 'isEmpty' attribute of the 'Tile' table that currently holds the item is modified to 0

1. **Kill running games**

When the administrator terminates a running game, the system will update the State field in the Game table to 'Over', update the game's 'EndTime', and finally delete all information about the game to free up space and optimize database performance.

1. **Add new players**

When an administrator adds a new player, the system will add a new record to the 'Player' table, including PlayerID, Username, Email, Password, LockState, LoginState, GameState, IsAdministrator.

1. **Update data of a player**

When the administrator updates the player information, the system will first update the segments in the Player table, such as Email, Password, GameState and other basic information.

1. **Delete a player**

When a player or administrator deletes a player, the system will delete all records related to the player from the Player table, including account information and game data. The record deletion operation is irreversible, so there will be a prompt box for further confirmation.