Battleships database details

|  |  |
| --- | --- |
| **Table name** | **Description** |
| users | * Stores user data (username, wins, losses, isAI) |
| lobbies | * lists open and ongoing games * stores current board state, user id‘s, lobby name, short description * only way of back-end <—> front-end communication * link to *history* table |
| history | * stores starting board state, player id‘s, winner id * link to *moves* table |
| moves | * stores player movement details |

Details of table variables

B – back-end application

F – front-end application

def – default definition of variable

NULL? – can the variable be NULL?

R – allowed to read from; W – allowed to write to.

Non – None, empty, ,,“.

user – back-end AI controlled application, or a front end user written AI (can be human).

opponent – the user who joined a lobby.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **users** | | | | | | |
| Name | Type | NULL? | def | B | F | Comments |
| userID | int | No | auto | R | R | Primary key of user table, auto increments. |
| username | char(45) | No | Non | RW | RW | Username of a user, can be renamed. |
| wins | int | No | 0 | RW | R | Win count, incremented by back-end. |
| losses | int | No | 0 | RW | R | Loose count, incremented by back-end. |
| is\_ai | tinyint | No | 0 | RW | R | Tells if user is controlled by back-end. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **history** | | | | | | |
| Name | Type | NULL? | def | B | F | Comments |
| gameID | int | No | Non | R | R | Primary key of lobbies table, auto increments. |
| game\_name | char(45) | No | Non | RW | R | Name of the lobby, set by B when starting game |
| player1\_ID | int | No | Non | RW | R | ID of the lobby creator, set during start of game by B, linked to user table. |
| player2\_ID | int | No | Non | RW | R | ID of the lobby creator, set during start of game by B, linked to user table. |
| map1 | char(100) | No | Non | RW | RW | starting map of player1, set at the start of the game by B. |
| map2 | char(100) | No | Non | RW | R | starting map of player2, set at the start of the game by B. |
| winnerID | int | No | -1 | RW | R | userID who won the game, set at the end of the game by B. -1 if none. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **moves** | | | | | | |
| Name | Type | NULL? | def | B | F | Comments |
| gameID | int | No | Non | R | R | Primary key of move table, set by B (equal to history.gameID), linked to history table. |
| moveID | int | No | Non | RW | RW | Primary key of move table, turn number set by B, incremented manually. linked to history table. |
| move\_pos | char(5) | No | Non | RW | R | set by B, equal to lobbies.user\_input |
| move\_res | char(5) | No | Non | RW | R | set by B, equal to lobbies.console\_output |
| playerID | int | No | Non | RW | R | ID of the turn maker, set by B, linked to user table. |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **lobbies** | | | | | | |
| Name | Type | NULL? | def | B | F | Comments |
| lobbyID | int | No | auto | R | R | Primary key of lobbies table, auto increments. |
| lobby\_name | char(45) | No | ,,unnamed  lobby“ | RW | RW | Name of the lobby, set when creating lobby. |
| description | tinytext | No | Non | RW | RW | Short lobby description, can be left empty |
| lobby\_status | char(1) | No | ,,w“ | RW | RW | definitions:   * ,,w“ – waiting for other user, set during init. * ,,r“ – both users ready, waiting for server input, set by the opponent. * ,,i“-game is running, set by the server. |
| adminID | int | No | Non | RW | RW | ID of the lobby creator, set during lobby creation, linked to user table. |
| opponentID | int | Yes | NULL | R | RW | ID of the opponent, set when joining lobby, linked to user table. |
| user\_input | char(5) | No | Non | R | W | written by users when game\_status is ,,w“ and current\_player is their ID. ex.: ,,1-5“ ; ,,10-10“. |
| console\_output | char(5) | No | Non | W | R | Console writes the result of a user‘s action. ex.: ,,0“ - miss; ,,1“ - hit ; ,,2-1“- sunk a ship of type 1. |
| admin\_map | char(100) | Yes | NULL | RW | R | current map of the lobby creator , changed during gameplay by the B, can be viewed by spectators. |
| opponent\_map | char(100) | Yes | NULL | RW | R | current map of the opponent , changed during gameplay by the B, can be viewed by spectators. |
| historyID | int | Yes | NULL | RW | R | link to history table, created by console when starting the game. |
| game\_status | char(1) | No | ,,n“ | RW | RW | allows flawless communication between F and B. Definitions:   * ,,n“ – waiting for game to start * ,,c“ – waiting for console input * ,,w“ – waiting for user input * ,,f“ – game is finished |
| current\_player | int | No | -1 | RW | R | Shows the user‘s id, who is now doing his turn. |

Ways of communication, function requirements: (bold are different functions which will be neatly packaged in a separate class file)

**Front-end:**

* **Registration**, create a new user with username — INSERT INTO users(username,is\_ai) VALUES(<<username>>,0)
* **Login**, get id of user from username — SELECT userID FROM users WHERE username=<<username>>

**After registration, login allow:**

* **Change name of user** — UPDATE users SET username=<<username>> WHERE userID=<<id>>
* **Get info (wins, losses) about user** — SELECT wins,losses FROM users WHERE userID=<<id>>
* **Create lobby** with a lobby name, optional description, (if not already in a lobby, or spectating) – INSERT INTO lobbies(lobby\_name,description,adminID) VALUES(<<lobbyName>>,<<optionaldescription>>,<<userID>>)
* **List lobbies as a player** — SELECT lobbyID, lobby\_name, description FROM lobbies WHERE lobby\_status=‘w‘
* **List lobbies as a spectator** — SELECT lobbyID, lobby\_name, description FROM lobbies
* **Join a lobby as a player** (if not in a lobby or spectating) select and save lobbyID then and do this once— UPDATE lobbies SET opponent\_id=<<userID>>, lobby\_status=r WHERE lobbyID=<<lobbyID>>
* **Join a lobby as a spectator —** — get usernames of players, maps, history.
* **Read gamestatus and console output during gameplay** in a while cycle until your turn (or when spectating) — SELECT game\_status, console\_output, current\_player FROM lobbies WHERE lobbyID=<<lobbyID>>
* **Get current Map state —** SELECT admin\_map, opponent\_map FROM lobbies WHERE lobbyID=<<lobbyID>>
* **Write your movement** when current\_player==playerID, game\_status==w — UPDATE lobbies SET user\_input=<<YourMove>>, game\_status=‘c‘ WHERE lobbyID=<<lobbyID>>
* **Leave lobby before the game start** — UPDATE lobbies SET opponent\_id=NULL, lobby\_status=w WHERE lobbyID=<<lobbyID>>
* **Gameplay is declared finished when** game\_status==,,f“ and current\_player==<<userID>>, acknowledge the end of the game and get the link to the history of the game — SELECT historyID FROM lobbies WHERE lobbyID=<<lobbyID>> **THEN** UPDATE lobbies SET user\_input=,,OK“, game\_status=‘c‘ WHERE lobbyID=<<lobbyID>>
* **Get the winner of the game —** SELECT username FROM users WHERE userID= SELECT winnerID FROM history WHERE gameID = <<historyID>>
* **list all of finished games** — SELECT gameID, game\_name, player1\_ID, player2\_ID FROM history WHERE winnerID!=-1
* **Get info about a particular game —** SELECT game\_name, player1\_ID, player2\_ID, map1, map2 FROM history WHERE gameID=<<gameID>> **THEN GET MOVE LIST** SELECT moveID, move\_pos, move\_res, playerID FROM moves WHERE gameID=<<gameID>>

**Back-end:**

* **Create users for AI applications** (if they don‘t have one) — INSERT INTO users(username,is\_ai) VALUES(<<username>>,1)
* **Create lobbies for AI applications** — INSERT INTO lobbies(lobby\_name,description,adminID) VALUES(<<lobbyName>>,<<optionaldescription>>,<<userID>>)
* **Listen to lobby status —** SELECT lobbyID FROM lobbies WHERE lobby\_status=‘r‘
* When a lobby is ready put it in a list of ongoing lobbies, set lobby\_status to ‚i‘, create a new history table entry and start managing it (choose who goes first, react to user inputs, save player moves in the moves table with specific gameID)
* **Create new game history entry** — INSERT INTO history(game\_name,player1\_ID,player2\_ID) VALUES(<<game\_name>>,<<player1\_ID>>,<<player2\_ID>>)
* **Read user inputs —** SELECT lobbyID, user\_input FROM lobbies WHERE game\_status=‘c‘
* **Save them to history —** INSERT INTO moves(gameID,moveID,move\_pos,move\_res,playerID) VALUES (<<gameID>>,<<TurnCount+1>>,<<user\_input>>,<<console\_output>>,<<current\_player>>)
* **Respond to user inputs —** UPDATE lobbies SET user\_input=,,“, game\_status=‘w‘, console\_output=<<consoleOutput>>, current\_player=<<currentPlayerID>>, admin\_map=<<updatedAdminMap>>, opponent\_map=<<updatedOpponentMap>>, WHERE lobbyID=<<lobbyID>>
* **If game is finished** inform players, then delete lobby**—** UPDATE lobbies SET user\_input=,,“, game\_status=‘f‘, console\_output=‘‘, current\_player=<<currentPlayerID>> WHERE lobbyID=<<lobbyID>>
* **Delete lobby after users left —** DELETE FROMlobbies WHERE lobbyID=<<lobbyID>>