Finalizing game statistics and Rule-based intelligence
Team 23

Authors - Tymur Slesarenko, Aman Singh, Willem Schillemans

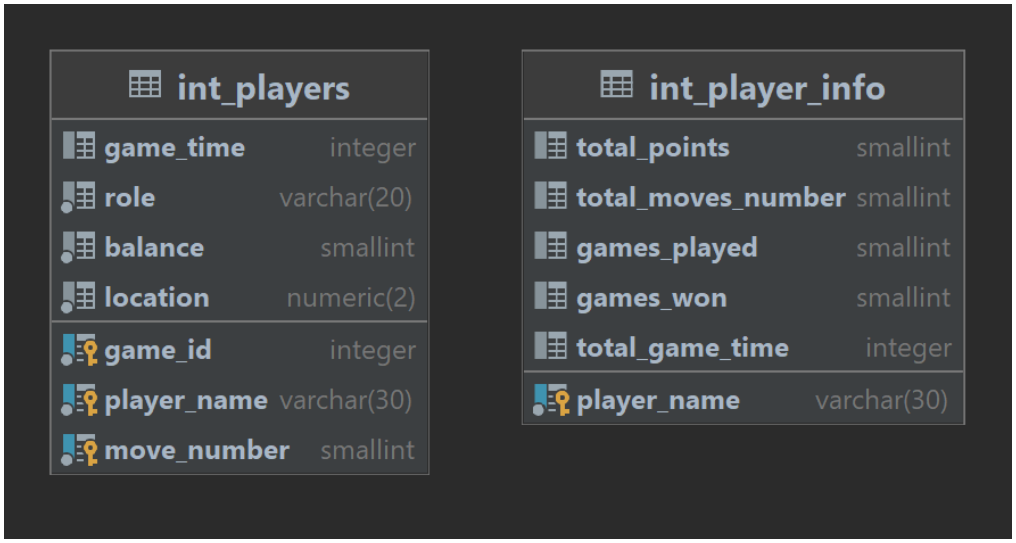
Rubric Line	Score	Improvements
Game statistics		
Creating the tables and ERD (DDL)	Meets	Game database has been dramatically simplified (from 7 to 2 tables).
Game logic and handling Data (DML)	NotMeets	It is possible to retrieve data from the database to the program. Data is saved correctly, without any duplicates or redundancies.
Processing game data	Invalid	The game statistics screen shows the requested data in text format. The data analysis is performed on the data. The leaderboard has been added.
Visual data representation	Invalid	The Model View Presenter design pattern has been respected. The optional requirements of an additional screen with statistics on all played games completely respect the MVP design pattern.
------Rule Base Intelligence--------		
Rule base:	Meets	More relevant rules have been derived.
MVP:	Invalid	The Model View Presenter design pattern has been respected.
This criterion is linked to a learning outcome Expert rules:	Invalid	Sufficient ‘expert rules’ were implemented to create credible intelligence and only effective 'expert rules' were used.

Game statistics

Database details :

Database name	game
DBUserName	game
Database password	7sur7
Database port number	5432
JDBC driver	postgresql-42.5.1.jar

Entity Relationship Diagram:



Primary keys, check constrains:

```
         List of relations
 Schema |    Name     | Type  | Owner
 player_name | character varying(30) |          | not null |
 move_number | smallint              |          | not null |
 game_time   | integer               |          | not null |
 role        | character varying(20) |          | not null |
 balance     | smallint              |          | not null |
 location    | numeric(2,0)          |          | not null |
Indexes:
    "int_players_pkey" PRIMARY KEY, btree (game_id, player_name, move_number, game_time)
Check constraints:
    "check_player_name_length" CHECK (length(player_name::text) >= 2)
```

```
                    Table "public.int_player_info"
       Column       |         Type          | Collation | Nullable | Default
--------------------+-----------------------+-----------+----------+---------
 player_name        | character varying(30) |           | not null |
 total_points       | smallint              |           |          |
 total_moves_number | smallint              |           |          |
 games_played       | smallint              |           |          |
 games_won          | smallint              |           |          |
 total_game_time    | integer               |           |          |
Indexes:
    "player_info_pkey" PRIMARY KEY, btree (player_name)
```

Storing data in the database:

Storing data if playing with the computer (INT_PLAYERS):

	game_id	player_name	move_number	balance	location	game_time	role
1	12861	Tymur	1	1860	11	11	MONOPOLIST
2	12861	Tymur	2	1700	14	21	MONOPOLIST
3	12861	Tymur	3	1460	24	29	MONOPOLIST
4	12861	Tymur	4	1460	10	38	MONOPOLIST
5	12861	Tymur	5	1260	15	51	MONOPOLIST
6	12861	Tymur	6	1040	23	59	MONOPOLIST
7	12861	Tymur	7	760	29	67	MONOPOLIST
8	12861	Tymur	8	410	37	80	MONOPOLIST
9	12861	Tymur	9	550	3	88	MONOPOLIST
10	12861	Tymur	10	550	14	102	MONOPOLIST
11	12861	Tymur	11	330	21	115	MONOPOLIST
12	12861	Tymur	12	70	27	122	MONOPOLIST
13	12861	Tymur	13	-250	34	127	MONOPOLIST

Storing data for local multiplayer (INT_PLAYERS):

	game_id	player_name	move_nu... 1	balance	location	game_time	role
1	25132	Aman	1	2000	0	4	COMPETITOR
2	25132	Tymur	1	2000	7	3	MONOPOLIST
3	25132	Aman	2	2000	29	14	COMPETITOR
4	25132	Aman	3	1800	35	23	COMPETITOR
5	25132	Tymur	3	1860	13	17	MONOPOLIST
6	25132	Tymur	4	1860	20	27	MONOPOLIST
7	25132	Aman	4	2050	7	30	COMPETITOR
8	25132	Aman	5	1910	11	33	COMPETITOR
9	25132	Tymur	6	1860	10	37	MONOPOLIST
10	25132	Aman	6	1810	17	41	COMPETITOR
11	25132	Aman	7	1810	20	47	COMPETITOR
12	25132	Aman	8	1530	29	54	COMPETITOR
13	25132	Tymur	8	1680	16	50	MONOPOLIST
14	25132	Tymur	9	1460	21	58	MONOPOLIST
15	25132	Aman	9	1530	35	63	COMPETITOR
16	25132	Aman	10	1330	38	74	COMPETITOR
17	25132	Tymur	10	1240	23	65	MONOPOLIST
18	25132	Tymur	10	940	32	69	MONOPOLIST
19	25132	Aman	11	1380	7	81	COMPETITOR
20	25132	Tymur	11	1080	1	76	MONOPOLIST
21	25132	Aman	12	1394	7	85	COMPETITOR
22	25132	Tymur	12	1066	11	84	MONOPOLIST
23	25132	Aman	13	1194	15	88	COMPETITOR
24	25132	Aman	14	994	25	95	COMPETITOR
25	25132	Tymur	14	1066	16	92	MONOPOLIST

During gameplay, data is stored for each move to maintain a comprehensive record of game progress and player actions. The stored data includes the game ID, player name, move number, balance, location on the board, role, and game time in seconds.
Duplicate data entries are avoided to maintain data integrity. For example, if a player is in jail and skips turns, redundant entries for those skipped turns are not added.
The stored attributes for each move are:

- Game ID: Unique identifier for the game session.
- Player Name: Name or identifier of the player.
- Move Number: Sequential number indicating the move's order.
- Balance: Current player's financial status after the move.
- Location on the Board: Position or tile where the move occurred.
- Role: Assigned role in the game.
- Game Time in Seconds: Duration of the game at the move.

By excluding the storage of computer data, the system optimizes storage resources and avoids redundancy. This design choice ensures that the recorded data focuses on the player's performance and allows for accurate analysis, statistical calculations, and generation of player statistics.

Storing players' data (INT_PLAYER_INFO):

	player_name	total_points ▲ 1	total_moves_number	games_played	games_won	total_game_time
1	drtytrd	229	3	1	0	21
2	wq	377	1	1	1	35
3	Sasha	405	8	1	0	14
4	new3	430	20	2	1	39
5	TEST1	463	371	7	3	1531
6	Max	515	8	1	1	14
7	New player 1	599	20	1	1	53
8	asdfgh	625	0	0	0	0
9	mikasa	711	188	5	3	1172
10	Willem	728	10	1	0	53
11	usa	825	0	0	0	0
12	Aman	844	590	41	18	4370
13	Tymur	886	3187	278	29	29655
14	qwert	998	0	0	0	0
15	AAAAA	1000	0	0	0	0
16	dtfklm	1000	1	1	0	4
17	PCA	1000	0	0	0	0
18	ytjt	1000	18	1	0	133
19	ewfjkn	1031	1	1	1	4
20	vietnam	1135	0	0	0	0
21	new1	1255	6	1	0	9
22	TEST2	1350	78	9	5	240
23	New player 2	1525	20	1	0	53
24	Test2	1537	93	8	3	531
25	Test1	1629	84	7	4	488
26	TESTTT	1648	104	2	0	442
27	new4	1746	20	2	1	39

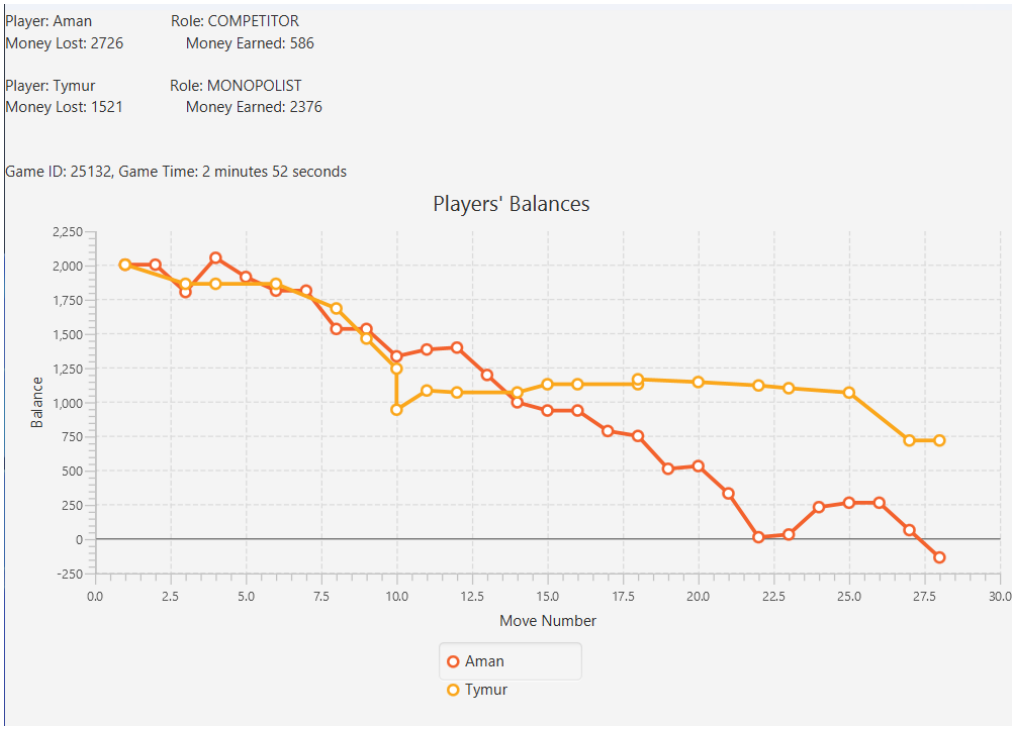
Game statistics:

At the end of each game, players are presented with a comprehensive set of game statistics that provide insights into their performance and progress. This feature aims to enhance the gameplay experience by offering players a summary of their achievements and a graphical representation of their balance throughout the game.

The game statistics screen includes the following information:

1. Money Earned and Lost: Players can view the total amount of money they have earned and lost during the game. This data helps them evaluate their financial success and provides a clear understanding of their monetary gains and losses.
2. Role: The player's assigned role or character in the game is displayed. This information reminds players of their chosen role and adds a personal touch to the game summary.
3. Game ID: A unique identifier assigned to each game session is shown. This ID helps players distinguish between different game sessions and can be used for reference or analysis purposes.
4. Game Duration: The duration of the game is presented to players, indicating the total time elapsed during the gameplay. This information allows players to assess the length of their gaming experience and compare it with previous sessions.
5. Balance Graph: A visual representation of the player's balance throughout the game is displayed in the form of a graph. This graph illustrates the fluctuation of the player's financial status over time, providing valuable insights into their decision-making and strategic choices.

Example of the Game Statistics Screen:



Within the game's main menu, players have access to two additional features: Statistics and Leaderboard. These features provide players with an overview of their personal game performance as well as a glimpse into the achievements of other players.

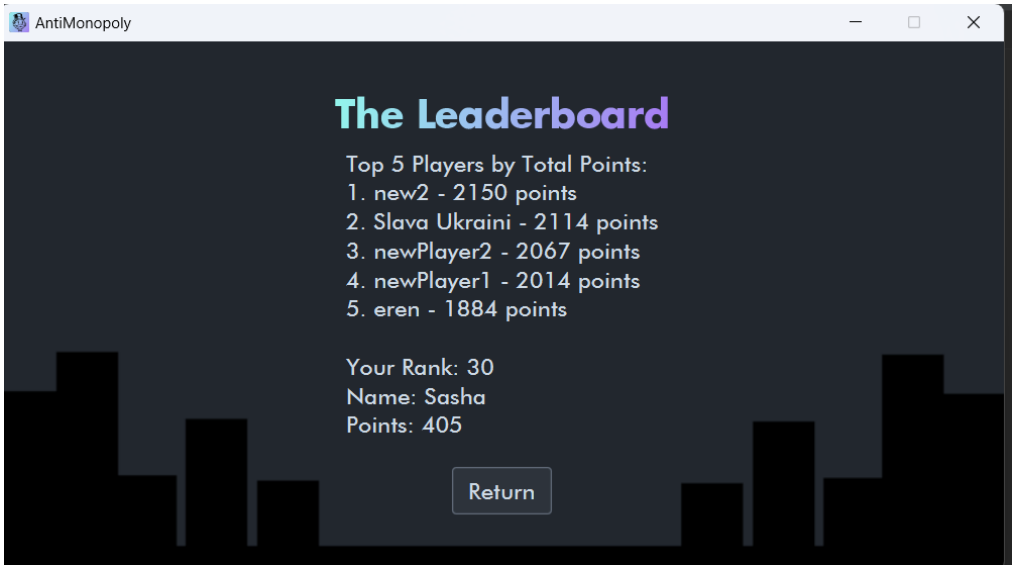
Player Statistics:

- Upon clicking the "Statistics" button, players are prompted to enter their name.
- If the entered name exists in the system, players can view their personal statistics, including:
    - Score: The player's overall score or total points earned in the game.
    - Total Moves Number: The number of moves the player has made throughout all games played.
    - Total Games Played: The total number of games the player has participated in.
    - Total Games Won: The number of games the player has won.
    - Total Game Time: The cumulative duration of all games played by the player, measured in seconds.
- Additionally, players can access average statistics for other players, providing a benchmark for comparison.
- This feature enables players to track their progress, assess their performance, and gain insights into their gaming habits and achievements.

Leaderboard:

- By selecting the "Leaderboard" button, players are presented with a list of the top five players ranked by their scores.
- Players can easily identify their own position on the leaderboard and view their corresponding score.

- The leaderboard showcases the highest-scoring players, fostering a sense of competition and providing a goal for players to strive for.
- This feature allows players to gauge their performance relative to others and serves as a source of motivation to improve their rankings.



The score in the game is calculated using the Elo system, which is a rating system commonly used in competitive games. The Elo system assigns a numerical rating to each player based on their performance in the game.

Code Snippets: Database, Calculating Statistics, and Score

Creating tables:

```sql
-- auto-generated definition
create table int_player_info
(
    player_name        varchar(30) not null
        constraint player_info_pkey
            primary key,
    total_points       smallint,
    total_moves_number smallint,
    games_played       smallint,
    games_won          smallint,
    total_game_time    integer
);

alter table int_player_info
    owner to game;
```

```sql
-- auto-generated definition
create table int_players
(
    game_id     integer     not null,
    player_name varchar(30) not null
        constraint check_player_name_length
            check (length((player_name)::text) >= 2),
    move_number smallint    not null,
    game_time   integer,
    role        varchar(20) not null,
    balance     smallint    not null,
    location    numeric(2)  not null,
    constraint players_pkey
        primary key (game_id, player_name, move_number)
);

alter table int_players
    owner to game;
```

Adding a new player to the INT_PLAYER_INFO table:

```java
public void addNewPlayer(String playerName) throws SQLException {
    if (doesPlayerExist(playerName)) {
        System.out.println("Player already exists.");
        return;
    }

    String sql = "INSERT INTO INT_PLAYER_INFO (player_name, total_points, total_moves_number, games_played, games_won, total_game_time) " +
            "VALUES (?, ?, ?, ?, ?, ?)";

    try (Connection connection = DriverManager.getConnection(JDBC_URL);
         PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString( parameterIndex: 1, playerName);
        pstmt.setShort( parameterIndex: 2, (short) 1000);
        pstmt.setShort( parameterIndex: 3, (short) 0);
        pstmt.setShort( parameterIndex: 4, (short) 0);
        pstmt.setShort( parameterIndex: 5, (short) 0);
        pstmt.setShort( parameterIndex: 6, (short) 0);
        pstmt.executeUpdate();
    }
}
```

Adding player's game data to the INT_PLAYERS table:

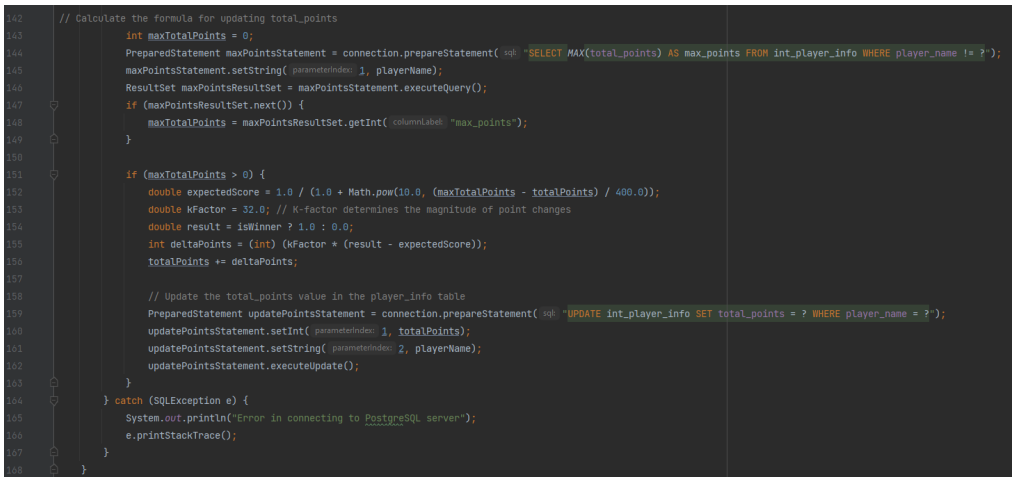
```java
public void addPlayerDataToPlayersTable(int gameId, String playerName, int moveNumber, long gameTime, String role, int balance, int location) {
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        PreparedStatement statement = conn.prepareStatement( sql: "INSERT INTO INT_PLAYERS (game_id, player_name, move_number, game_time, role, balance, location) VALUES (?, ?, ?, ?, ?, ?, ?)");
        statement.setInt( parameterIndex: 1, gameId);
        statement.setString( parameterIndex: 2, playerName);
        statement.setInt( parameterIndex: 3, moveNumber);
        statement.setLong( parameterIndex: 4, gameTime);
        statement.setString( parameterIndex: 5, role);
        statement.setInt( parameterIndex: 6, balance);
        statement.setInt( parameterIndex: 7, location);
        statement.executeUpdate();
        System.out.println("Data added successfully.");
    } catch (SQLException e) {
        System.out.println("Error updating data: " + e.getMessage());
    }
}
```

Updating player's data after the end of the game:

```java
    public void updatePlayerInfo(String playerName, boolean isWinner, int gameId) {
        try (Connection connection = DriverManager.getConnection(JDBC_URL);
             PreparedStatement statement = connection.prepareStatement( sql: "UPDATE int_player_info " +
                     "SET total_moves_number = total_moves_number + t.max_move, " +
                     "games_played = games_played + 1, " +
                     "games_won = games_won + ?, " +
                     "total_game_time = total_game_time + (SELECT MAX(game_time) FROM int_players WHERE player_name = ? AND game_id = ?) " +
                     "FROM (SELECT player_name, MAX(move_number) AS max_move FROM int_players WHERE game_id = ? GROUP BY player_name) t, " +
                     "(SELECT MAX(game_time) AS max_game_time FROM int_players WHERE player_name = ?) u, " +
                     "(SELECT total_points FROM int_player_info WHERE player_name = ?) v " +
                     "WHERE int_player_info.player_name = t.player_name AND int_player_info.player_name = ? " +
                     "AND v.total_points != 0")) {
            statement.setInt( parameterIndex: 1, isWinner ? 1 : 0);
            statement.setString( parameterIndex: 2, playerName);
            statement.setInt( parameterIndex: 3, gameId);
            statement.setInt( parameterIndex: 4, gameId);
            statement.setString( parameterIndex: 5, playerName);
            statement.setString( parameterIndex: 6, playerName);
            statement.setString( parameterIndex: 7, playerName);
            int updatedRows = statement.executeUpdate();
            if (updatedRows > 0) {
                System.out.println("Successfully updated data for player: " + playerName);
            }

            // Retrieve the updated total_points value for the player
            PreparedStatement selectStatement = connection.prepareStatement( sql: "SELECT total_points FROM int_player_info WHERE player_name = ?");
            selectStatement.setString( parameterIndex: 1, playerName);
            ResultSet resultSet = selectStatement.executeQuery();
            int totalPoints = 0;
            if (resultSet.next()) {
                totalPoints = resultSet.getInt( columnLabel: "total_points");
            }

```



```
// Calculate the formula for updating total_points
            int maxTotalPoints = 0;
            PreparedStatement maxPointsStatement = connection.prepareStatement( sql: "SELECT MAX(total_points) AS max_points FROM int_player_info WHERE player_name != ?");
            maxPointsStatement.setString( parameterIndex: 1, playerName);
            ResultSet maxPointsResultSet = maxPointsStatement.executeQuery();
            if (maxPointsResultSet.next()) {
                maxTotalPoints = maxPointsResultSet.getInt( columnLabel: "max_points");
            }

            if (maxTotalPoints > 0) {
                double expectedScore = 1.0 / (1.0 + Math.pow(10.0, (maxTotalPoints - totalPoints) / 400.0));
                double kFactor = 32.0; // K-factor determines the magnitude of point changes
                double result = isWinner ? 1.0 : 0.0;
                int deltaPoints = (int) (kFactor * (result - expectedScore));
                totalPoints += deltaPoints;

                // Update the total_points value in the player_info table
                PreparedStatement updatePointsStatement = connection.prepareStatement( sql: "UPDATE int_player_info SET total_points = ? WHERE player_name = ?");
                updatePointsStatement.setInt( parameterIndex: 1, totalPoints);
                updatePointsStatement.setString( parameterIndex: 2, playerName);
                updatePointsStatement.executeUpdate();
            }
        } catch (SQLException e) {
            System.out.println("Error in connecting to PostgreSQL server");
            e.printStackTrace();
        }
    }
```

In the following method, we are updating the player's data and score using the Elo system. The Elo system is a mathematical model commonly used in competitive games to calculate the relative skill levels of players. Here's how it works:

1. We first retrieve the maximum total points achieved by any player, excluding the current player. This is done using the SQL query SELECT MAX(total_points) AS max_points FROM int_player_info WHERE player_name != ?.
2. If there are other players with a higher score (i.e., maxTotalPoints > 0), we calculate the expected score for the current player using the Elo formula: expectedScore = 1 / (1 + Math.pow(10, (maxTotalPoints - totalPoints) / 400)). The expected score represents the predicted performance of the player against opponents with higher scores.
3. The kFactor is a constant that determines the magnitude of point changes. It controls how quickly the player's score adjusts based on the outcome of the game.
4. The result is set to 1.0 if the player is the winner, indicating a victory, and 0.0 if the player is the loser, indicating a defeat.
5. We calculate the deltaPoints using the formula deltaPoints = (int) (kFactor * (result - expectedScore)). This value represents the change in points for the player based on the outcome and the expected score.
6. The player's totalPoints are updated by adding deltaPoints to the current score: totalPoints += deltaPoints.
7. Finally, we update the total_points value in the int_player_info table for the specific player using the SQL statement UPDATE int_player_info SET total_points = ? WHERE player_name = ?.

By updating the player's score using the Elo system, we ensure that the score reflects the player's performance relative to other players in the game. The Elo system takes into account the outcome of the game, the ratings of the players involved, and adjusts the scores accordingly.

Methods getting player's statistics:

```java
public int getTotalPoints(String playerName) {
    int totalPoints = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT * FROM INT_PLAYER_INFO WHERE player_name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            totalPoints = rs.getInt( columnLabel: "total_points");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return totalPoints;
}

1 usage  Tymur
public int getTotalMovesNumber(String playerName) {
    int totalMovesNumber = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT * FROM INT_PLAYER_INFO WHERE player_name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            totalMovesNumber = rs.getInt( columnLabel: "total_moves_number");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return totalMovesNumber;
}
```

```java
public int getGamesPlayed(String playerName) {
    int gamesPlayed = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT * FROM INT_PLAYER_INFO WHERE player_name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            gamesPlayed = rs.getInt( columnLabel: "games_played");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return gamesPlayed;
}

1 usage  Tymur
public int getGamesWon(String playerName) {
    int gamesWon = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT * FROM INT_PLAYER_INFO WHERE player_name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            gamesWon = rs.getInt( columnLabel: "games_won");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return gamesWon;
}
```

```java
public int getTotalGameTime(String playerName) {
    int totalGameTime = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT * FROM INT_PLAYER_INFO WHERE player_name = ?";
        PreparedStatement pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            totalGameTime = rs.getInt( columnLabel: "total_game_time");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return totalGameTime;
}
```

Methods for calculating average data for the players:

```java
public float getAveragePoints() {
    float averagePoints = 0;
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        String query = "SELECT ROUND(AVG(total_points), 2) AS avg_points FROM int_player_info WHERE player_name != 'PC'";
        PreparedStatement pstmt = conn.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();
        if (rs.next()) {
            averagePoints = rs.getFloat( columnLabel: "avg_points");
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return averagePoints;
}
```

Similar methods are used to calculate other values such as average total moves number, average total games played, average total games won, and average total game time.

Get Leaderboard:

```java
public String getLeaderboard(String playerName) {
    StringBuilder sb = new StringBuilder();
    try (Connection conn = DriverManager.getConnection(JDBC_URL)) {
        // Retrieve the top 5 players by total points, excluding "PC" and ""
        String query = "SELECT player_name, total_points, total_moves_number, games_played, games_won, total_game_time "
                + "FROM int_player_info "
                + "WHERE player_name != 'PC' "
                + "AND player_name != '' "
                + "ORDER BY total_points DESC "
                + "LIMIT 5";
        PreparedStatement pstmt = conn.prepareStatement(query);
        ResultSet rs = pstmt.executeQuery();

        // Build the leaderboard string
        sb.append("Top 5 Players by Total Points:\n");
        int rank = 1;
        while (rs.next()) {
            sb.append(rank).append(". ").append(rs.getString( columnLabel: "player_name")).append(" - ").append(rs.getInt( columnLabel: "total_points")).append(" points\n");
            rank++;
        }

        // Retrieve the current player's data and their rank, excluding "PC"
        query = "SELECT COUNT(*) AS rank "
                + "FROM int_player_info "
                + "WHERE player_name != 'PC' AND player_name != '' AND total_points > (SELECT total_points FROM int_player_info WHERE player_name = ?)";
        pstmt = conn.prepareStatement(query);
        pstmt.setString( parameterIndex: 1, playerName);
        rs = pstmt.executeQuery();
        if (rs.next()) {
            int playerRank = rs.getInt( columnLabel: "rank") + 1; // Add 1 to account for tied ranks
            query = "SELECT * FROM int_player_info WHERE player_name = ? AND player_name != 'PC' AND player_name != ''";
```

```
                pstmt = conn.prepareStatement(query);
                pstmt.setString( parameterIndex: 1, playerName);
                rs = pstmt.executeQuery();
                if (rs.next()) {
                    // Build the player's data string and add it to the leaderboard
                    sb.append("\nYour Rank: ").append(playerRank).append("\n");
                    sb.append("Name: ").append(rs.getString( columnLabel: "player_name")).append("\n");
                    sb.append("Points: ").append(rs.getInt( columnLabel: "total_points")).append("\n");
                }
            }
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return sb.toString();
    }

```

Check if the data is duplicated:

```
public boolean isDuplicatePlayerData(int gameId, String playerName, String role, int balance, int location) {
    boolean duplicateDataExists = false;

    try (Connection connection = DriverManager.getConnection(JDBC_URL);
         PreparedStatement statement = connection.prepareStatement( sql: "SELECT COUNT(*) AS count FROM int_players WHERE game_id = ? AND player_name = ? AND role = ? AND balance = ? AND location = ?')) {

        statement.setInt( parameterIndex: 1, gameId);
        statement.setString( parameterIndex: 2, playerName);
        statement.setString( parameterIndex: 3, role);
        statement.setInt( parameterIndex: 4, balance);
        statement.setInt( parameterIndex: 5, location);

        ResultSet resultSet = statement.executeQuery();
        if (resultSet.next()) {
            int count = resultSet.getInt( columnLabel: "count");
            duplicateDataExists = count > 0;
        }
    } catch (SQLException e) {
        // Handle any potential exceptions here
        e.printStackTrace();
    }

    return duplicateDataExists;
}
```

Game Over Statistics:

```
@   public String getGameStatistics(Game game, int gameId) {
        int initialBalance = game.getInitialBalance();
        int maxGameTimeInSeconds = 0;
        StringBuilder result = new StringBuilder();
        Map<String, List<String>> playerStats = new HashMap<>();

        try {
            // Assuming you have a connection to the database
            Connection connection = DriverManager.getConnection(JDBC_URL);
            Statement statement = connection.createStatement();

            // Fetch the game data for the specified gameId
            ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM int_players WHERE game_id = " + gameId +
                    " ORDER BY player_name, move_number");

            String currentPlayer = "";
            String currentRole = "";
            int moneyLost = 0;
            int moneyEarned = 0;

            while (resultSet.next()) {
                String playerName = resultSet.getString( columnLabel: "player_name");
                String role = resultSet.getString( columnLabel: "role");
                int balance = resultSet.getInt( columnLabel: "balance");
                int gameTimeInSeconds = resultSet.getInt( columnLabel: "game_time");

                // Calculate money lost and money earned based on balance changes
                if (!currentPlayer.equals(playerName)) {
                    if (!currentPlayer.equals("")) {
                        // Add the statistics for the previous player to the map
                        List<String> playerStat = new ArrayList<>();
                        playerStat.add("Player: " + currentPlayer);
                        playerStat.add("Role: " + currentRole);
```

```
                    playerStat.add("Money Lost: " + moneyLost);
                    playerStat.add("Money Earned: " + moneyEarned);
                    playerStats.put(currentPlayer, playerStat);
                }

                currentPlayer = playerName;
                currentRole = role;
                moneyLost = 0;
                moneyEarned = 0;
            }

            if (balance < initialBalance) {
                moneyLost += (initialBalance - balance);
            } else if (balance > initialBalance) {
                moneyEarned += (balance - initialBalance);
            }

            initialBalance = balance;

            // Calculate the maximum game time in seconds
            if (gameTimeInSeconds > maxGameTimeInSeconds) {
                maxGameTimeInSeconds = gameTimeInSeconds;
            }
        }

        // Add the statistics for the last player to the map
        List<String> playerStat = new ArrayList<>();
        playerStat.add("Player: " + currentPlayer);
        playerStat.add("Role: " + currentRole);
        playerStat.add("Money Lost: " + moneyLost);
        playerStat.add("Money Earned: " + moneyEarned);
        playerStats.put(currentPlayer, playerStat);

        // Close the database resources
        resultSet.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    // Format the statistics into two columns
    int columnWidth = 30;
    for (Map.Entry<String, List<String>> entry : playerStats.entrySet()) {
        List<String> playerStat = entry.getValue();
        for (int i = 0; i < playerStat.size(); i += 2) {
            String leftData = playerStat.get(i);
            String rightData = (i + 1 < playerStat.size()) ? playerStat.get(i + 1) : "";
            result.append(String.format("%-" + columnWidth + "s %-" + columnWidth + "s%n", leftData, rightData));
        }
        result.append(System.lineSeparator());
    }

    // Convert game time to minutes and seconds
    int minutes = maxGameTimeInSeconds / 60;
    int seconds = maxGameTimeInSeconds % 60;

    // Append the game ID and maximum game time in minutes and seconds
    result.append(System.lineSeparator())
            .append("Game ID: ").append(gameId)
            .append(", Game Time: ").append(minutes).append(" minutes ").append(seconds).append(" seconds");

    // Return the formatted result as a string
    return result.toString();
}
}
```

Drawing the graph:

```
public void drawBalanceGraph(int gameId, LineChart<Number, Number> lineChart) {
    ObservableList<XYChart.Series<Number, Number>> seriesList = FXCollections.observableArrayList();

    try {
        // Assuming you have a connection to the database
        Connection connection = DriverManager.getConnection( url: "jdbc:postgresql://10.134.178.23:5432/game?user=game&password=7sur7");
        Statement statement = connection.createStatement();

        // Fetch the game data for the specified gameId
        ResultSet resultSet = statement.executeQuery( sql: "SELECT * FROM int_players WHERE game_id = " + gameId +
                " ORDER BY player_name, move_number");

        String currentPlayer = "";
        XYChart.Series<Number, Number> currentSeries = null;

        while (resultSet.next()) {
            String playerName = resultSet.getString( columnLabel: "player_name");
            int moveNumber = resultSet.getInt( columnLabel: "move_number");
            int balance = resultSet.getInt( columnLabel: "balance");

            // Check if the player has changed
            if (!currentPlayer.equals(playerName)) {
                // If the previous player series exists, add it to the series list
                if (currentSeries != null) {
                    seriesList.add(currentSeries);
                }

                // Initialize a new series for the new player
                currentPlayer = playerName;
                currentSeries = new XYChart.Series<>();
                currentSeries.setName(currentPlayer);
            }

            // Add data points to the series
            assert currentSeries != null;
            currentSeries.getData().add(new XYChart.Data<>(moveNumber, balance));
        }

        // Add the last player's series to the series list
        if (currentSeries != null) {
            seriesList.add(currentSeries);
        }

        // Close the database resources
        resultSet.close();
        statement.close();
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }

    // Add series to the chart
    lineChart.getData().addAll(seriesList);
}
}
```

Adding new players to INT_PLAYER_INFO happens in StartPresenter setView():

```
private void setView() throws SQLException {
    String playerName1 = view.getTextField1().getText();
    String playerName2 = view.getTextField2().getText();

    if (Objects.equals(playerName2, b: "")) {
        playerName2 = "PC";
    }

    model.getStatistics().addNewPlayer(playerName1);
    model.getStatistics().addNewPlayer(playerName2);

    if (view.getCheckBox().isSelected()) {
        model.setDifficulty(view.getDifficultySelector().getValue().toString());
    }
```

Collecting data for every move happens in GamePresenter UpdateView():

```
public void updateView() {
    // update the view with player data
    view.setName1(model.getPlayer()[0].getName() + "\n$" + model.getPlayer()[0].getBalance());
    view.setName2(model.getPlayer()[1].getName() + "\n$" + model.getPlayer()[1].getBalance());
    subGamePresenter.showPlayer();
    subGamePresenter.updateEstate();

    // record player data in the statistics table for the current move
    boolean duplicateDataExists = false;
    for (Player player : model.getPlayer()) {
        String playerName = player.getName();

        // Skip saving data if player's name is empty or "PC"
        if (playerName.isEmpty() || playerName.equals("PC")) {
            continue;
        }

        String role = player.getRole().toString();
        int balance = player.getBalance();
        int location = player.getPawn().getLocation();

        // Check if the same player data exists in the int_player_info table
        duplicateDataExists = model.getStatistics().isDuplicatePlayerData(gameId, playerName, role, balance, location);

        if (!duplicateDataExists) {
            long currentTime = System.currentTimeMillis(); // get the current time
            long elapsedTime = (currentTime - startTime) / 1000; // calculate the elapsed time in seconds
            int elapsedTimeAdjusted = (int) elapsedTime; // store the adjusted elapsed time

            model.getStatistics().addPlayerDataToPlayersTable(gameId, playerName, moveNumber, elapsedTimeAdjusted, role, balance, location);
        }
    }
```

Updating player's data in INT_PLAYER_INFO happens in GameOverPresenter GameOverPresenter(Game model, GameOverView view, int gameId):

```
public GameOverPresenter(Game model, GameOverView view, int gameId) {
    gameID = gameId;
    setModel(model);
    setView(view);
    String winner = model.whoWon();
    view.getDescription().setText(winner + " won by bankrupting the competition!");
    boolean player1IsWinner = model.getPlayer()[0].getName().equals(winner);
    model.getStatistics().updatePlayerInfo(model.getPlayer()[0].getName(), player1IsWinner, gameId);
    model.getStatistics().updatePlayerInfo(model.getPlayer()[1].getName(), !player1IsWinner, gameId);
    addEventHandlers();
}
```

Rule-based intelligence

Expert rules

First, I searched on google how to win at monopoly and found these rules from: Insider-How-To-Win-At-Monopoly. Yes, this is for monopoly and not anti-monopoly. But the rules to win are almost the same, next to some Role based advantages. I took some of these rules to make my AI. The rules go as followed:
1. Be aggressive in the beginning.
2. Buy orange and red Properties.
3. Do not save your money.
4. Do not bother with Utilities.
5. Develop three houses as fast as possible.
6. Depending on the role the AI will act accordingly.

Code Explanation

The AI is split into multiple difficulties. The more difficult the AI is the more rules it will implement. Hard mode is an accumulation of its own rule set and the lower difficulties. It goes as forth: the haphazard difficulty is just a random choice. Easy mode will have an added rule: not being able to go bankrupt while buying and always buying to be aggressive. The medium difficulty will actively look at the tile types which are advantageous or disadvantageous and act accordingly. The hard mode will try to actively block the other players' progress or advance it's own depending on its own role.

Difficulty Haphazard: randomly buys a tile.

```
/**
 * Method to randomly buys a tile
 *
 * @return boolean to buy or not to buy
 * that's the question!
 */
public boolean difficultyHapHazard() {
    Random rand = new Random();
    return rand.nextBoolean();
}
```

Difficulty Easy: Implements rule 1 and 3 while not becoming bankrupt in one line of code. AI will not go under the 300 balance after purchase.

```
/**
 * Easy mode: PC always buys unless the balance will go under 300 after buy.
 *
 * @param player Current Player
 * @param board Game board
 * @return Boolean, buy or not
 */
public boolean difficultyEasy(Player player, Board board) {
    return (player.getBalance() - board.getLoc(player).getAmount()) > 300;
}
```

Difficulty Medium: Checks for the tile type for rules 2 and 4 respectively, if AI lands in Boston or Philadelphia, it will use the easy mode method to buy if it won't bring it under 300. If the AI lands on a Utility tile, he will do the haphazard method to randomly buy the utility or not. If it's neither of those tiles it will buy the property if it won't go under 400 balance.

```
/**
 * Medium mode: Checks which tile the PC is standing on IF the PC stands on a green or
orange tile easy difficulty will apply
 * If PC stands a utility tile it will implement the haphazard difficulty a.k.a. random
choice
 * If any other tile, PC will always buy unless final balance is lower than 400
 *
 * @param player Current Player
 * @param board  Game board
 * @return boolean, buy or not
 */
public boolean difficultyMedium(Player player, Board board) {
    if (board.getLoc(player.getPawn().getLocation()).getLocation().equals("Boston") ||
board.getLoc(player.getPawn().getLocation()).getLocation().equals("Philadelphia")) {
        return difficultyEasy(player, board);
    } else if ((player.getBalance() -
board.getLoc(player.getPawn().getLocation()).getAmount()) > 400) {
        if (board.getLoc(player.getPawn().getLocation()).getTileType() ==
TileType.UTILITY) {
            return difficultyHapHazard();
        } else {
            return true;
        }
    }
    return false;
}
```

Difficulty Hard: This time the AI will check depending on its own role. If it is a MONOPOLIST, it will check if the tile that it is standing on belongs to one of the streets, he already has a property on and will then do the easy method to buy it. If the AI is a COMPETITOR, he will then check if the tile is on the street of the opponent. If so, then he will buy it using the easy method. If neither apply it will default back to the medium difficulty method.

```
 * @param player  Current Player
 * @param board   Game board
 * @param player2 The opponent
 * @return buy or not
 */
public boolean difficultyHard(Player player, Board board, Player player2) {
    if (board.getLoc(player).getTileType() == TileType.PROPERTY) {
        if (player.getRole() == Role.MONOPOLIST) {
            if (player.getOwnedPropertyName(board, player)) {
                return true;
            } else {
                return difficultyEasy(player, board);
            }
        } else if (player.getRole() == Role.COMPETITOR) {
            if (player2.getOwnedPropertyName(board, player)) {
                return true;
            } else {
                return difficultyMedium(player, board);
            }
        }
    }
    return false;
}
```

MVP Activating: The event handler will first check if the current player is the PC if so, it will activate the PCTurn method. Else the button will activate for the player itself.

```
if (model.getCurrentPlayer().getIsPC()) {
    PCTurn();
} else {
    view.getRollButton().setOnAction(event -> {
        turn();
        view.getDiceSound().stop();
        view.getDiceSound().seek(view.getDiceSound().getStartTime());
        view.getDiceSound().play();
    });

}
```

The PC's turn: the pc's turn is set in a timeline. The AI will follow the same course as a normal player. The timeline is a set of timed actions the Ai decides and activates. These actions are animated and given sounds so you will get the illusion of the AI pressing buttons and doing its turn.

```
private void PCTurn() {
    Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(0.2), event -> {
    }), new KeyFrame(Duration.seconds(0.3), event -> {
        view.getRollButton().setId("hoverOn");
    }), new KeyFrame(Duration.seconds(0.8), event -> {
        view.getRollButton().fire();
        view.getRollButton().setId("hoverOff");

    }), new KeyFrame(Duration.seconds(2), event -> {
        if (model.getCurrentLocation().getTileType() == TileType.PROPERTY ||
model.getCurrentLocation().getTileType() == TileType.TRANSPORT ||
model.getCurrentLocation().getTileType() == TileType.UTILITY) {
            if (view.getYesno().isVisible()) {
                if (getDifficulty()) {
                    view.getYes().setId("hoverOn");
                } else {
                    view.getNo().setId("hoverOn");
                }
            }
        }
    }), new KeyFrame(Duration.seconds(2.5), event -> {
        if (model.getCurrentLocation().getTileType() == TileType.PROPERTY ||
model.getCurrentLocation().getTileType() == TileType.TRANSPORT ||
model.getCurrentLocation().getTileType() == TileType.UTILITY) {
            if (view.getYesno().isVisible()) {
                if (getDifficulty()) {
                    view.getYes().fire();
                    view.getYes().setId("hoverOff");
                } else {
                    view.getNo().fire();
                    view.getNo().setId("hoverOff");
                }

            }
        }
    }), new KeyFrame(Duration.seconds(3), event -> {
        view.getEndTurn().setId("hoverOn");
```

```
    }), new KeyFrame(Duration.seconds(3.5), event -> {
        view.getEndTurn().fire();
        view.getEndTurn().setId("hoverOff");

    }), new KeyFrame(Duration.seconds(4), event -> {
        if (model.getCurrentPlayer().HasDoubles()) {
            PCTurn();
        }
    }));

    timeline.play();
}
```

Making decisions: In the previous piece of code getDifficulty() is marked. This method will get the difficulty you selected in the start screen and will execute the corresponding difficulty we explained above.

```
private boolean getDifficulty() {
    switch (model.getDifficulty()) {
        case "Haphazard":
            return model.getPc().difficultyHapHazard();
        case "Easy":
            return model.getPc().difficultyEasy(model.getCurrentPlayer(),
model.getBoard());
        case "Normal":
            return model.getPc().difficultyMedium(model.getCurrentPlayer(),
model.getBoard());
        case "Hard":
            return model.getPc().difficultyHard(model.getCurrentPlayer(),
model.getBoard(), model.getNotCurrentPlayer());
    }
    return false;
}
```

Different Situations: If the AI rolled a double it will activate PCTurn() again. If the ‘Human player’ ends his turn he will activate PCTurn().

Game details

The current game isn’t an exact copy of Anti-monopoly. Anti-monopoly has trading and selling as a main component which I don’t have. I did implement some rules like needing to own a street as a monopolist to ask for double rent or needing to pick different cards depending on your role. It’s difficult to add more rules because the only thing the player does is decide if they want to buy a property or not.