I. Problem

RACING ARENA.

Develops a simple game "**Racing Arena**" with a server as a Referee and N clients (players) for each game (N is defined by the server in advance, 2<= N <= 10). The rules of the each set are:

1. **For each player, a player needs to register to the server to join the game and choose the nickname. You cannot choose the nickname which is already registered. The nickname is composed by the following characters 'a'...'z', 'A'...'Z', '0'...'9', '_' and the length is not longer than 10 characters. If the player uses the same nickname with other players, the server will ask the player to choose another nickname. For each successful registration player, the server announces "Registration Completed Successfully".**
2. **When the server receives enough connections (players), the server sends the length of the race and the start position to the players (the length of the race should be greater than 3 and less than 26). The length of the race is defined by the administrator who manages the server and the start position is 1.**
3. **For each set:**

1. Server randomly choose 2 integers (from -10,000 to 10,000) and a operator (+,-,*,/,%) and sends them to the players.
2. The players receive the expression and the players should send the result of that expression to the server.
3. When the server receives all the answer from the players, the server send the correct answer to the players and the points is granted as follow:
    i. If the player has a wrong answer or loses their turn (out of time), they get -1 point. The duration for each question is defined by the server at the beginning of the game.
    ii. If the player has a correct answer and he/she is the fastest player, he/she will get M points, which M is the total points that the other players lose. The other players who have the correct answer get 1 point.
    iii. When a player has wrong answers 3 times consecutively, that player is disqualified, the number of the remaining players is updated and is announced to the players.
    iv. The points in each question are the steps that players move in the race (move forward if they get positive points and move backward if they get negative points and they cannot move behind the starting points).
4. Server updates the points, the position of each player in the race and announces them to the players.

4. **Repeat the game until we have a winner who reaches the finish first.**

5. **Server starts another set.**
6. **You should set the countdown for each turn.**

II. Game Story

The client sends out the Username to register. Then the server begins to send to the client the question. The question is a random number you need to file from the problem. Example: 1 + 3 - ? = 32.

```
int arr[5] = {QRandomGenerator::global()->bounded(1, 20), QRandomGenerator::global()->bounded(1, 20),
              QRandomGenerator::global()->bounded(1, 20), QRandomGenerator::global()->bounded(1, 20)};
QString operArr[4] = {"+", "-","*", "="};
arr[4] = arr[0] + arr[1] + arr[2] * arr[3];

int mask = QRandomGenerator::global()->bounded(5);
m_answer = QString::number(arr[mask]);

arr[mask] = -1;
QString question = "";
for(int i=0;i<4;i++){
    QString number;
    if(arr[i] == -1){
        number = "?";
    } else {
        number = QString::number(arr[i]);
    }

    question += number + " " + operArr[i] + " ";
}
if(arr[4] == -1){
    question += "?";
} else {
    question += QString::number(arr[4]);
```

The client after receiving the question will have 10s to answer the question. If answer is correct and that client is the fastest one answer this question, this client will get point equal to total lost point of other clients.

```
if(fastestClient != nullptr){
    fastestClient->setScore(-1);
    fastestClient->setScore(totalLostPoints);
}
```

The chat box will show out the last answer and the chat of clients, scoreboard will show the point off remaining player. This process repeat until the last client remain will be the winner.

III. Structure of the packets and steps/condition to check packets.

The client will connect to the server through the IP address. Then the clients needed to input the Username. Then the client will send that username to the server as a JSON using the QDataStream.

```
message[QStringLiteral("type")] = QStringLiteral("login");
message[QStringLiteral("username")] = userName;
// send the JSON using QDataStream
clientStream << QJsonDocument(message).toJson(QJsonDocument::Compact);
```

By using the Q_ASSERT(sender). The server will obtain the JSON that client had sent. Server will check if that Json file contain the value of the username or not. If that username duplicate, the server will create QJsonObject contain the type of the message, success or not, reason. Then send this Json file to the sender.

```
QJsonObject message;
message[QStringLiteral("type")] = QStringLiteral("login");
message[QStringLiteral("success")] = false;
message[QStringLiteral("reason")] = QStringLiteral("duplicate username");
sendJson(sender, message);
```

If this is a success, the server will setUserName then send back a success message and write out a log through the server screen and broadcast to others client that a new client had connected.

```
sender->setUserName(newUserName);
QJsonObject successMessage;
successMessage[QStringLiteral("type")] = QStringLiteral("login");
successMessage[QStringLiteral("success")] = true;
sendJson(sender, successMessage);
QJsonObject connectedMessage;
connectedMessage[QStringLiteral("type")] = QStringLiteral("newuser");
connectedMessage[QStringLiteral("username")] = newUserName;
broadcast(connectedMessage, sender);
```

After the client had successful connect to the server. The server begin to send question to client. First server will check if clients had an answer before or not.

```
if(QString::compare(m_answer, QStringLiteral("NAN")) != 0){ // if not the first time
```

If this is the first time, the server will run the getQuestion() function to create the question. After that, the server will create a QJsonObject: questionMessage contain: type of the message, question, score list, last answer of the client and broadcast it to client.

```
if(QString::compare("NAN", m_answer) == 0)
    lastAnswer = "";
QString sendQuestion = getQuestion();
QJsonObject questionMessage;
questionMessage[QStringLiteral("type")] = QStringLiteral("questionarrive");
questionMessage[QStringLiteral("question")] = sendQuestion;
questionMessage[QStringLiteral("scorelist")] = userScoreList;
questionMessage[QStringLiteral("lastanswer")] = lastAnswer;
broadcast(questionMessage, nullptr);
```

If this is not the first time, the server will check the answer that client send, if this client answer corrected and the fastest. Server will increase win streak by 1 and decrease the point of other client by 1, the score of the fastest one will increase by total lost points of the other.

```
int totalLostPoints = 0;
if(QString::compare(m_answer, QStringLiteral("NAN")) != 0){ // if not the first time
    for (ServerWorker *worker : m_clients) {
        Q_ASSERT(worker);
        if(QString::compare(worker->getLastAnswer(), m_answer) == 0){
            worker->setScore(1);
            worker->setWinStreak(1);
        } else{
            worker->setScore(-1);
            worker->setWinStreak(-1);
            totalLostPoints += 1;
        }
    }
}
if(fastestClient != nullptr){
    fastestClient->setScore(-1);
    fastestClient->setScore(totalLostPoints);
}
```

To announce the win streak of the user, we create a List of user name score. Then push into that list name, win streak and score of users. Then we sort that list by the user Score. Then we will print out all the clients with winstreak % 3 = 0.

```
int i = 1;
for(QPair<QPair<QString, int>, int> userNameScore : userName_Score){
    const QString userName = userNameScore.first.first;
    userScoreList += QString::number(i) + "." + userName + ": " + QString::number(userNameScore.second) + "\n";
    const int winStreak = userNameScore.first.second;
    if(winStreak % 3 == 0 && winStreak != 0){
        announceWinStreak += userName + " have " + QString::number(winStreak) + " winstreak!!\n";
    }
    i++;
}
```

In the client, we will check if the user name that we choose is good or not. We will check the type of the message, if the message don't have the type, we will ignore or if we already logged in we will ignore too.

```
const QJsonValue typeVal = docObj.value(QLatin1String("type"));
if (typeVal.isNull() || !typeVal.isString())
    return; // a message with no type was received so we just ignore it
if (typeVal.toString().compare(QLatin1String("login"), Qt::CaseInsensitive) == 0) { //It's a login message
    if (m_loggedIn)
        return; // if we are already logged in we ignore
    // the success field will contain the result of our attempt to login
```

If the server send out the fail login, we will extract the reason of the failure from the JSON and notify it via the loginError signal.

```
  const QJsonValue reasonVal = docObj.value(QLatin1String("reason"));
  emit loginError(reasonVal.toString());
```

If this is a chat message, we extract the text field containing the chat text and extract the sender field containing the username of the sender. We notify a new message was received via the messageReceived signal

```
  } else if (typeVal.toString().compare(QLatin1String("message"), Qt::CaseInsensitive) == 0) { //It's a chat message
      // we extract the text field containing the chat text
      const QJsonValue textVal = docObj.value(QLatin1String("text"));
      // we extract the sender field containing the username of the sender
      const QJsonValue senderVal = docObj.value(QLatin1String("sender"));
      if (textVal.isNull() || !textVal.isString())
          return; // the text field was invalid so we ignore
      if (senderVal.isNull() || !senderVal.isString())
          return; // the sender field was invalid so we ignore
      // we notify a new message was received via the messageReceived signal
      emit messageReceived(senderVal.toString(), textVal.toString());
```

If a new user join the chat, we extract the username of the new user and notify of the new user via the userJoined signal.

```
 else if (typeVal.toString().compare(QLatin1String("newuser"), Qt::CaseInsensitive) == 0) { // A user joined the chat
   // we extract the username of the new user
   const QJsonValue usernameVal = docObj.value(QLatin1String("username"));
   if (usernameVal.isNull() || !usernameVal.isString())
       return; // the username was invalid so we ignore
   // we notify of the new user via the userJoined signal
   emit userJoined(usernameVal.toString());
```

If a user left the chat, we extract the username of the new user and notify of the user disconnection the userLeft signal.

```
 else if (typeVal.toString().compare(QLatin1String("userdisconnected"), Qt::CaseInsensitive) == 0) { // A user left the chat
    // we extract the username of the new user
   const QJsonValue usernameVal = docObj.value(QLatin1String("username"));
   if (usernameVal.isNull() || !usernameVal.isString())
       return; // the username was invalid so we ignore
   // we notify of the user disconnection the userLeft signal
   emit userLeft(usernameVal.toString());
```

When the user send an answer to server, we create a QDataStream operating on the socket, set the version so that programs compiled with different versions of Qt can agree on how to serialise, create the JSON we want to send and send the JSON to server using QDataStream.

```
  if (text.isEmpty())
      return; // We don't send empty messages
  // create a QDataStream operating on the socket
  QDataStream clientStream(m_clientSocket);
  // set the version so that programs compiled with different versions of Qt can agree on how to serialise
  clientStream.setVersion(QDataStream::Qt_5_7);
  // Create the JSON we want to send
  QJsonObject message;
  message[QStringLiteral("type")] = QStringLiteral("answer");
  message[QStringLiteral("answer")] = text;
  // send the JSON using QDataStream
  clientStream << QJsonDocument(message).toJson();
```

When the user send an message to other client, we create a QDataStream operating on the socket, set the version so that programs compiled with different versions of Qt can agree on how to serialise, create the JSON we want to send and send the JSON to server using QDataStream.

```
if (text.isEmpty())
    return; // We don't send empty messages
// create a QDataStream operating on the socket
QDataStream clientStream(m_clientSocket);
// set the version so that programs compiled with different versions of Qt can agree on how to serialise
clientStream.setVersion(QDataStream::Qt_5_7);
// Create the JSON we want to send
QJsonObject message;
message[QStringLiteral("type")] = QStringLiteral("message");
message[QStringLiteral("text")] = text;
// send the JSON using QDataStream
clientStream << QJsonDocument(message).toJson();
```

IV. Team Information

No.	Name	StudentID	Contribution (%)
1	Nguyễn Quốc Bình	1751052	40
2	Phạm Nguyễn Huy Hoàng	1751068	40
3	Lê Thanh Bình	1751051	20

V. Score Sheet:

No.	Requirements	Score	Evaluate
1	Use C/C++, Java, C#	2	2
2	Implement whole gameplay properly	3	3
3	Socket Non-blocking	2	2
4	Have a good GUI (MFC, WPF, Swing, etc.)	3	2.5
	Total	**10**	**9.5**

VI. Reference

How to create a simple chat application: https://wiki.qt.io/WIP-How_to_create_a_simple_chat_application?fbclid=IwAR0aq0ot9EZpRJPEJW_HX_wMsvVtlhjjWBe_9hO1KBm2tTjucHLQul1klU8
QT library: https://www.qt.io/product/development-tools