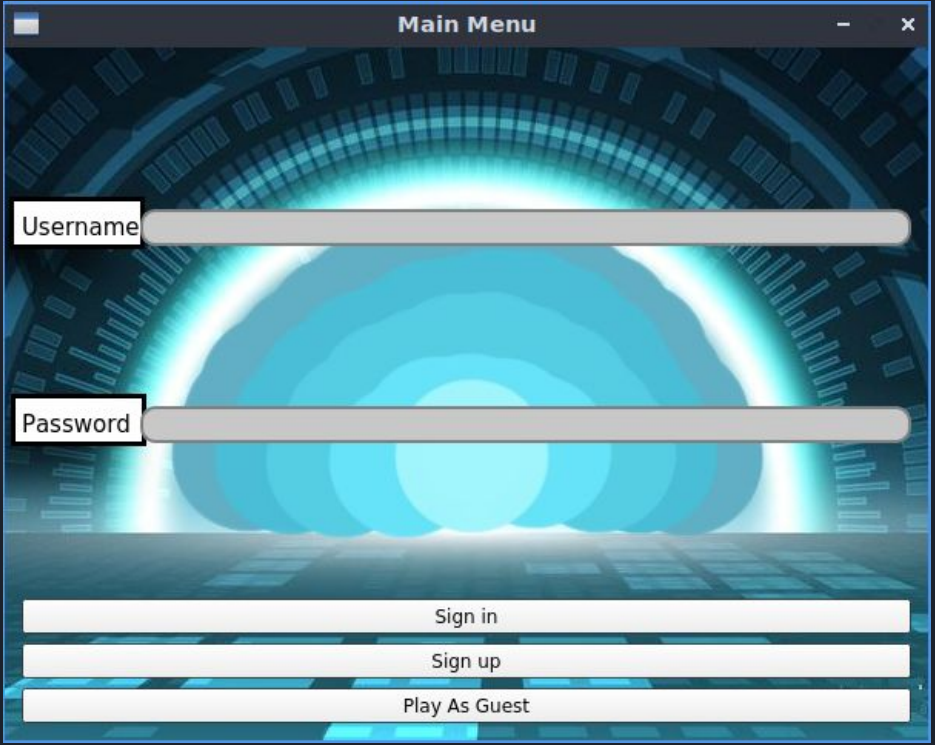
Qt Games

Hassan El Khatib
Ali Hijjawi

GENERAL OPERATIONS AND FUNCTIONALITIES

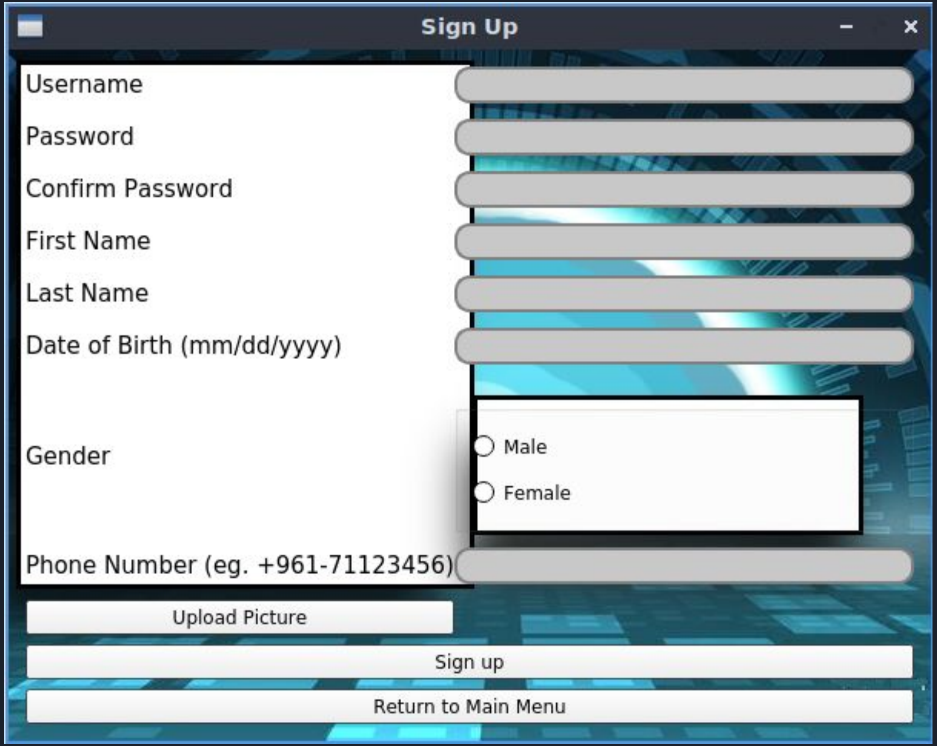
Main Widget:

- The player is prompted to sign in if he already has a registered account
- The player can also choose to play as a guest where his scores will not be scored
- The player can also sign up if he does not have an account



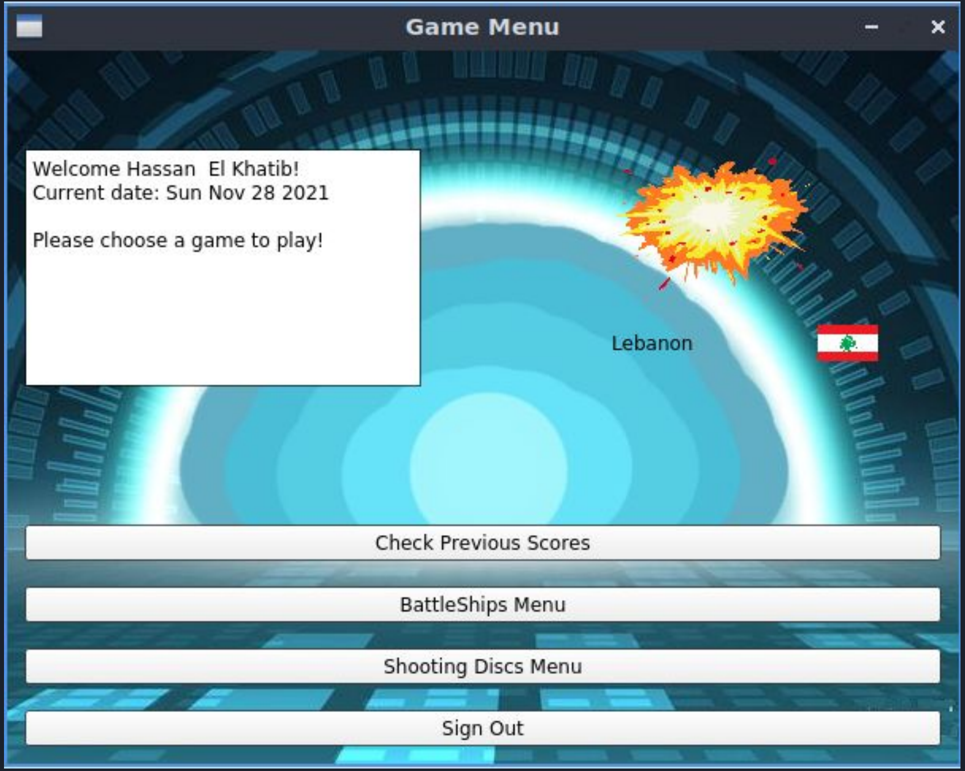
Sign Up Page:

- The player is prompted to enter all his information
- All fields are required; as in the player can not sign up with any of the fields empty
- In the backend we check if the entered password fits the requirements (at least 8 characters and contain at least one number, upper and lower case letters)



Game Menu:

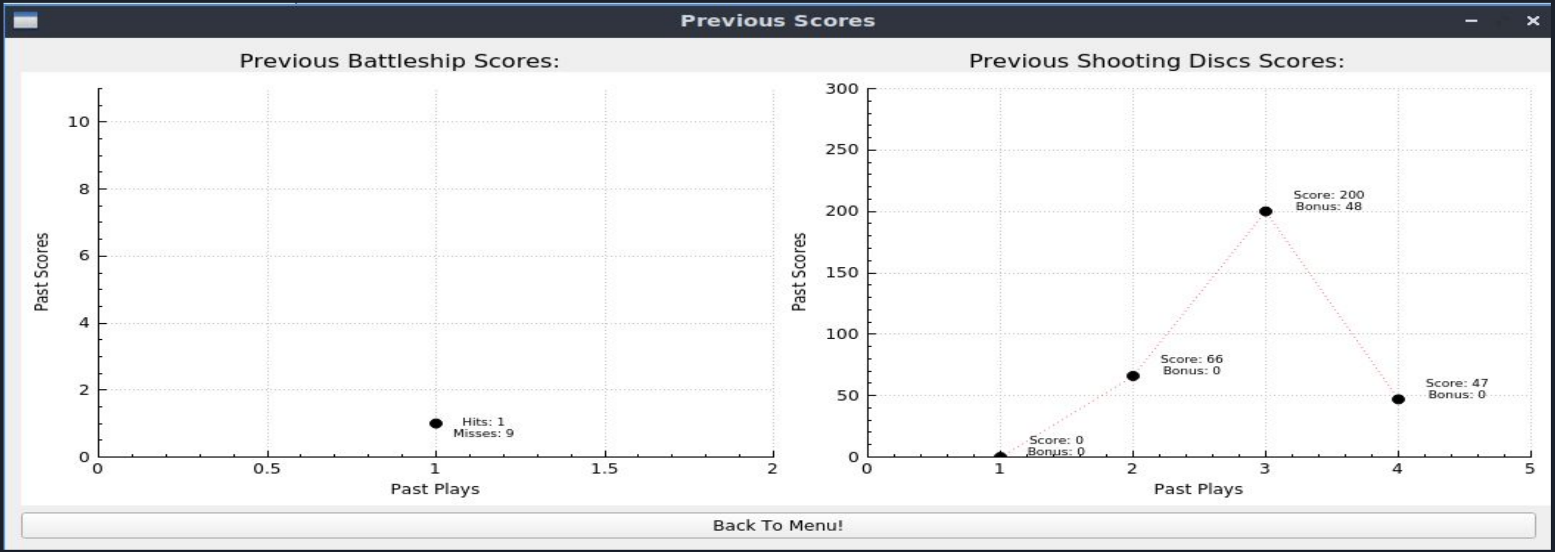
- The player is shown the current date along with his full name, country, flag and profile picture
- If the player is a logged in user he can check his previous scores
- The player can later chose to either play Game 1 or Game 2 using the corresponding buttons


Welcome Hassan El Khatib!
Current date: Sun Nov 28 2021

Please choose a game to play!

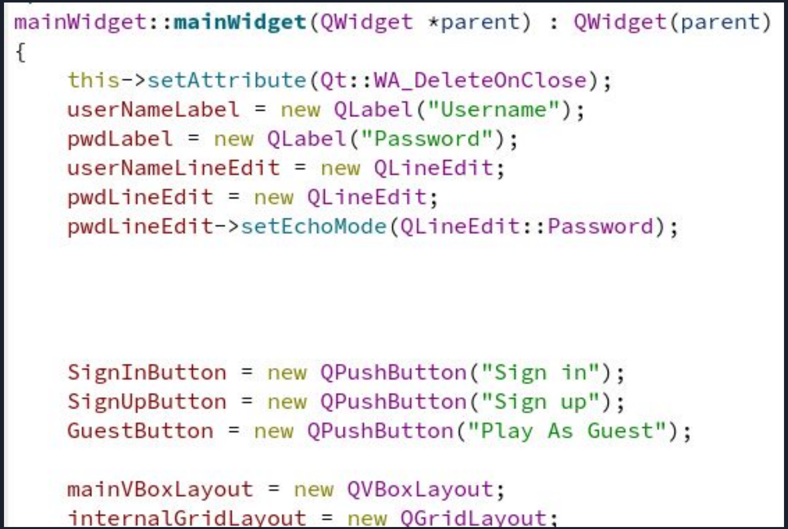
Previous Scores:

- The player is shown the history of his scores displayed as line graphs to see his progress.
- To achieve this we used an external Qt plotting library named QCustomPlot and fetched the data from the accounts JSON file

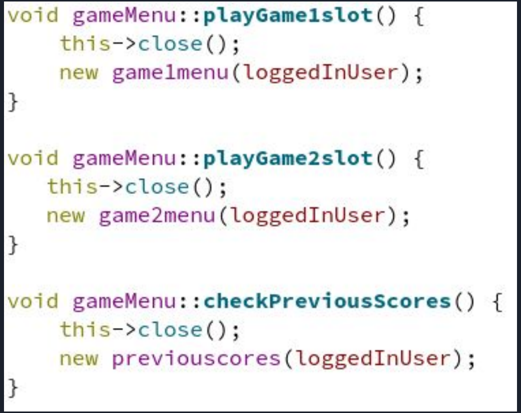


General Operations

- All the previously mentioned windows are QWidgets.
- To add functionally and style to these widgets we added different Qt items such as Labels, Line Edits and Push Buttons with custom slots to custom layouts for positioning.
- Navigation between widgets is also achieved by the usage of push buttons with slots.
- The deletion of dynamically allocated memory is also handled when navigating to avoid any memory leaks.



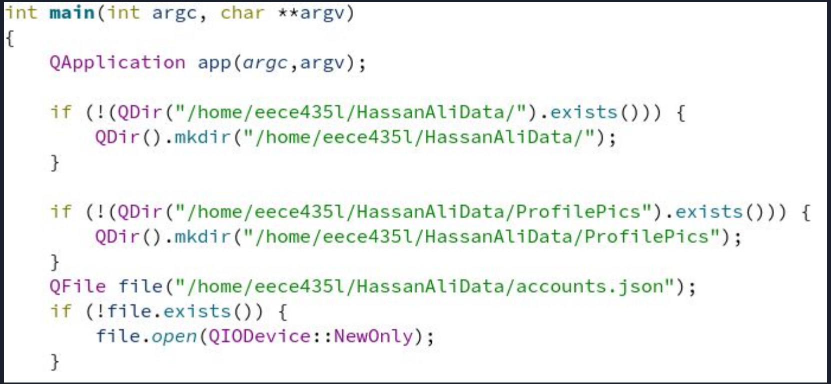
```
mainWidget::mainWidget(QWidget *parent) : QWidget(parent)
{
    this->setAttribute(Qt::WA_DeleteOnClose);
    userNameLabel = new QLabel("Username");
    pwdLabel = new QLabel("Password");
    userNameLineEdit = new QLineEdit;
    pwdLineEdit = new QLineEdit;
    pwdLineEdit->setEchoMode(QLineEdit::Password);



    SignInButton = new QPushButton("Sign in");
    SignUpButton = new QPushButton("Sign up");
    GuestButton = new QPushButton("Play As Guest");

    mainVBoxLayout = new QVBoxLayout;
    internalGridLayout = new QGridLayout;
```



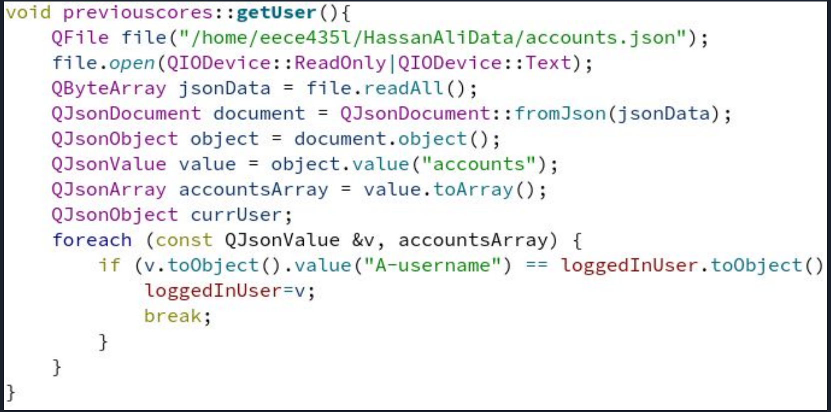
```
void gameMenu::playGame1slot() {
    this->close();
    new game1menu(loggedInUser);
}

void gameMenu::playGame2slot() {
   this->close();
   new game2menu(loggedInUser);
}

void gameMenu::checkPreviousScores() {
    this->close();
    new previouscores(loggedInUser);
}
```

General Operations (2)

- To fetch and receive our data for accounts, game rules, and game information we used JSON files.
- For the parsing of the data we had used the QJsonObject class. When we need to write data to the JSON file we we also use this class to encapsulate our data into JSON objects
- To avoid any issues with paths our accounts JSON file and profile pictures are located in the home directory of the OS. This is done by ensuring the directory is created at run time.



```
int main(int argc, char **argv)
{
    QApplication app(argc,argv);

    if (!(QDir("/home/eece435l/HassanAliData/").exists())) {
        QDir().mkdir("/home/eece435l/HassanAliData/");
    }

    if (!(QDir("/home/eece435l/HassanAliData/ProfilePics").exists())) {
        QDir().mkdir("/home/eece435l/HassanAliData/ProfilePics");
    }
    QFile file("/home/eece435l/HassanAliData/accounts.json");
    if (!file.exists()) {
        file.open(QIODevice::NewOnly);
    }
```



```
void previouscores::getUser(){
    QFile file("/home/eece435l/HassanAliData/accounts.json");
    file.open(QIODevice::ReadOnly|QIODevice::Text);
    QByteArray jsonData = file.readAll();
    QJsonDocument document = QJsonDocument::fromJson(jsonData);
    QJsonObject object = document.object();
    QJsonValue value = object.value("accounts");
    QJsonArray accountsArray = value.toArray();
    QJsonObject currUser;
    foreach (const QJsonValue &v, accountsArray) {
        if (v.toObject().value("A-username") == loggedInUser.toObject()
            loggedInUser=v;
            break;
        }
    }
}
```

Battleships

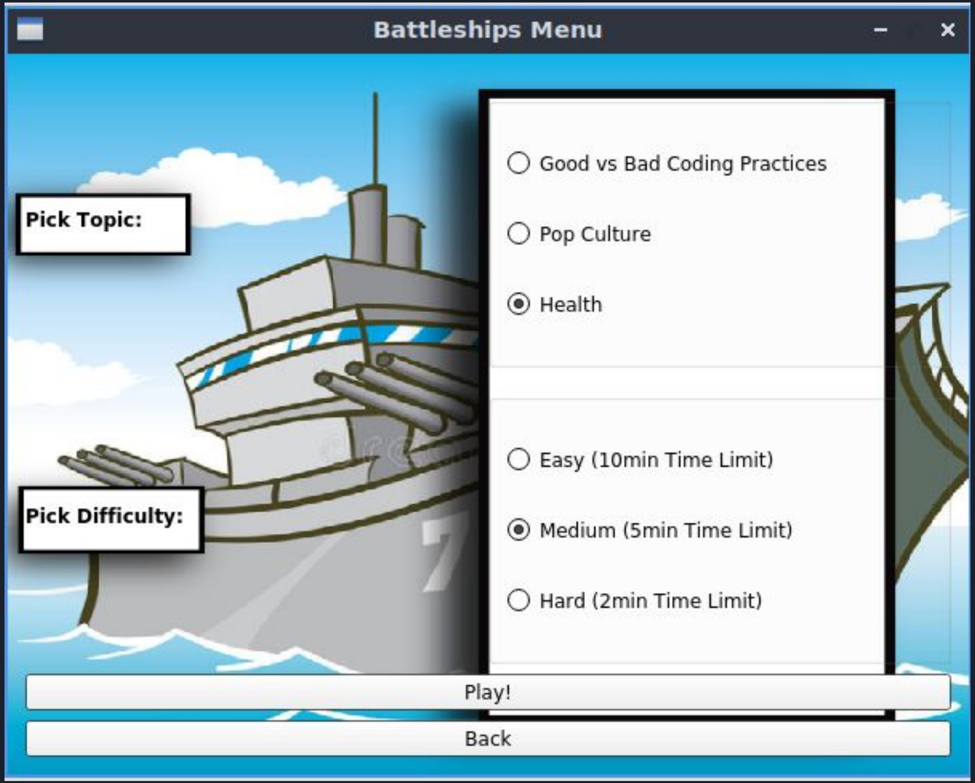
Game Rules:

- Game is played on a 4x4 Grid
- Game ends when all 16 grid boxes are discovered or when the timer ends
- Timer ranges from 10 mins to 2 mins depending on difficulty
- When clicking on a grid box there are two possibilities:
  - A enemy ship is underneath the grid box and the player is prompted to answer a question. Should he answer correctly the enemy ship is destroyed and his successful hits are incremented. Should he incorrectly one of his ships are destroyed and a miss icon is presented on that grid box along with his unsuccessful hits being incremented
  - There is no ship beneath the grid box and a miss represented by an X is placed on the grid indicating the missed hit.
  - In both scenarios the player loses one of his 16 tries.
- When all 16 tries are consumed if the player answered 7 questions correctly he wins, however if he answers 4 incorrectly he loses.
- The questions range from three topics good vs bad coding practices, pop culture, and health practices. At any time an admin can change the questions by editing the JSON files that include the question
- If the player is not a guest, his scores are saved in the accounts JSON file to be later fetched when he wants to view the history of his games.

Battleships Menu

- The player can choose the topic to play for the questions as well as the difficulty (time available to win)

- The games starts when they press play or click F1



Pick Topic:

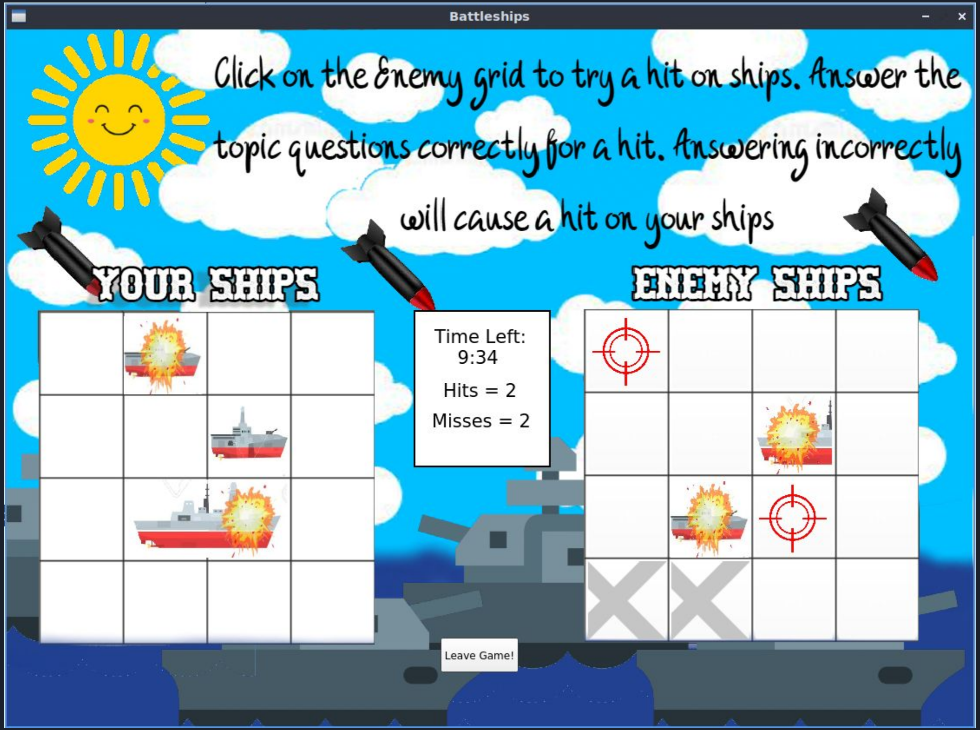
Good vs Bad Coding Practices
Pop Culture
[✓ Health]

Pick Difficulty:

Easy (10min Time Limit)
[✓ Medium (5min Time Limit)]
Hard (2min Time Limit)

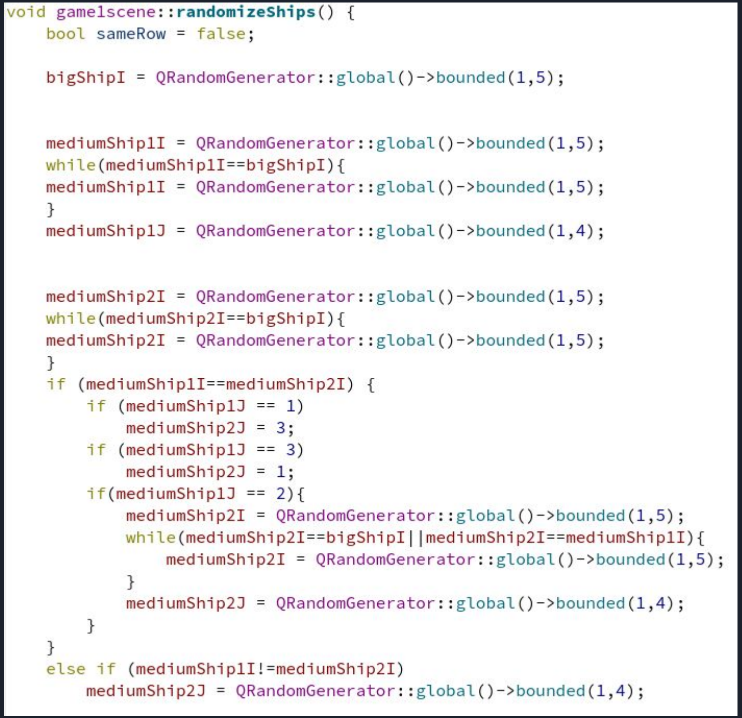
Game Functionalities

- The timer starts as the game starts, and the ships are spawned randomly on the enemy grid
- The player chooses to shoot a grid spot and either misses, or gets a question through a popup window with a drop down answer grid
- They get a question according to the topic chosen, and if answered correctly they hit, and miss if they answer incorrectly
- When they miss ie choose an incorrect answer, their own ships get hit


Battleships
Click on the Enemy grid to try a hit on ships. Answer the topic questions correctly for a hit. Answering incorrectly will cause a hit on your ships
YOUR SHIPS
ENEMY SHIPS
Time Left: 9:34
Hits = 2
Misses = 2
Leave Game!

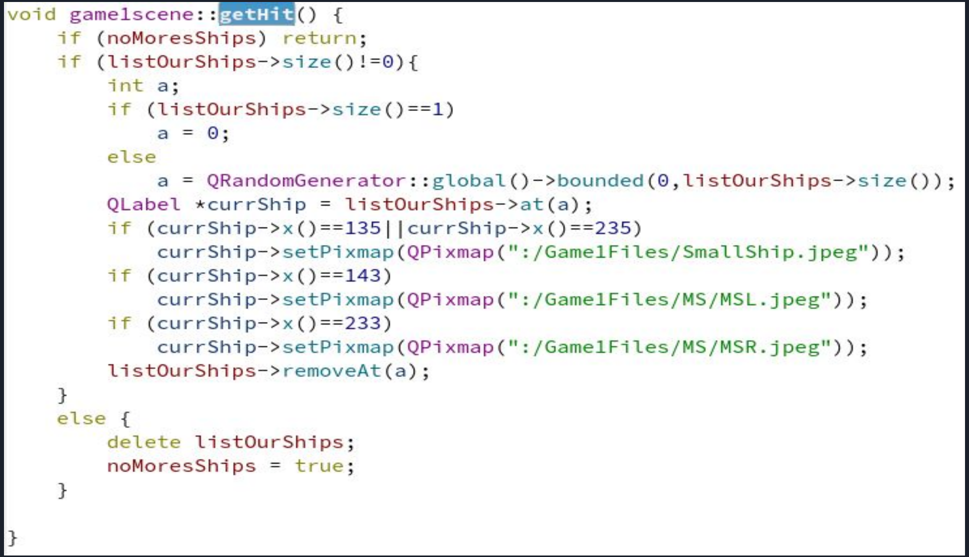
Important Functions

- A function that randomizes the ship spawning on enemy grid, where we randomize the i and j coordinates, and anytime the ship collides with another previous one, we re-randomize



```
void game1scene::randomizeShips() {
    bool sameRow = false;

    bigShipI = QRandomGenerator::global()->bounded(1,5);


    mediumShip1I = QRandomGenerator::global()->bounded(1,5);
    while(mediumShip1I==bigShipI){
    mediumShip1I = QRandomGenerator::global()->bounded(1,5);
    }
    mediumShip1J = QRandomGenerator::global()->bounded(1,4);


    mediumShip2I = QRandomGenerator::global()->bounded(1,5);
    while(mediumShip2I==bigShipI){
    mediumShip2I = QRandomGenerator::global()->bounded(1,5);
    }
    if (mediumShip1I==mediumShip2I) {
        if (mediumShip1J == 1)
            mediumShip2J = 3;
        if (mediumShip1J == 3)
            mediumShip2J = 1;
        if(mediumShip1J == 2){
            mediumShip2I = QRandomGenerator::global()->bounded(1,5);
            while(mediumShip2I==bigShipI||mediumShip2I==mediumShip1I){
                mediumShip2I = QRandomGenerator::global()->bounded(1,5);
            }
            mediumShip2J = QRandomGenerator::global()->bounded(1,4);
        }
    }
    else if (mediumShip1I!=mediumShip2I)
        mediumShip2J = QRandomGenerator::global()->bounded(1,4);
```

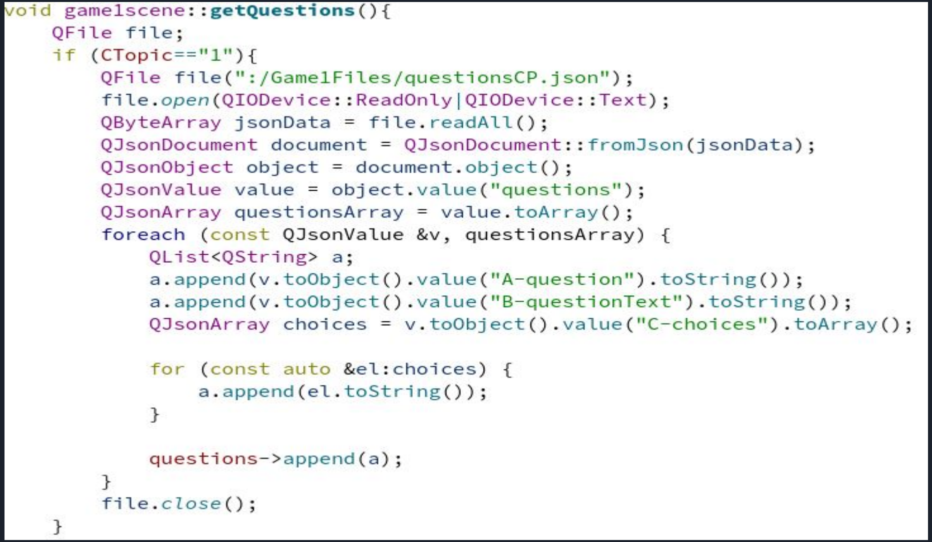
- When the player answers incorrectly, we randomly hit their own ships and change the pixmap to a hit ship



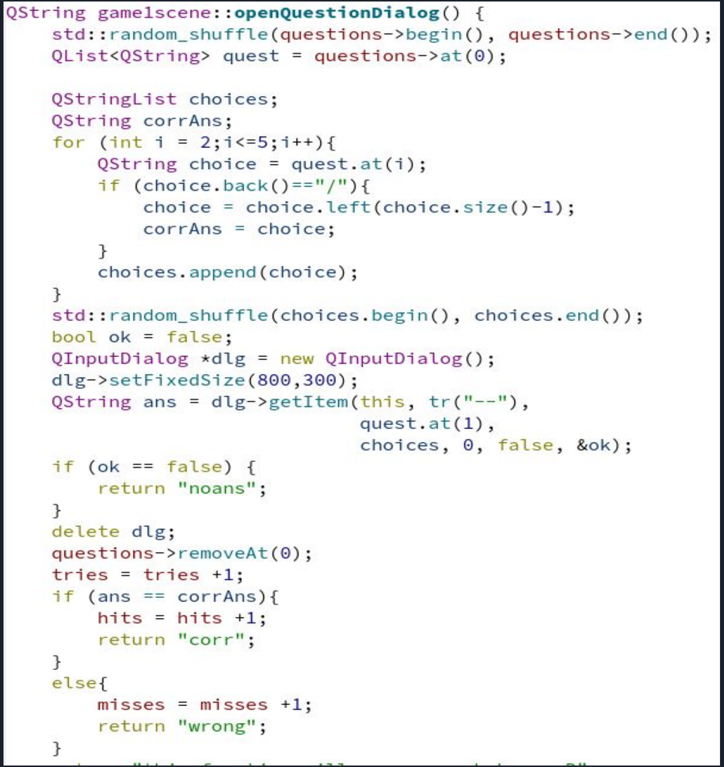
```
void game1scene::getHit() {
    if (noMoresShips) return;
    if (listOurShips->size()!=0){
        int a;
        if (listOurShips->size()==1)
            a = 0;
        else
            a = QRandomGenerator::global()->bounded(0,listOurShips->size());
        QLabel *currShip = listOurShips->at(a);
        if (currShip->x()==135||currShip->x()==235)
            currShip->setPixmap(QPixmap(":/Game1Files/SmallShip.jpeg"));
        if (currShip->x()==143)
            currShip->setPixmap(QPixmap(":/Game1Files/MS/MSL.jpeg"));
        if (currShip->x()==233)
            currShip->setPixmap(QPixmap(":/Game1Files/MS/MSR.jpeg"));
        listOurShips->removeAt(a);
    }
    else {
        delete listOurShips;
        noMoresShips = true;
    }

}
```

Important Functions (2)

- A function that opens the message box for the user to answer the question. It returns whether the player answered correctly or answered incorrectly or did not answer.



```
void game1scene::getQuestions(){
    QFile file;
    if (CTopic=="1"){
        QFile file(":/Game1Files/questionsCP.json");
        file.open(QIODevice::ReadOnly|QIODevice::Text);
        QByteArray jsonData = file.readAll();
        QJsonDocument document = QJsonDocument::fromJson(jsonData);
        QJsonObject object = document.object();
        QJsonValue value = object.value("questions");
        QJsonArray questionsArray = value.toArray();
        foreach (const QJsonValue &v, questionsArray) {
            QList<QString> a;
            a.append(v.toObject().value("A-question").toString());
            a.append(v.toObject().value("B-questionText").toString());
            QJsonArray choices = v.toObject().value("C-choices").toArray();

            for (const auto &el:choices) {
                a.append(el.toString());
            }

            questions->append(a);
        }
        file.close();
    }
```

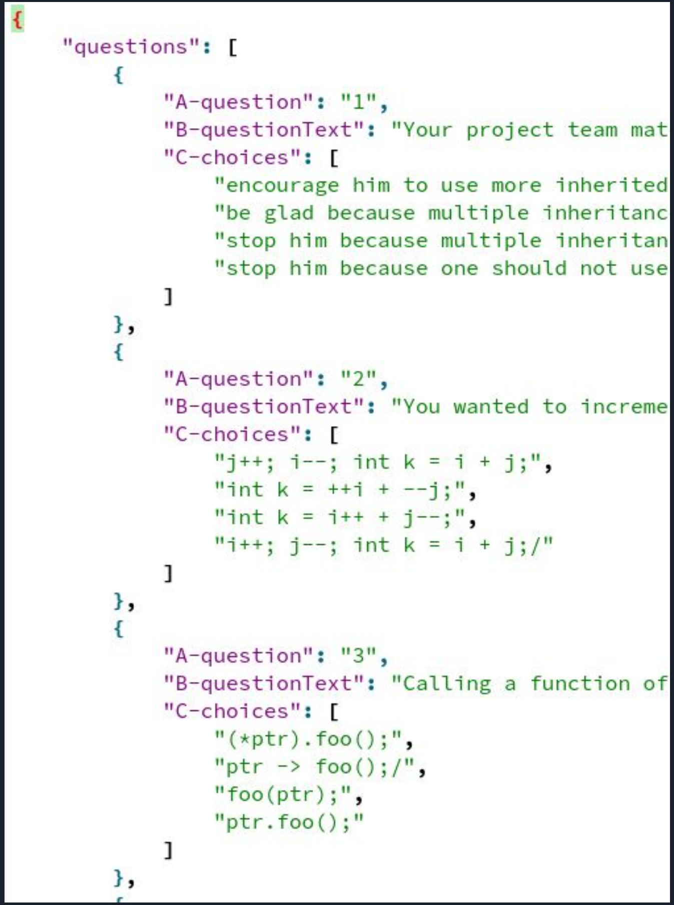


```
QString game1scene::openQuestionDialog() {
    std::random_shuffle(questions->begin(), questions->end());
    QList<QString> quest = questions->at(0);

    QStringList choices;
    QString corrAns;
    for (int i = 2;i<=5;i++){
        QString choice = quest.at(i);
        if (choice.back()=="/"){
            choice = choice.left(choice.size()-1);
            corrAns = choice;
        }
        choices.append(choice);
    }
    std::random_shuffle(choices.begin(), choices.end());
    bool ok = false;
    QInputDialog *dlg = new QInputDialog();
    dlg->setFixedSize(800,300);
    QString ans = dlg->getItem(this, tr("--"),
                               quest.at(1),
                               choices, 0, false, &ok);
    if (ok == false) {
        return "noans";
    }
    delete dlg;
    questions->removeAt(0);
    tries = tries +1;
    if (ans == corrAns){
        hits = hits +1;
        return "corr";
    }
    else{
        misses = misses +1;
        return "wrong";
    }
```

- A function that gets all the questions in the json file and appends them to a class variable to be later fetched when needed in game

Questions

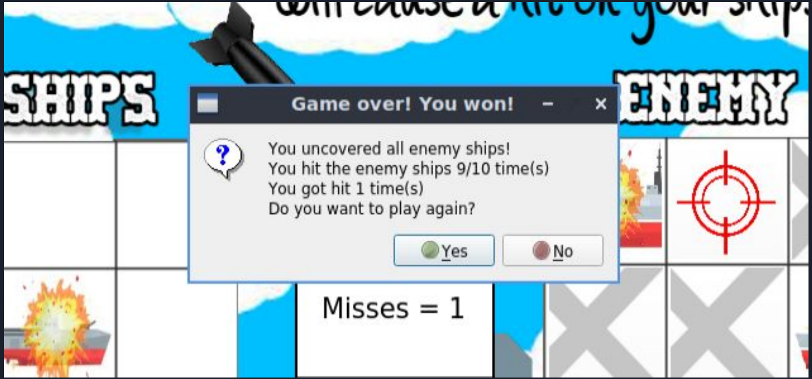
- For each topic, the questions are stored in a JSON file where each question is an index of a JSON Array. Each question is a JSON Object that contains the question number, text and the choices
- In order to distinguish the correct answer from the wrong ones, the right answer is always ended with a forward slash which is later removed when parsing the question



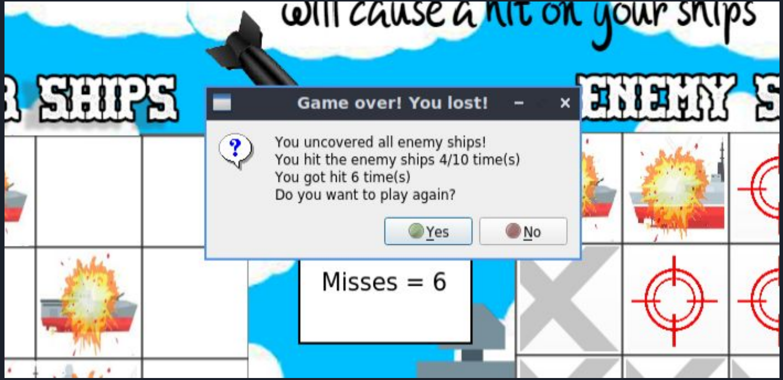
```
{
    "questions": [
        {
            "A-question": "1",
            "B-questionText": "Your project team mat
            "C-choices": [
                "encourage him to use more inherited
                "be glad because multiple inheritanc
                "stop him because multiple inheritan
                "stop him because one should not use
            ]
        },
        {
            "A-question": "2",
            "B-questionText": "You wanted to increme
            "C-choices": [
                "j++; i--; int k = i + j;",
                "int k = ++i + --j;",
                "int k = i++ + j--;",
                "i++; j--; int k = i + j;/"
            ]
        },
        {
            "A-question": "3",
            "B-questionText": "Calling a function of
            "C-choices": [
                "(*ptr).foo();",
                "ptr -> foo();/",
                "foo(ptr);",
                "ptr.foo();"
            ]
        },
```

Game Over

- They win when they answer 7 correct answers


Game over! You won!

You uncovered all enemy ships!
You hit the enemy ships 9/10 time(s)
You got hit 1 time(s)
Do you want to play again?


Game over! You lost!

You uncovered all enemy ships!
You hit the enemy ships 4/10 time(s)
You got hit 6 time(s)
Do you want to play again?

- They lose if they answer 4 incorrectly

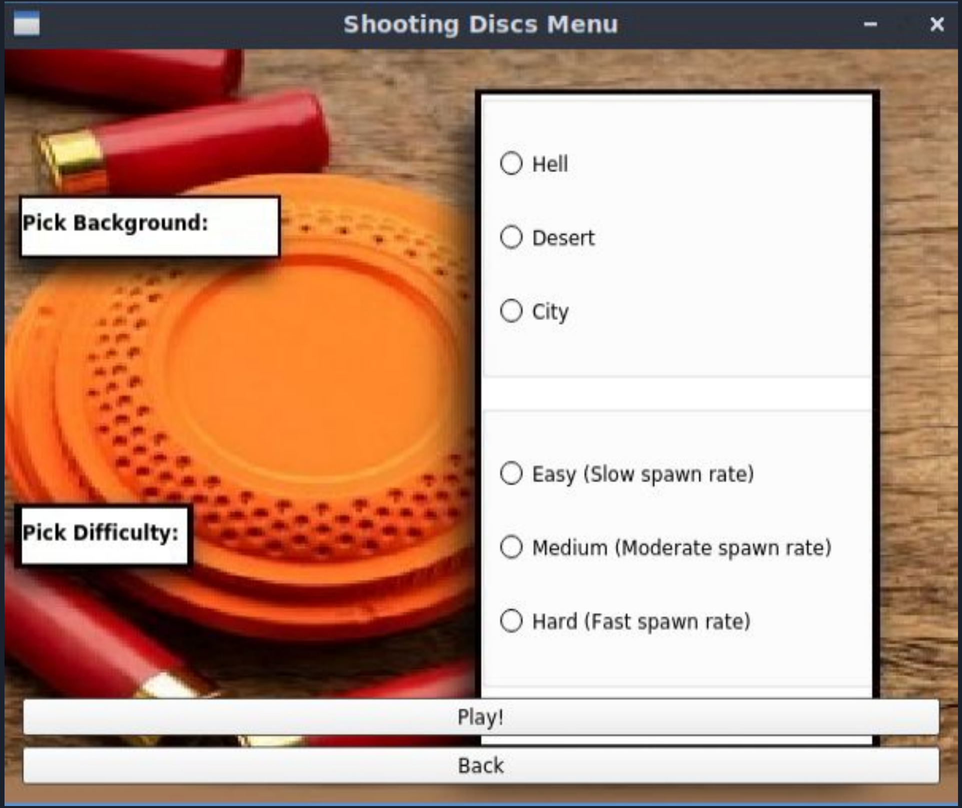
Shooting Cars

Game Rules:

- Game is played on a rectangular window
- Game ends when all lives are consumed
- There are three possible places where cars can spawn from and the spawn position of the cars are randomly generated at runtime.
- There are three difficulties for the game; the more difficult the game the faster the spawn rate of the cars
- Black cars contribute to 3 points, white cars contribute to 5, while red cars contribute to 7.
- The speed of cars moving down the lanes is doubled every 30 points reaching a maximum of 16x speed at 120 score
- When all lives are consumed if the score of the player is greater than the winning score hw wins the game and any points beyond the winning score are saved separately as bonus points. If he receives less than the winning score he loses the game.
- At any time an admin can change the rules of the game by simply changing the winning score and total lives in the corresponding JSON file.
- If the player is not a guest, his scores are saved in the accounts JSON file to be later fetched when he wants to view the history of his games.

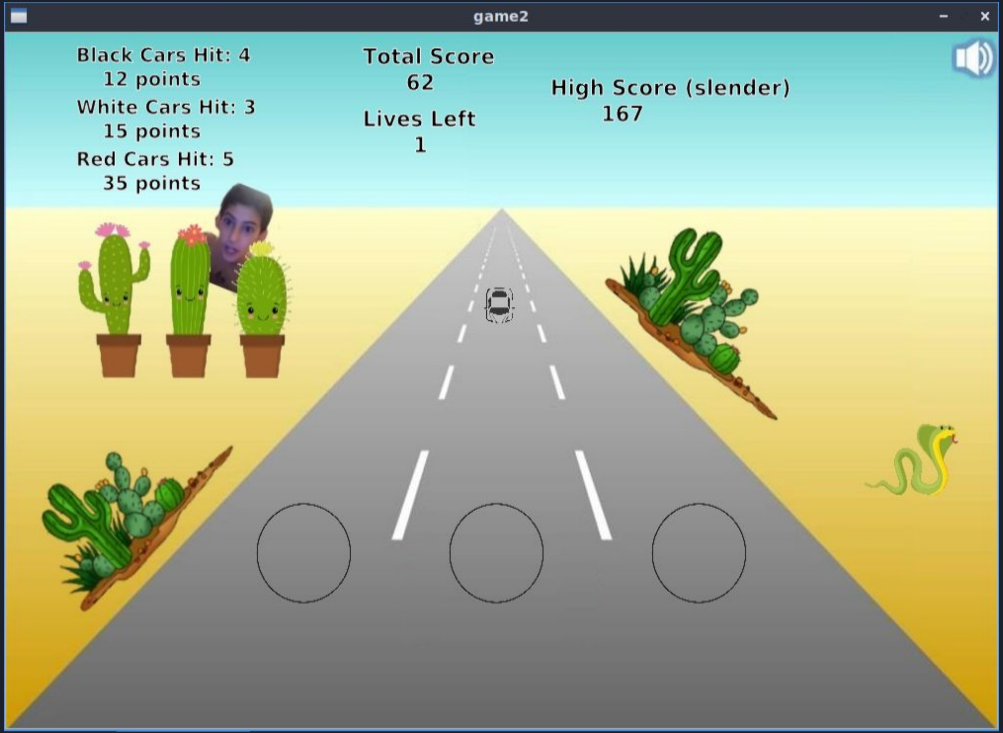
Shooting Cars Menu

- The player can choose the background to play in as well as the difficulty (spawn rate for the cars)
- The games starts when they press play or click F1



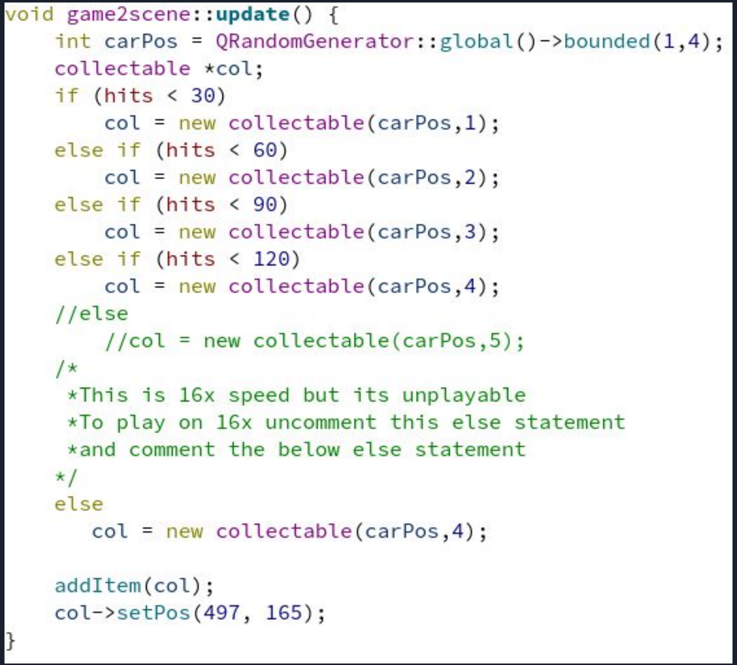
Game Functionalities

- The game starts showing the different scores of the different cars, the total score, lives left, and the highscore set by other users in the game hub
- The cars are spawned randomly and at a rate specified by game difficulty
- The game consists of 2 classes, one for the moving cars, and one for the whole scene
- At any time the player can mute the ambient music of the game by clicking the button on the top right

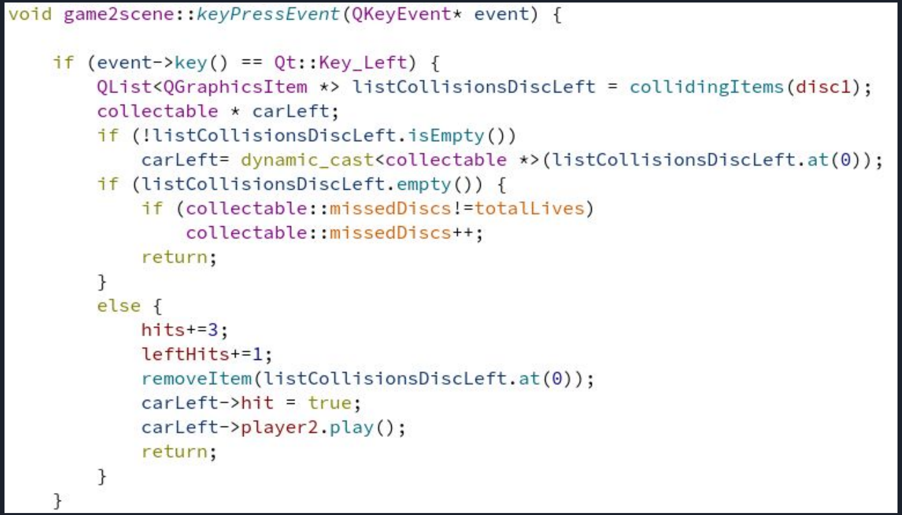


Important Functions

- A function that randomizes the car spawning and car speed. The position is randomly generated and the speed is incremented every 30 points. The difficulty of the game is changed when the timer on this slot is changed.



```cpp
void game2scene::update() {
    int carPos = QRandomGenerator::global()->bounded(1,4);
    collectable *col;
    if (hits < 30)
        col = new collectable(carPos,1);
    else if (hits < 60)
        col = new collectable(carPos,2);
    else if (hits < 90)
        col = new collectable(carPos,3);
    else if (hits < 120)
        col = new collectable(carPos,4);
    //else
        //col = new collectable(carPos,5);
    /*
     *This is 16x speed but its unplayable
     *To play on 16x uncomment this else statement
     *and comment the below else statement
    */
    else
       col = new collectable(carPos,4);

    addItem(col);
    col->setPos(497, 165);
}
```

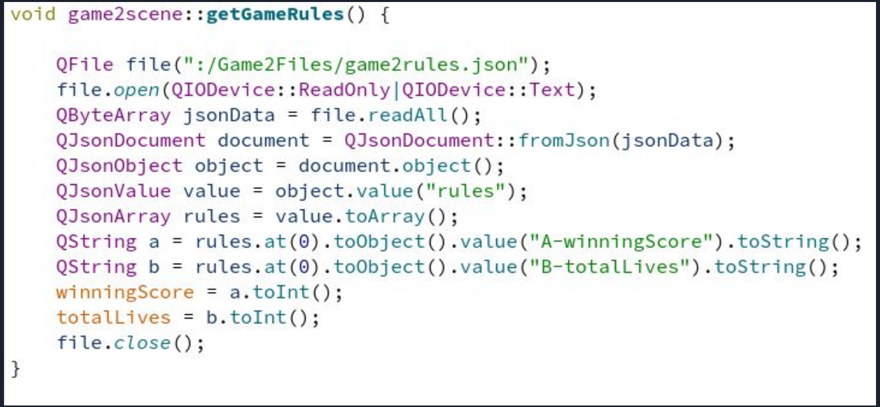


```cpp
void game2scene::keyPressEvent(QKeyEvent* event) {

    if (event->key() == Qt::Key_Left) {
        QList<QGraphicsItem *> listCollisionsDiscLeft = collidingItems(disc1);
        collectable * carLeft;
        if (!listCollisionsDiscLeft.isEmpty())
            carLeft= dynamic_cast<collectable *>(listCollisionsDiscLeft.at(0));
        if (listCollisionsDiscLeft.empty()) {
            if (collectable::missedDiscs!=totalLives)
                collectable::missedDiscs++;
            return;
        }
        else {
            hits+=3;
            leftHits+=1;
            removeItem(listCollisionsDiscLeft.at(0));
            carLeft->hit = true;
            carLeft->player2.play();
            return;
        }
    }
```

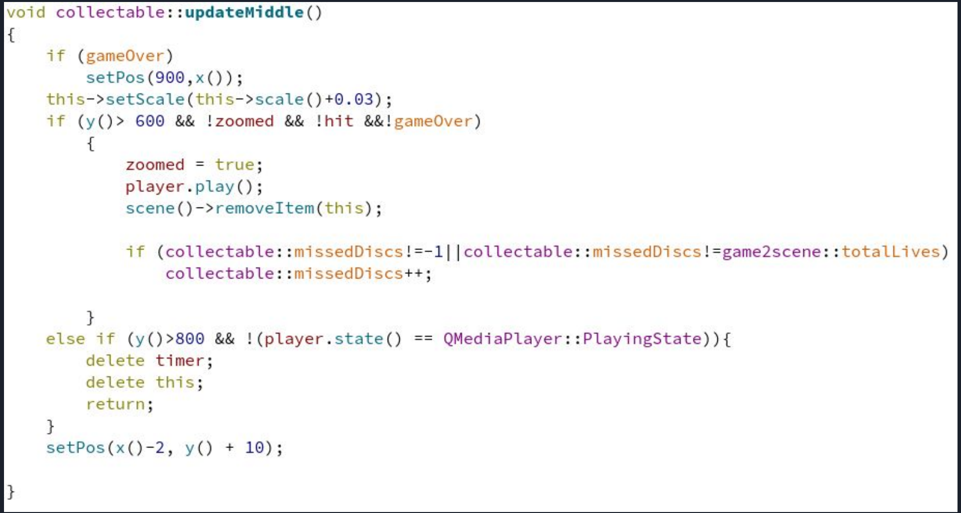
- When the car is colliding with the black circle and the player clicks, the scores update and the collectable is deleted, or the player loses a life

Important Functions (2)

- A function that fetches the game rules from the JSON file. These rules include the total winning score that a player must achieve to win and total lives that can be lost before losing the game



```
void game2scene::getGameRules() {

    QFile file(":/Game2Files/game2rules.json");
    file.open(QIODevice::ReadOnly|QIODevice::Text);
    QByteArray jsonData = file.readAll();
    QJsonDocument document = QJsonDocument::fromJson(jsonData);
    QJsonObject object = document.object();
    QJsonValue value = object.value("rules");
    QJsonArray rules = value.toArray();
    QString a = rules.at(0).toObject().value("A-winningScore").toString();
    QString b = rules.at(0).toObject().value("B-totalLives").toString();
    winningScore = a.toInt();
    totalLives = b.toInt();
    file.close();
}
```

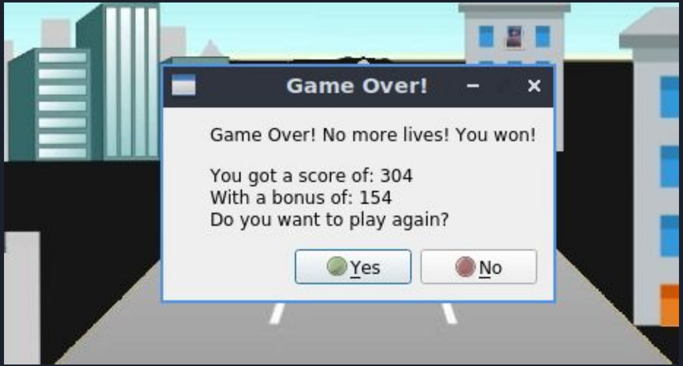


```
void collectable::updateMiddle()
{
    if (gameOver)
        setPos(900,x());
    this->setScale(this->scale()+0.03);
    if (y()> 600 && !zoomed && !hit &&!gameOver)
        {
            zoomed = true;
            player.play();
            scene()->removeItem(this);

            if (collectable::missedDiscs!=-1||collectable::missedDiscs!=game2scene::totalLives)
                collectable::missedDiscs++;

        }
    else if (y()>800 && !(player.state() == QMediaPlayer::PlayingState)){
        delete timer;
        delete this;
        return;
    }
    setPos(x()-2, y() + 10);

}
```

- One of three update functions for the cars (depending on position). This function moves and scales the car to emulate 3D movement
- The car gets deleted when the item is out of scope to avoid memory leaks.

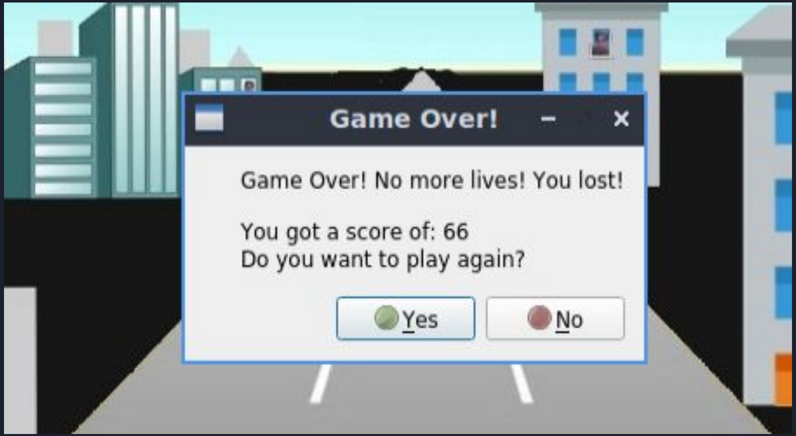
Game Over

- They win when the score is more than the set winning score. Bonus points are saved separately


Game Over!

Game Over! No more lives! You won!

You got a score of: 304
With a bonus of: 154
Do you want to play again?


Game Over!

Game Over! No more lives! You lost!

You got a score of: 66
Do you want to play again?

- They lose when the lives are 0 and the score is less than the set winning score