# Design Documentation

This is a detailed design of all the classes using a program. Here the pseudocode is used for the representation of all our code. State diagrams to represent the behavior of all of the classes are included and the overall class diagram to describe the overall interaction between classes is also included.

## Finite State Machine Diagrams

**AI Class State Diagram**

/home/shiro/Documents/A-Team/projectDocumentation/Finite state machine diagrams/Images/A.I. class.pngA.I. class

* Initiates the Board.
* The game has the player choose which mode to play: Player vs Player, (guest vs. guest) and Player vs AI (Guest vs AI).
* User selects the level of difficulty (easy, medium, and hard) if AI mode is chosen.
* Game prompts the user to choose tile and creates an X or O.
* The player is prompted to select who goes first.
* Initiate the players turn.
  + Program starts the game
  + Game paints an X or O on the game board.
  + Game subtracts from the counter
  + Game updates the score
* User selects tile if the player vs. Player mode is selected
* Once the tile is selected, it is disabled from being selected again and user is prompted to select who goes first.
  + The program starts the game
  + The game paints an X or O on the game board.
  + Game subtracts the counter.
  + Game updates the score
* Board is filled up and program ends the game
  + Game checks for a winner
  + Game updates statistics
* User is prompted to start another game
  + User selects to start new game or to quit and close current window.

Guest 

* User one is on guest window menu
  + Game asks user to choose game mode
  + User selects to either play against AI or another player
* User selects to play against a second player
  + - Game asks the second user to either play as guest or to login.
    - User selects to play as guest and enters username “guest”
    - User selects to play as registered player and enters their username and password.
* If the player needs help, they can click the help button and it will display instructions on how to continue.
  + - User will press OK and it will be returned to the play mode option
    - User can choose the change the game mode and it will exit to the previous menu.
* Once the game has started, the player can choose to:
  + - Quit the game by clicking the Quit Game button
    - User will be prompted with a confirmation window
      * If the player clicks no, it will return to where they left off in the game
      * If the player clicks yes, it will return to the main menu
    - Continue and finish the game

**Login State Diagram**



* From the main menu, the player clicks login
  + - Game asks for the username and password
* The player can click the Help button
  + - It will display instructions to further aid the player
    - They click OK and will be returned to the username and password window
* The player clicks submit
  + - Login successful
      * They click OK and it returns them to the main menu
* From the main menu, the player clicks quit
  + User will be prompted with a confirmation window
    - If the player clicks no, it will return to the main menu
    - If the player clicks yes, it will exit out of the game

**Player Vs. AI State Diagram**

Player vs Ai. (2)

* Player vs AI mode
* The player selects the difficulty they want
* The user is prompted to choose between X or O
  + User can click Help button
  + Program will display instructions to further aid the player
  + User clicks OK and is returned to the selection of X or O
* Once the user selects the stone they like, they are prompted again to choose who goes first
  + User selects who goes first
* Once the player is selected, the game starts and the player can choose to:
  + Quit the game by clicking the Quit Game button
  + User will be prompted with a confirmation window
    - If the player clicks no, it will return to where they left off in the game
    - If the player clicks yes, it will return to the main menu
* Continue and finish the game
* The player can logout
  + game will end the session
* Return to the main menu
* The player can click on the exit button
  + - * They will be prompted with a confirmation window
        + If the player clicks no, it will return to where they left
        + If the player clicks yes, it will return to the main menu

Player vs. player  (3)

* Player vs Player mode
  + They can click the Help button
    - It will display instructions to further aid the player
      * They will click OK and are returned to the Player vs Player mode
  + Once the player makes the Player vs Player selection, they are prompted to choose which player goes first
    - They can click the Help button
      * It will display instructions to further aid the player
        + They will click OK and are returned to select which player goes first
  + When they select which player goes first, the game starts
    - They can click Help button
      * It will display instructions to further aid the player
        + They will click OK and are returned to the game
  + Once the player is selected, the game starts and the player can choose to:
    - Quit the game by clicking the Quit Game button
      * They will be prompted with a confirmation window
        + If the player clicks no, it will return to where they left off in the game
        + If the player clicks yes, it will return to the main menu
      * Continue and finish the game
  + The player can click on the exit button
    - They will be prompted with a confirmation window
      * If the player clicks no, it will return to where they left off
      * If the player clicks yes, it will return to the main menu

Register

* Main menu
  + The user clicks the Sign-Up button
    - If the user clicks exit, they are returned to the main menu
      * It asks for the user’s information, such as
        + Username
        + Password
        + First name
        + Last name
        + Security question
        + Security answer
      * They can click the Help button
        + It will display instructions to further aid the player

They will click OK and are returned to the sign-up window

* + - * The player submits their information
        + If there is an error in the user’s information, it will display Unsuccessful Registration
        + If the information is correct, it will display Successful Registration
      * The user clicks OK and is returned to the main menu
  + The user clicks Quit
    - They will be prompted with a confirmation window
      * If the player clicks no, it will return to the main menu
      * If the player clicks yes, it will exit the game

Reset password

* Main menu
  + The player selects reset password
    - The player can click quit and they will be returned to the main menu
      * They click on forgot username
        + The user will enter their security answer

When they click quit, they will be returned to enter the username information

They can click the Help button

It will display instructions to further aid the player

They will click OK and are returned to the username information window

* + - * They enter their username information
        + They can click the Help button

It will display instructions to further aid the player

They will click OK and are returned to the username information window

* + - * + Click the Submit button
        + It resets their password successfully
        + Returns the user to the main menu
  + The user clicks Quit

**Start Game**

Start game

* The player starts a game
  + The first player makes a move
    - It creates a stone and disables the tile so the player can’t create another stone in its place.
      * They can click the Help button
        + It will display instructions to further aid the player

They will click OK and are returned to the game

* + - It updates and displays the current score after every move
    - It waits for the second player to make a move
      * They player can click exit
        + If they click yes, it will abort the game
        + If they click no, it will return to waiting for the next move
        + They can click the Help button

It will display instructions to further aid the player

They will click OK and are returned to the game

* + - Player two makes a move
  + The player clicks Exit game
    - They will be prompted with a confirmation window
      * If the player clicks no, it will return to the main menu
      * If the player clicks yes, it will return to waiting for the next move

## Pseudo Codes

### Pseudo Code for ResetPassword.cpp

**resetpassword.cpp**

#include “resetpassword.h”

#include “ui\_resetpassword.h”

#include <QmessageBox>

//Constructor

resetPassword::resetPassword(QWidget \*parent){

create a user interface ibject

create instances of widgets described in ui file

new ui:: resetPassword

}

//Deconstructor

resetPassword::~resetPassword()

{ delete ui; }

resetPassword::resetSumbitButton(){

/\*Submit button for resetting password and declare database and set parameters\*/

Qsql instance for representing connection

call static addDatabase() function and specify driver (“QMySQL”)

//setting attributes

setHostname

setDatabaseName

setUserName

setPassword

open connection

//Now most information collected will be used to reset password

firstName =resetFirstName

lastName=resetLastName

userName=resetUserName

if(connection fails){

Display message QmessageBox

return

}

else query to get information

search for realLastName, realQuestion, realAnswer;

if(searching fails)

Display error message ;

else {

reset information

realFirstName=searching

realLastName=searching

realQuestion=searching

realAnswer=searching

}

hold provided answer

compare to realAnswer

if(not matching)

{

Display error message(“Answers don't match);

return;

}

else

{ hold newPassword and retypedPassword

Update password to user inputs

Compare if they are the same password

while(not matching)

Display error message(“passwords don't match')

Hold the new entered password again

Compare again

}

Update password

if(update if successful)

{

Display message(“Password Update”)

close connection

}

else Display error message ;

}

resetPassword::resetHelpButton(){

MessageBox onHelp

give instructions

}

### Pseudo Code for selectgamemode.cpp

**selectgamemode.cpp**

#include “selectgamemode.h”

#include “ui\_selectgamemode.h”

#include “difficultylevel.h”

//Constructor

selectGameMode::selectGameMode (Qwidget \*parent) :

Create a selectGameMode user interface object

Create instances of widgets described in ui file

//Destructor

selectGameMdode::~selectGameMode()

{ delete ui ; }

selectGameMode::okButton(){

if(no mode is selected)

{ Display message(“Select game mode”);

else if (mode was selected)

{ Select a difficulty level now

difficultyLevel selectLevel

close ()

}

else call gameBoard

startGame

}

selectGameMode::BackButton(){

close current window

### Pseudo Code for gamemode.cpp

**gamemode.cpp**

#include "gamemode.h"

#include "ui\_gamemode.h"

#include "gameoption.h"

#include "difficultylevel.h"

#include "playergameoptions.h"

#include <QMessageBox>

#include <QLineEdit>

#include <QDebug>

gameMode::gameMode(QWidget \*parent) : QDialog(parent), ui(new Ui::gameMode)

{

ui->setupUi(this);

}

gameMode::~gameMode() //destructor

{

delete ui;

}

void gameMode :: switchingMode ()

{

}

void gameMode::on\_GameOptionMode\_clicked()

{

//this is going to determine either player vs player

//or player vs AI

//using a combobox to select choice

switchingMode ();

//ui->GamePlayMode->itemIcon(2);

if(ui->GamePlayMode->currentIndex() == 0)

{

QMessageBox :: information(this,tr("Make a choice"),tr("Please select player vs player or Player vs A.I"));

}

else if(ui->GamePlayMode->currentIndex() == 1)

{

//calls the player vs player function

PlayerGameOptions playerVsPlayer;

playerVsPlayer.setModal(true);

playerVsPlayer.exec();

close();

}

else if (ui->GamePlayMode->currentIndex() == 2)

{

//for player vs a.i.

//prompts user for difficulty of a.i.

difficultyLevel myLevel;

myLevel.setModal(true);

myLevel.exec();

close();

}

else

{

//an exception handler for combobox

QMessageBox :: information(this,tr("Make a choice"),tr("Please select player vs player or Player vs A.I"));

}

}

### Pseudo Code for aiclass.cpp

**aiclass.cpp**

#include "aiclass.h"

#include "time.h"

#include <QInputDialog>

#include "string.h"

int takingTurns; //global variables for board and turn

int AiLevel = 0;

bool AiTurn = false, callingEndGame= false;

int numbOfSquaresLeft =36;

int p1Score = 0;

int p2Score = 0;

QString username,username2;

AiClass \* board[6][6];

QGraphicsView \* myView;

QGraphicsScene \* myScene;

QGraphicsTextItem \* boardLabel;

void AiClass::AiBoard()

{

//now this is going to design the board with some menu on the board

myScene = new QGraphicsScene ();

myView = new QGraphicsView (myScene);

secondUserInformation(username2);

updatingScoreBoard(myScene);

myView->showMaximized();

int left=100, right=100, up=100, down=100;

//for loop to initialze boar

for (int x=0; x< 6; x++)

{

for(int y = 0; y < 6; y++) //changed all y

{

//now drawing the board by using QGraphicsRectItem

board[x][y] = new AiClass ();

board[x][y]->setPen(QPen(QColor("black"),5));

board[x][y]->setRect(left,right,up,down);

myScene->addItem(board[x][y]);

left+=100;

if(y==5)

{

left=100;

right+=100;

}

}

}

//showing the board and seeding the random generator

myView->show();

srand(time(NULL));

}

//now doing the AI mode

void AiClass :: easyAiMode ()

{

// will be used to generate a random colom and row

int col, row;

col=rand()%6;

row=rand()%6;

if(board[col][row]->isEnabled() == false)

{

//this means if the bord is dissable, i will be repeating the guess

easyAiMode();

}

else

{

//not disable, then i can play it as my board

board[col][row]->playEvent();

AiTurn = false;

}

}

//now doing medium AI mode

void AiClass :: mediumAiMode()

{

//this is the medium Ai Mode

//by using a randomized alrogirthm

//have to make it mor structure and respond towards players move

//int see = rand()%2;

//if (see == 1)

// easyAiMode();

//else

// hardAiMode();

int col, row;

col=rand()%6;

row=rand()%6;

if(board[col][row]->isEnabled() == false)

{

//this means if the bord is dissable, i will be repeating the guess

mediumAiMode();

}

else

{

board[col][row]->playEvent();

AiTurn = false;

}

}

void AiClass :: hardAiMode()

{

/\*int count = 0;

int phx = -1;

int phy=-1;

int bestMovex, bestMovey;

for (int x=0; x < 6; x++) //looks at the board to find any moves

{

for (int y= 0; y < 6; y++)

{

if (board[x][y]->data(takingTurns) == 1 || board[x][y]->data(takingTurns) == -1)

{

count++;

phx=x;

phy=y;

}

}

}

if (count < 1) // random move in center 9 square

{ // if no moves have been made

centerNine = new List<int>(){6,7,8,11,12,13,16,17,18};

if (ph > 0)

centerNine.Remove(ph);

bestMove = centerNine.ElementAt(random.Next(0, centerNine.Count-1));

Debug.WriteLine("Random move at " + bestMove);

}

else

{ //copies the game board

for (int i = 0; i < 25; i++)

{

if (String.Equals(board[i].Content, "X"))

{

game[i] = "X";

}

else if (String.Equals(board[i].Content, "O"))

{

game[i] = "O";

}

else

{

game[i] = "";

}

}

depth = 0; //sets depth to 0

bestMove = MinMax(game);

}

board[bestMovex][bestMovey]->playEvent();\*/

}

void AiClass::settingAiLevel(int level)

{

//this funciton is used to design a level

AiLevel = level;

}

void AiClass::mousePressEvent(QGraphicsSceneMouseEvent \*event)

{

qDebug() << AiLevel;

if(AiTurn == false)

{

if(event->button() == Qt::LeftButton)

{

this->playEvent();

}

}

AiTurn = true;

if(AiTurn == true && AiLevel == 3)

{

this->hardAiMode();

}

else if(AiTurn == true && AiLevel == 2)

{

this->mediumAiMode();

}

else if(AiTurn == true && AiLevel == 1)

{

this->easyAiMode();

}

else

AiTurn = false;

}

/\*

void AiClass::mouseReleaseEvent(QGraphicsSceneMouseEvent \*event)

{

if(AiTurn == true && AiLevel == 3)

this->easyAiMode();

else if(AiTurn == true && AiLevel == 2)

this->mediumAiMode();

else if(AiTurn == true && AiLevel == 1)

this->easyAiMode();

<<<<<<< HEAD

else

AiTurn = false;

}

=======

}

else{AiTurn = false;}

}\*/

void AiClass :: playEvent()

{

if(takingTurns == 1)

{

this->setBrush(QPixmap(":/images/X.png"));

}

else

{

this->setBrush(QPixmap(":/images/O\_file.png"));

}

this->setData(takingTurns,QVariant(takingTurns));

this->setEnabled(false);

myScene->removeItem(boardLabel);

checkScore();

updatingScoreBoard(myScene);

takingTurns \*=-1;

numbOfSquaresLeft--;

if(numbOfSquaresLeft == 0)

{

QMessageBox endGame;

qDebug () << username << ": "<< p1Score;

qDebug () << username2 << ": "<< p2Score;

if(p1Score > p2Score)

{

endGame.setInformativeText(username+" wins "+QString::number(p1Score));

endGame.exec();

callingEndGame=true;

}

else if (p1Score < p2Score)

{

endGame.setInformativeText(username2+" wins "+QString::number(p2Score));

endGame.exec();

callingEndGame=true;

}

else

{

endGame.setInformativeText("Tie Game!");

endGame.exec();

callingEndGame=true;

}

if(callingEndGame)

newGame(myView);

}

}

void AiClass::checkScore()

{

int x, y;

int col = 1;

int row = 1;

int diag = 1;

int diag2 = 1; //initiated to 1 since there is always the one just placed

int score;

int player = takingTurns;

int startingx, staringy, endingx, endingy;

for(int i = 0; i < 6; i++)

{

for(int j = 0; j<6; j++)

{

if(board[i][j] == this)

{

x = i;

y = j;

}

}

}

if(takingTurns == 1)

score = p1Score;

else

score = p2Score;

//column

for(int i = 1; i < 4; i++)

{

if(y+i > 5)

break;

if (board[x][y+i]->data(takingTurns).toInt() == player)

{

col++;

if(col==2)

{

startingx=x;

staringy=y;

}

}

else

break;

}

for(int i = 1; i < 4; i++)

{

if(y-i < 0)

break;

if (board[x][y-i]->data(takingTurns).toInt() == player)

{

col++;

endingx =x;

endingy=y;

}

else

break;

}

if (col >=4)

{

score += (col - 3);

qDebug () << x << " " << y;

}

//row

for(int i = 1; i < 4; i++)

{

if(x+i > 5)

break;

if (board[x+i][y]->data(takingTurns).toInt() == player)

{

row++;

qDebug () << x << " " << y << endl;

}

else

break;

}

for(int i = 1; i < 4; i++)

{

if(x-i < 0)

break;

if (board[x-i][y]->data(takingTurns).toInt() == player)

{

row++;

qDebug () << x << " " << y;

}

else

break;

}

if (row >= 4)

score += (row - 3);

//diag '\'

for(int i = 1; i < 4; i++)

{

if(y+i > 5 || x+i > 5)

break;

if (board[x+i][y+i]->data(takingTurns).toInt() == player)

{

diag++;

qDebug () << x << " " << y << endl;

}

else

break;

}

for(int i = 1; i < 4; i++)

{

if(y-i < 0 || x-i < 0)

break;

if (board[x-i][y-i]->data(takingTurns).toInt() == player)

{

diag++;

qDebug () << x << " " << y << endl;

}

else

break;

}

if (diag >= 4)

score += (diag - 3);

//diag2 /

for(int i = 1; i < 4; i++)

{

if(y-i < 0 || x+i > 5)

break;

if (board[x+i][y-i]->data(takingTurns).toInt() == player)

{

diag2++;

qDebug () << x << " " << y << endl;

}

else

break;

}

for(int i = 1; i < 4; i++)

{

if(y+i >5 || x-i < 0)

break;

if (board[x-i][y+i]->data(takingTurns).toInt() == player)

{

diag2++;

qDebug () << x << " " << y << endl;

}

else

break;

}

if (diag2 >= 4)

score += (diag2 - 3);

if(takingTurns == 1)

p1Score = score;

else

p2Score = score;

}

void AiClass:: drawingEvent (int left, int right,int up, int down)

{

//this is going to draw on the board

left=0,right=0,up=0,down=0;

QGraphicsLineItem \* newPainting = new QGraphicsLineItem ();

newPainting->setPen(QPen(QColor("Green"),20));

newPainting->setLine(110,150,490,150);

//vertical

//newPainting->setLine(150,110,150,490);

//diagonal

//newPainting->setLine(100,100,500,500);

}

void AiClass::updatingScoreBoard(QGraphicsScene \*)

{

//now let me check the AI level and see if it has been changed

//this function updates the score board

boardLabel = new QGraphicsTextItem;

boardLabel->setPlainText("Username: "+username+"\t "+"Score: "+QString::number(p1Score)+

"\t\t"+"Username2: "+username2+"\t"+"Score: "+QString::number(p2Score)+

"\t\t"+"Ai Level: "+QString::number(AiLevel)

);

boardLabel->setFont(QFont("Times"));

myScene->addItem(boardLabel);

}

QString AiClass :: secondUserInformation(QString secondUsername)

{

QString secondPassword;

QInputDialog secondUsernamePrompt, secondePasswordPrompt;

if(AiLevel == 0 && username.isEmpty() == true)

{

username="Guest";

secondUsernamePrompt.setLabelText("Enter username");

secondUsernamePrompt.exec();

secondUsername = secondUsernamePrompt.textValue();

if(secondUsername.toLower() == "guest")

{

username2="Guest";

return username2;

}

if(username2.toLower() !="guest")

{

secondePasswordPrompt.setLabelText("Enter your password");

secondePasswordPrompt.exec();

secondPassword=secondePasswordPrompt.textValue();

username2=secondUsername;

//now using the database in order to querry you

}

}

else if(AiLevel == 0)

{

//meaning username is not empty

secondUsernamePrompt.setLabelText("enter username");

secondUsernamePrompt.exec();

secondUsername = secondUsernamePrompt.textValue();

if(secondUsername.toLower() == "guest")

{

username2="Guest";

return username2;

//meaning username2 is going to play as guest

}

else

{

secondePasswordPrompt.setLabelText("Enter your password");

secondePasswordPrompt.exec();

secondPassword=secondePasswordPrompt.textValue();

username2=secondUsername;

//query the database

if (!secondUserLogin(secondUsername,secondPassword))

secondUserInformation(secondUsername);

else

username2=secondUsername;

return username2;

}

}

else

username2 = "A.I";

return username2;

}

void AiClass :: settingTurn(int turn)

{

//this will change the global variable of the turn

takingTurns = turn;

}

QString AiClass::settingUsername(QString myUsername)

{

username=myUsername;

return username;

}

bool AiClass::secondUserLogin(QString secondPass,QString secondUser)

{

//this is for login in as a second user

bool successStatus = false;

QSqlDatabase db = QSqlDatabase :: addDatabase("QMYSQL"); //driver of database

db.setHostName("localhost");

db.setDatabaseName("tictactoe");

db.setUserName("root");

db.setPassword("Amatarasu76");

db.setPort(3306);

bool connectionAttemps = db.open();

if(!connectionAttemps)

{

//failure to connect to database

QMessageBox errorMessage;

errorMessage.setInformativeText("failed to load");

errorMessage.exec();

return successStatus;

}

else

{

//sucessful connection

//checking if user exsists in database

QSqlQuery myQuery;

myQuery.prepare("SELECT `userName`, `password` FROM `players` WHERE userName = ?"); //searching for user

myQuery.bindValue(0,secondUser);

myQuery.exec();

//string for username and password in the user database

QString realUsername, realPassword;

if(myQuery.next())

{

realUsername = myQuery.value(0).toString();

realPassword = myQuery.value(1).toString();

}

//now comparing inputted username and password with database entries

int x=QString :: compare(realUsername,secondUser); //comparing username

int y=QString :: compare(secondPass,realPassword); //comparing password

//if username or password do not match in database entries

if(x!=0 || y!=0)

{

//display error message

QMessageBox wrongUser;

wrongUser.setInformativeText("Wrong Username or Password");

wrongUser.exec();

successStatus=false;

}

else

{

QMessageBox goodUser;

goodUser.setInformativeText("Welcome "+secondUser);

goodUser.exec();

successStatus=true;

}

}

return successStatus;

}

void AiClass::newGame(QGraphicsView \* myView)

{

/\*when this function is called,

\* it will determine start a new board,

\* allow you to choose the level once more

\* or to play as player against another play\*/

QMessageBox msgBox;

msgBox.setWindowTitle("New Game or Quit");

msgBox.setText("Would you like to start a new game or Quit?");

msgBox.setStandardButtons(QMessageBox::Yes);

msgBox.addButton(QMessageBox::No);

msgBox.setDefaultButton(QMessageBox::Yes);

if(msgBox.exec() == QMessageBox::Yes)

{

myView->close();

p1Score=p2Score=0;

delete myView;

numbOfSquaresLeft=36;

gameMode startingNewGame;

startingNewGame.setModal(true);

startingNewGame.exec();

}

else

{

//delete the game send them to the

myView->close();

delete myView;

numbOfSquaresLeft=36;

//delete [] board;

}

}

### Pseudo Code for AIClass.cpp

**AIClass.cpp**

#include "AIClass.h"

#include "ui\_AIClass.h"

#include <time.h>

#include <QGraphicsScene>

#include <QGraphicsItem>

#include <QGraphicsRectItem>

#include <QGraphicsView>

#include <QDebug>

#include <QPixmap>

#include <QMouseEvent>

#include <QMenuBar>

#include <QMessageBox>

int squaresLeft=36, turn = 1; //global variables for board and turn

QString Username;

CustomItem \* myBoard[6][6];

void CustomItem :: mousePressEvent(QGraphicsSceneMouseEvent \*event)

{

if(event->button() == Qt::LeftButton)

{

this->playEvent();

}

}

//mouse release event in order to make the

QString AIClass:: setUsername (QString username)

{

Username = username;

return username;

}

void CustomItem :: playEvent ()

{

if(turn== 1)

{

this->setBrush(QPixmap(":/images/X.png"));

}

else

{

this->setBrush(QPixmap(":/images/O\_file.png"));

}

this->setData(turn,QVariant::QVariant(turn));

this->setEnabled(false);

turn \*= -1;

squaresLeft=squaresLeft-1;

if(squaresLeft == 0)

{

QMessageBox \* endGame = new QMessageBox ();

endGame->setInformativeText("call the end Game");

endGame->show();

}

}

//now adding a menu content in our board

AIClass::AIClass(QWidget \*parent) : QDialog(parent), ui(new Ui::AIClass)

{

ui->setupUi(this);

this->setMouseTracking(true);

}

AIClass::~AIClass()

{

delete ui;

}

void AIClass::gameStart()

{

//now this is going to design the board with some menu on the board

QGraphicsScene \* myScene = new QGraphicsScene ();

QGraphicsView \* myView = new QGraphicsView (myScene);

//display, user ID, score, and amount of turns

myScene->addText("UserID: "+Username+" \t\tScore: "+"\t\tTurns: ")->mapToScene(0.0,0.0).toPoint();

myView->setFixedSize(800,900);

myView->setMouseTracking(true);

//build the array of board

//pixel sizes of the four dimensions

int left=100, right=100, up=100, down=100;

//for loop to initialze boar

for (int x=0; x< 6; x++)

{

for(int y = 0; y < 6; y++) //changed all y

{

//now drawing the board by using QGraphicsRectItem

myBoard[x][y] = new CustomItem ();

myBoard[x][y]->setRect(left,right,up,down);

myScene->addItem(myBoard[x][y]);

left+=100;

if(y==5)

{

left=100;

right+=100;

}

}

}

//display board

myView->show();

}

void gameBoard :: settingTurn()

{

//this will change the global variable of the turn

turn = -1;

}

### Pseudo Code for AIClass.cpp

#include "allheadertoinclude.h"

using namespace std;

//gobla qt variable for setting scenes, usernames, and graphic items

QString username,username2;

QGraphicsView \* myView;

QGraphicsScene \* myScene;

QGraphicsTextItem \* boardLabel;

AiClass \* board[6][6];

int takingTurns;

int AiLevel = 0;

bool AiTurn = false, callingEndGame= false;

int numbOfSquaresLeft =36;

int p1Score = 0;

int p2Score = 0;

int MIN\_VALUE = std::numeric\_limits<int>::min(); // minimum value

int MAX\_VALUE = std::numeric\_limits<int>::max();

int maxDepth = 5;

int aWeight = -1;

int pWeight = 1;

int b[6][6];

int MVP1[] = {15, 16, 21, 22, 8, 11, 26, 29};

struct AiClass::Point{

int x;

int y;

} MVP[8];

struct AiClass::pAndS{

int score;

Point p;

};

vector<AiClass::pAndS> leafScores;

void AiClass::AiBoard(){

//now this is going to design the board with some menu on the board

Setting initial scre to 0

Counter set to 36 to keep track of scores

Create a new scene

Request for second user information

Call function to update the sore board getting info from my scene

int left=100, right=100, up=100, down=100;

//for loop to initialize board

for (int x=0; x< 6; x++)

{

for(int y = 0; y < 6; y++)

{

//now drawing the board by using QGraphicsRectItem

board[x][y] = new AiClass ()

;

boardposition->setPen;

boardposition->setRect

myScene->addItem (board to sce

left+=100;

}

}

//showing the board and seeding the random generator

Show board

Initialize coordinates with random numbers

for(int i = 0; i < 8; i++)

{

Array[i].y = Array[i]%6-1;//col

Array[i].x = Array[i]/6;//row

if(AiTurn == true)

//if AI goes first show AI first move

{

aiMovesFirst = new QGraphicsSceneMouseEvent ();

AI moves first

}

}

easyAiMode (){

//Randomly places token

// will be used to generate a random colom and row

int col, row;

col=rand()%6;

row=rand()%6;

if(chosen position is already taken){//If already chosen, will guess again.

easyAiMode();

}

else{

//If available, play.

playEvent();

Disabble tile

}

}

mediumAiMode(){

maxDepth = 4;

hardAiMode();

}

hardAiMode(){

while(AiTurn){

// bool mvp = false;

for(int i = 0; i < 4; i++){

if(chosen tile->can be played){

choose->play that position);

Set AITurn to false

break;

}

}

if(board psition -1 && AiTurn && board position == -1 && AiTurn){

if(board position == -1 && AiTurn && board position->an be played){

atBoard position->play that position;

AiTurn = false;

break;

}

if(boardposition->can be played){

boardpostion->playEvent();

AiTurn = false;

break;

}

}

if(board position == -1 && is AiTurn && positon == -1 && is AiTurn){

if(b[3][1] == -1 && AiTurn && board[3][4]->can be played){

boardPosition]->playEvent();

AiTurn = false;

break;

}

if(boardPosiotn->isEnabled() == can be played){

board[3][1]->playEvent();

AiTurn = false;

break;

}

}

if(board position == -1 && AiTurn && board position== -1 && AiTurn){

if(b[1][2] == -1 && AiTurn && board[4][2]->can be played){

board[4][2]->play that position;

AiTurn = false;

break;

}

if(boardposition->can be playes){

boardposition->play that position;

AiTurn = false;

break;

}

}

if(board position == -1 && AiTurn && board position == -1 && AiTurn){

if(b[1][3] == -1 && AiTurn && board[4][3]->can be played){

board[4][3]->play event;

AiTurn = false;

break;

}

if(boardposition->can be played){

boardposition->play that position;

AiTurn = false;

break;

}

}

if(board position1 && AiTurn && boardposition == -1 && AiTurn){

if(bposition == -1 && AiTurn && boardposition->can be chosen){

Board position->play that position;

AiTurn = false;

break;

}

if(boardposition->can be played){

Board position->play that position

AiTurn = false;

break;

}

}

if(board position == -1 && AiTurn && board position== -1 && AiTurn){

if(board position== -1 && AiTurn && board postion-> == true){

Board position>play that position);

AiTurn = false;

break;

}

if(board position-> chosen {

Board position->play that position ;

AiTurn = false;

break;

}

}

if(AiTurn){

for(int i = 4; i < 8; i++){

if(board[MVP[i].x][MVP[i].y]->isEnabled() == true){

board[MVP[i].x][MVP[i].y]->playEvent();

AiTurn = false;

break;

}

}

}

if(AiTurn){

minmax(MIN\_VALUE, MAX\_VALUE, 0, -1, 0);

Point p = best();// chooses best postion

if(board position->can be played == true){

Board position = -1;

Board position->play that position;

AiTurn = false;

}

settingAiLevel(int level){

//this funciton is used to design a level

AiLevel = level;

PlayerGameOptions::PlayerGameOptions(QWidget \*parent) : QDialog(parent), ui(new Ui::PlayerGameOptions)

{

Set UI for game option

}

PlayerGameOptions::~PlayerGameOptions()

{

delete UI;

}

void PlayerGameOptions::on\_pushButton\_clicked()

{

//allows user to choose either X or O

if(ui->playerGameOption->currentIndex()== 0)

{

//if default item is chosen in combo box

QMessageBox :: information(this,tr("Game Information"),tr("Choose between X and O"));

}

else if(ui->playerGameOption->currentIndex() == 1)

{

//starts the game with player 1 as X and player2 as O

Initialize board

Start game

close();

}

else if(ui->playerGameOption->currentIndex() == 2)

{

//starts the game with player 1 as O and player2 as X

Initialize Board

Setting Turn

Initialize game Star

close();

}

else

{

//Exception handler

QMessageBox :: information(this,tr("Game Information"),tr("Choose between X and O"));

}

}

void PlayerGameOptions::on\_playerOptionHelpButton\_clicked()

{

//Help button

QMessageBox :: information(this,tr("help button"),tr("Choose to go either as X or O"));

}

### Pseudo Code for difficultylevel.cpp

#include "difficultylevel.h"

#include "ui\_difficultylevel.h"

#include "gameoption.h"

#include "aiclass.h"

#include <QMessageBox>

#include <QDebug>

difficultyLevel::difficultyLevel(QWidget \*parent) :

QDialog(parent),

ui(new Ui::difficultyLevel)

{

Display UI for difficulty level

playGameButton is Enabled(false);

Play logOutButton->setEnabled(false);

}

difficultyLevel::~difficultyLevel()

{

delete ui;

}

void difficultyLevel::on\_mediumButton\_clicked()

{

//return 2; //for the meduim level of a.i.

if (ui->mediumButton->isChecked())

ui->playGameButton->setEnabled(true);

}

void difficultyLevel::on\_hardButton\_clicked()

{

//return 3; //for the hard level of a.i.

if(ui->hardButton->isChecked())

ui->playGameButton->setEnabled(true);

}

void difficultyLevel::on\_logOutButton\_clicked()

{

//this is the logout button

QMessageBox logoutMessage;

logoutMessage.setText("Thank you for playing "); //message for logout

logoutMessage.exec();

close();

}

void difficultyLevel::on\_exitButton\_clicked()

{

//close form

close();

}

void difficultyLevel::on\_difficultyHelpButton\_clicked()

{

//this is the help button

QMessageBox helpMessage;

helpMessage.setText("Please choose Easy, Medium or hard and press play"); //message

helpMessage.exec();

}

void difficultyLevel::on\_easyButton\_clicked()

{

//this is enabling the playGameButton

if(ui->easyButton->isChecked())

ui->playGameButton->setEnabled(true);

}

void difficultyLevel::on\_playGameButton\_clicked()

{

//now this button will call the game mode based on what which is selected and play

if(ui->hardButton->isChecked())

{

AiClass hardMode;

hardMode.settingAiLevel(3);

hardMode.hardAiMode();

close();

}

else if(ui->mediumButton->isChecked())

{

AiClass mediumMode;

mediumMode.settingAiLevel(2);

mediumMode.mediumAiMode();

close();

}

else

{

AiClass staringGame;

staringGame.AiBoard();

staringGame.settingAiLevel(1);

close();

}

### Pseudo Code for registrationScene.cpp

Constructor

registrationScene::registrationScene(QWidget \*parent){

Create a registration user interface object

Create instances of widgets described in Ui file

}

Destructor

registrationScene::~registrationScene(){

delete ui;

}

registrationScene::signUpButton(){

/\*Function to write SQL codes to connect to the database and check username, password, etc

make connection to a database (MySQL). \*/

setting hostName

setting databasaName

setting userName

settingPassword

Now QSQLDatabase class instance that represents a connection

call static addDatabase function

and specify driver (“QMYSQL”)

//Next check that the inputs gathered from ui math by using compare function

if(inputs don’t match)

Display QMessageBox with message error

else

send info to database

if (not Okay )

report an error occurred during connection to database

Message::critical(“Error”)

return;

else

assign values to signUpQuery;

if (signUpQuery is successful in exec())

Display QMessageBox (“Thank you for signing up”) ;

else Display another error of possible mismatch

QMessageBox(“Possible mismatch”)

close database connection

}

}

registrationScence::helpButton(){

//Provide help t user

QMessageBox(“Confirm password”)

}

### Pseudo Code for playergameoption.cpp

#include "playergameoptions.h"

#include "ui\_playergameoptions.h"

#include <QMessageBox>

#include "aiclass.h"

PlayerGameOptions::PlayerGameOptions(QWidget \*parent) : QDialog(parent), ui(new Ui::PlayerGameOptions)

{

ui->setupUi(this);

}

PlayerGameOptions::~PlayerGameOptions()

{

delete ui;

}

void PlayerGameOptions::on\_pushButton\_clicked()

{

//allows user to choose either X or O

if(ui->playerGameOption->currentIndex()== 0)

{

//if default item is chosen in combo box

QMessageBox :: information(this,tr("Game Information"),tr("Choose between X and O"));

}

else if(ui->playerGameOption->currentIndex() == 1)

{

//starts the game with player 1 as X and player2 as O

AiClass pvpGame;

pvpGame.settingAiLevel(0);

pvpGame.settingTurn(1);

pvpGame.AiBoard();

close();

}

else if(ui->playerGameOption->currentIndex() == 2)

{

//starts the game with player 1 as O and player2 as X

AiClass pvpGame;

pvpGame.settingTurn(-1);

pvpGame.settingAiLevel(0);

pvpGame.AiBoard();

close();

}

else

{

//Exception handler

QMessageBox :: information(this,tr("Game Information"),tr("Choose between X and O"));

}

}

void PlayerGameOptions::on\_playerOptionHelpButton\_clicked()

{

//Help button

QMessageBox :: information(this,tr("help button"),tr("Choose to go either as X or O"));

}

### Pseudo Code for maintictactoe.cpp

/\*constructor takes object of QWidget type as parameter for user interface

to receive mouse, keyboard an other events from window system.\*/

Constructor

{

if ( QWidget parameter parent is equal to 0, then widget will be a window)

{

QMainWindow(parent)

//Construct a QMainWindow with given parent

new Ui window mainTicTacToe

//Creating instances of widgets described in ui file

}

Destructor

{ delete ui}

mainTicTacToe :: helpButton(){

//inform of ask user

QMessageBox (“Choose option to play as guest or as user”)

}

mainTicTacToe::quitButton(){

QMessagebox: Gives option to quit

Standards button Yes or No

Check if (yes to quit )

call quit() function

else do nothing at that moment, keep the game running

}

mainTicTacToe::loginButton() {

//this function is going to create a new window or scene to bring username and //password

Dialog class is implemented to set modal to true

//now use database functionalities to connect to database

mainTicTacToe:: signUpButton()

// Bring up menu for signing up

Ask for information : username

password

re-enter password

sign in button

}

mainTicTacToe::playAsGuest(){

open game board

SelectGameMode and create object

show dialog

Create new object for gameOption()

}

mainTicTacToe::resetButton(){

//function to help reset password

}

### Pseudo Code for Main.cpp

#include <QApplication> // For managing the GUI applications control flow and main settings

//Only one Application object for any GUI application.

int main (int argc, char \* argv[]){ //for command line arguments

/\* Here we initialize the window system and construct and application object with argc command line arguments in argh.\*/

mainTicTacToe w; //create mainTicaTacToe object

call function show();

return execution to Application.

}

### Pseudo Code for LoginScene.cpp

#include “loginScene.h”

#include “ui.loginScene.h”

//Constructor

loginScene::loginScene(QWidget \*parent) :

Create a loginScene user interface object

Create instances of widgets described in ui file

//Destructor

loginScene::~loginScene()

{ delete ui; }

loginScene::helpButton(){

//This will show how to login or quit the screen

Display message box (“Enter username and password.....”)

Give instructions

}

loginScene::logginIn(){

//This is going to query to the database to check if user exists

Hold username and password

Assign user inputs

Add to database

Qsql instance representing connection

Call static addDatabase()

and specify driver (“QMYSQL”)

setHostname

setDatabaseName

setUserName

setPassword

setPort

Open connection

if(connection fails)

{

Display error message

return

}

else

{ Check if user exists

QsqlQuery prepare for execution

set placeholders and execute

Hold username and realPassword

while (there is records in result)

assign to realUsername

assign to realPassword

}

//Comparing now

Compare (realUsername and userName)

Compare (realPassword and password)

if(either one is not equal)

Display error message

}

else Display welcome message

close connection

}

### Pseudo Code for User.cpp

#include "user.h"

#include <stdlib.h>

#include <fstream>

#include <iostream>

#include <string>

#include <QDebug>

using namespace std;

string uName, pass, fName, lName, quest, ans;

User::User(){}

User::User(QString uName, QString pass, QString fName, QString lName, QString quest, QString ans){

info.uName = uName;

info.pass = pass;

info.fName = fName;

info.lName = lName;

info.quest = quest;

info.ans = ans;

info.wins = 0;

info.loss = 0;

info.ties = 0;

}

void User::save(){

uName = info.uName.toStdString();

pass = info.pass.toStdString();

fName = info.fName.toStdString();

lName = info.lName.toStdString();

quest = info.quest.toStdString();

ans = info.ans.toStdString();

encrypt();

ofstream ofs(("Users/" + info.uName.toStdString()).c\_str(), ios::binary);

ofs << uName << endl << pass << endl << fName << endl << lName << endl << quest

<< endl << ans << endl << info.wins << endl << info.loss << endl << info.ties;

}

void User::open(QString user){

ifstream ifs(("Users/" + user.toStdString()).c\_str(), ios::binary);

ifs >> uName >> pass >> fName >> lName >> quest >> ans >> info.wins >> info.loss >> info.ties;

encrypt();

info.uName = QString::fromStdString(uName);

info.pass = QString::fromStdString(pass);

info.fName = QString::fromStdString(fName);

info.lName = QString::fromStdString(lName);

info.quest = QString::fromStdString(quest);

info.ans = QString::fromStdString(ans);

}

void User::encrypt(){

string key = "Suck it Ibra!";

while (key.size() < uName.size())

key += key;

for (int i = 0; i < uName.size(); ++i)

uName[i] ^= key[i];

while (key.size() < pass.size())

key += key;

for (int i = 0; i < pass.size(); ++i)

pass[i] ^= key[i];

while (key.size() < fName.size())

key += key;

for (int i = 0; i < fName.size(); ++i)

fName[i] ^= key[i];

while (key.size() < lName.size())

key += key;

for (int i = 0; i < lName.size(); ++i)

lName[i] ^= key[i];

while (key.size() < quest.size())

key += key;

for (int i = 0; i < quest.size(); ++i)

quest[i] ^= key[i];

while (key.size() < ans.size())

key += key;

for (int i = 0; i < ans.size(); ++i)

ans[i] ^= key[i];

}