Non-functional testing

Inspection

* **Overview:**  The original pseudocode used for testing

**Pseudo Code for gameoption.cpp**

**gameoption.cpp**

#include “gameoption.h”

#include “ui\_gameoption.h”

#include “gameboard.h”

#include “difficulty llevel.h”

//Constructor

gameOption::gameOption(QWidget \*parent) :

Create a gameOption user interface object

Create instances of widgets described in Ui file

//Deconstructor

gameOption::~gameOption()

{ delete ui; }

gameOption::gameOptionOkay(){

if(user chooses to go first)

{create new board

start game

close}

else if(user chooses to go second)

{create a playerVsAi board

start game

close}

else Display message box (“Please select game mode”)}

**Pseudo Code for LoginScene.cpp**

**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 maintictactoe.cpp**

**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**

**Main.cpp** //Main implementation

#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 registrationScene.cpp**

**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 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}

* **Preparation:** Planned out what code needed fixing, such as:
  + Function names
  + Function calls
  + Pointer references
  + Pointer deletion
  + Final clean-up
* **Inspection:** Applied code clean-up to the original pseudocode.
* **Re-work:**

Code cleanup

gameboard.cpp

* Changed the second variable of the for loop to ‘y’ (line 93-109)
* Changed spacing in class ‘CustomItem’
* Deleted old comments (line 87)
* Voidgamestart()
  + Added comments explaining the board drawing

gameboard.h

* Cleaned up line spacing (line 10-12)
* Fixed class ‘gameBoard’ spacing

Difficultylevel.cpp

* Added comments on:
  + A.I. difficulties
  + Logout button
  + Exit button
  + ‘Play’ button

Difficultylevel.h

-Added spaces and comments to the function prototypes

gamemode.cpp

* Added comments on game options functions
  + Destructor
  + Added comments on all ‘if’ / ’else-if’ statements

gamemode.h

* Fixed class spacing

gameoption.cpp

* Explained the functionality of when player vs. A.I. is selected
  + Added comments to ‘if’ statements for A.I.

gameoption.h

* Fixed constructor spacing
* Fixed function prototype spacing

Loginscene.cpp

* Fixed constructor spacing
* Added login scene comments fixed spacing
  + Added comments on query
  + Added comments on database
  + Added comments on ‘if-else’ statements
* Added comments to database connection
  + Determines if user successfully logs in or not
  + Then opens the ui form for gamemode option

Loginscene.h

* Fixed constructor spacing, and constructor prototype

Main.cpp - No changes

Maintictactoe.cpp

* Deleted comments on constructor
* Fixed spacing and added comments on quit button function
* Login button function
  + fixed spacing and deleted old comments

maintictactoe.h

* Fixed constructor spacing
* Added comments on function prototypes
* Deleted old database functionality class

Playergamemdoe.cpp

* Fixed spacing on constructor
* Added comments to ‘if-else’ statements
* Added comment on help button
* Refactored slot for ‘playasGuest’

Playergamemode.h – fixed spacing and added comments on prototypes

Regristrationscene.cpp

* Fixed constructor spacing
* Sign up button slot functionality
  + Fixed comments on ‘if-else’
  + Fixed spacing

Registrationscene.h – fixed spacing and added comments on function prototypes

Resetpassword.cpp

* Added comments to the ‘if-else’ statements
  + Fixed spacing in these statements

Resetpasssword.h

* Fixed constructor spacing
* **Follow up:** Code clean-up has been applied to the project.