**Testing Plan and Implementation**

Table of Contents

[Test Cases 2](#_Toc449891322)

[Execution Based Testing 5](#_Toc449891323)

[Black Box Testing 5](#_Toc449891324)

[Glass Box Testing 5](#_Toc449891325)

[Integration Testing 6](#_Toc449891326)

[Non-functional and Functional Testing 6](#_Toc449891327)

[Non-functional testing 7](#_Toc449891328)

[ Overview: 7](#_Toc449891329)

[Pseudo Code for gameoption.cpp 7](#_Toc449891330)

[Pseudo Code for LoginScene.cpp 7](#_Toc449891331)

[Pseudo Code for maintictactoe.cpp 8](#_Toc449891332)

[Pseudo Code for Main.cpp 9](#_Toc449891333)

[Pseudo Code for registrationScene.cpp 9](#_Toc449891334)

[Pseudo Code for ResetPassword.cpp 10](#_Toc449891335)

[Pseudo Code for selectgamemode.cpp 12](#_Toc449891336)

[ Preparation: 12](#_Toc449891337)

[ Inspection: 12](#_Toc449891338)

[ Re-work: 12](#_Toc449891339)

[ Follow up: 14](#_Toc449891340)

[Functional Testing 14](#_Toc449891341)

[Unintended Actions Report 19](#_Toc449891342)

[Bug report: 19](#_Toc449891343)

In our implementation test plan, we are implementing Glass-Box Unit Testing. Specifically, the Branch Testing is implemented to make sure that these features are working correctly:

* Login as an existing player or a guest.
* Player vs Player (PvP), Guest Vs Guest and Player Vs Guest.
* Single player (guest or registered player) vs AI.
* Choose level of difficulty.
* Choose game piece (X is blue or O is red).
* Choose who goes first.

# Test Cases

We test for every single statement in our code and all our modules.

These are the test cases:

|  |  |
| --- | --- |
| TEST CASE 1: Login case | |
| Input | Legit user |
| Legit password |
| Output | Welcome message and players option |

|  |  |
| --- | --- |
| TEST CASE 2: Fail login case | |
| Input | Legit user name |
| Wrong password |
| Output | Wrong user name or password |

|  |  |
| --- | --- |
| TEST CASE 3: Failed login with wrong user | |
| Input | Incorrect user |
| Wright password |
| Output | Wrong username or password |

|  |  |
| --- | --- |
| TEST CASE 4: Pre case after login | |
| Input | Players VS player chosen |
| Output | Player tile style of stone chosen |

|  |  |
| --- | --- |
| TEST CASE 5: Pre case after successful login | |
| Input | Player VS AI |
| Output | Difficulty level shows |

|  |  |
| --- | --- |
| TEST CASE 6: Pre case after Successful login | |
| Input | Player choose X or O |
| Output | Option to go first or last |

|  |  |
| --- | --- |
| TEST CASE 7: Pre case after successful login | |
| Input | Player chooses easy mode, medium or hard |
| Tile option is presented for player as X or O |
| Output |  |

|  |  |
| --- | --- |
| TEST CASE 8: When the player decide to go first | |
| Input | Player chooses to go first |
| Output | Game board initiates and player goes as tittle option |

|  |  |
| --- | --- |
| TEST CASE 9: Player chooses to go last | |
| Input | Player decide to go last |
| Output | Game board initiates and player goes first |

|  |  |
| --- | --- |
| TEST CASE 10: Forgot password | |
| Input | Username |
| Firstname |
| Lastname |
| Output | Security question asked |
| Answer provided and |
| Input box to reset password and confirm it |

|  |  |
| --- | --- |
| TEST CASE 11: Pressing Help button | |
| Input | Press event “Help” |
| Output | Message display to it content |

|  |  |
| --- | --- |
| TEST CASE 12: Registration | |
| Input | Firstname, lastname, username, password |
| Re-type password, security question |
| Answer, re-type answer. |
| Output | Query the database and display either successful |
| If user is not repeated or password are matched |

|  |  |
| --- | --- |
| TEST CASE 13: Press Guest Button | |
| Input | Press Guest button |
| Output | Show the game Board and |
| * Player vs Guest |
| * Player vs AI |

|  |  |
| --- | --- |
| TEST CASE 14: Pre-case Guest vs Player | |
| Input | Guest vs Player |
| Output | Ask user to login and choose tile |
|  | Then game starts based on turns and player goes first |

|  |  |
| --- | --- |
| TEST CASE 15: Guest vs AI | |
| Input | Guest vs AI |
| Output | Shows difficult level based on selected terms: easy, medium, and hard |

|  |  |
| --- | --- |
| TEST CASE 16: Guest vs AI “Easy” | |
| Input | Choose easy mode and play |
| Output | Game Starts |
| * Plays first turn |
| * Guest play turns |
| * End game called after 36 turns |
| * Update players score |

|  |  |
| --- | --- |
| TEST CASE 17: Guest vs AI “Medium” | |
| Input | Choose medium mode and play |
| Output | Game Starts |
| * Plays first turn |
| * Guest play turns |
| * End game called after 36 turns |
| * Update players score |

|  |  |
| --- | --- |
| TEST CASE 18: Guest vs AI “Hard” | |
| Input | Choose hard mode and play |
| Output | Game Starts |
| * Plays first turn |
| * Guest play turns |
| * End game called after 36 turns |
| * Update players score |

|  |  |
| --- | --- |
| TEST CASE 19: Player 1 vs Player 2 | |
| Input | During game mode, players take turns |
| Output | Game ends after 36 squares are filled |
| * Determine winner between P1 and P2 |
| * Display winner message |
| * Update database |
| * Return to main menu |

|  |  |
| --- | --- |
| TEST CASE 20: Draw game Player 1 vs Player 2 | |
| Input | End game player 1 vs Player 2 |
| Output | Option presented to start a new game |
| Or quit |

|  |  |
| --- | --- |
| TEST CASE 21: Draw game Player 1 vs Guest | |
| Input | End game as draw Player 1 vs Guest |
| Output | Choose New Game or Quit |

|  |  |
| --- | --- |
| TEST CASE 22: Player vs AI Draw Game | |
| Input | Game ends in a draw |
| Output | Choose New Game or Quit |

|  |  |
| --- | --- |
| TEST CASE 23: End Game Follow-Up | |
| Input | New game after game is ended |
| Output | Destroy old game point |
| Initiate new game |

|  |  |
| --- | --- |
| TEST CASE 24: Quitting Game | |
| Input | Quit game |
| Output | Clean up memory uses |
| Game closes |

# Execution Based Testing

Once the program is in a final state, we do an execution based test to make sure that all the requirements for the product are met.

We are testing for the following components in our program:

* Reliability
* Robustness
* Performance
* Utility
* Correctness

We answer these questions related to each one of these areas.

The questions we have according to our test plan include:

* How easy is our product to use?
  + Our game is simple and easy to use for any user.
* How reliable is our database?
  + Low probability of having an issue that wipes out our database.
* Is the communication well established between all the modules?
  + Yes, the modules call upon each other.
* Are we having unacceptable results when utilizing valid inputs?
  + We have a way to manage incorrect inputs and guide the user to the correct utilization of our product.
* How badly does our product react with incorrect use?
  + In the case of second guest user, the user is not given instructions to enter “guest” as the username and they receive an error message “Wrong username”.
* How often are we updating the scores to keep track of the winner?
  + We start updating after 7th turn, then we update after each turn.

# Black Box Testing

* + Choose who goes first.

# Glass Box Testing

## Integration Testing

**Start game test:** The user is given the option to pick between O and X before starting the game. At this stage, the option to go first is given as well. The transition between choosing O or X mode to starting game mode executes smoothly and the icon is displayed correctly according to user's input.

**Track user name test:** The user is able to see his/her current score with this feature. The logged in user name is tracked in order to keep and display score updates. The application is able to help the user whenever they want to see their score by viewing the top area. This is done with the implementation of a score bar that's displayed to the right side of the current window.

**Change between modes test:** The user is able to go directly from Player Vs. AI or Player Vs. Player to the start game mode. The user input is taken and implemented as a trigger to transition between these modes. It also gives the option to go back and change the play mode again if desired. If the user decides to end the game on either of these two modes, the option is given and the play game mode is ended. Scores are lost at this point and the user has to start the game over, then go through all the options in order to enter the play mode again. The help mode is always accessible in all modes of the game.

**Choose level of difficulty test:** The user is expected to choose between three levels of difficulty only when the player Vs. AI mode is implemented. These options remain disabled whenever the user enters the player Vs. player mode. Each selected option will return a value which will trigger the specific difficulty level mode: easy returns 1, medium returns 2, and hard return a 3.

A call is made based on the returned values to select and enter a difficulty level, then a transition from this mode to the start game mode is made.

**Check if board is full test:** The user gets a notice of completely full board after a function call is made to enter the check board mode. This will detect if all the squares are full and makes a call to a function dedicated to check the scores. The check winner mode is implemented at this point. The user is given a notice of who is the winner and is the option to start a new game. If the user decides to enter into play mode again, the same settings are kept. However, if the user decides not to play again, the exit game mode is implemented and the user would need to login or play as guest before starting a new game.

**GUI applicability test:** The user is able to easily adapt to the interface presented. A few modifications are made based on minor suggestions by the user after this test. The resizing, realignment, the integration of representative icons for buttons, and redesigning of windows is made to create a consistent look throughout all of the interfaces. With these modifications, the user has a better change to adapt easily to all of the modes and options presented.

# Non-functional and Functional Testing

## Non-functional testing

### Overview:

The original pseudocode used for testing

### 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.

## Functional Testing

Functional Testing using glass box testing along with pathway steps taken:

mainTicTacToe

LoginScene

ForgotPassword

Register

Help

Guest

A

C

D

E

F

B

Main

* Login: Can go from main to login scene. One imitated can call the event to show

A

B

* User input and password
* Query the database and print the success message or failure
* Back to main menu

Main forgotPassword

A

C

* Can go from main to forgot password
* Once initiated, event applies by asking the following:
* Enter first name, last name, and username.
* Query by database and asks for the security answer of the question.

o If the answer is right, then request new password.

* Else, if wrong answer, ask to input answer until successful then back to main.

Main Register

A

D

Even starts:

1. Ask for fist name, last name, username, password, repeated password, see question and answer, and repeated answer.
2. Query input into database then log if all inputs are right, go back to the registration form if not successful.

A

E

Press help button: initiate even handler to imitate and states: login or sign in as a user or play as a guest.

A

B

C

D

E

F

Guest

A

F

Press the guest button and events,

1. Propose to play as guest player as X or O 2. Choose method, then game event trigger 3. Game starts:

Game option o

Play as X or O

Game board

Difficulty level

G

H

I

J

G. allows to call Game Option

* Choose player vs player or player vs AI
* Choose one option
* Then go to tile option as X or O
* Then game statistics

F. Chooses AI as Game Option

* Initiate difficulty level as easy, medium, or hard
* Game calls for difficulty level
* Once initiated, chooses to go as X or O.
* Then either you or AI play first.
* Game start depends on the level.

A

B

C

D

E

F

G

H

J

I

Player Vs Player

M

K

K

This is the complete integration test by using glass box with pathway

# Unintended Actions Report

Utility: user’s needs are met, ease of use, useful functions, etc.

Reliability: easier of frequency and critically of failure.

Robustness: range of operating conditions, handling invalid inputs.

Performance:

Program testing to verify the presence of bugs

Unintended interactions

## Bug report:

#1)

Type: Correctness

Detail: Bug on PvP, and PvAI, enter the Username and password should not be empty once submit is pressed

Status: Fixed! Added isEmpty checker to return the function user is empty. Also checked password in case it is empty to return the function from the beginning

#2)

Type: Correctness

Detail: Select option on PvAI and PvP, icons are not displaying.

Status: Fixed! Icons are implemented now.

#3)

Type: Robustness

Detail:

For guest option only: After choosing the stone and inputting the username, once "Cancel" is clicked, or the window is closed it will run the game anyways.

Status: Fixed! Once cancelled is clicked, it will bring you back to the main menu.

#4)

Type: Utility

Detail: Change the window or give instructions to specify to type "guest" as the username.

Status: Fixed! We added a ReadMe.txt file for instructions.

#5)

Type: Correctness

Details: Username losses its value after the first play from either user or AI.

Status: Fixed!

Cause: username1, and username2 are set to be a private variable. This would work if you have getting and setter functions to set it and get it for every team.

#6)

Type: Utility

Details: A Boolean value was set as isGuest in the playAsGuest function which is not being used.

Cause: isGuest was declared and later on not used, which came to have a different value being used elsewhere.

Status: fixed! Simply used the other variable from the AiClass to set the state of a guest into guest1 and guest2.

#7)

Type: Enhancement

Details: Too many files were being included everywhere and repetitively.

Status: Fixed! Created a header file to include on each class. This header file has all of the library we are using for the game.

#8)

Type: Utility

Details: Database issues with different types of platform of software. Mac OS, Windows, Linux and other type of known software have been having issues to connect to the database.

Status: Fixed! We decided to host SQLite within the application in order to prevent users from having issues with updating scores or login.

#9)

Type: Utility

Details: If the close button is clicked, the variables are not reset. They remain within the memory.

Status: Fixed! Every time the game board is cleared, it resets all the variables with it.

#10)

Type: Reliability

Details: Once finishing the second game of player vs. AI on medium mode, the AI’s turn will get stuck on the loop and fail to execute properly. The program will then freeze and crash. This bug is found when using guest or as a registered user.

Status: Fixed!

#11)

Type: Utility

Detail: On Guest vs AI easy mode, there is no option to prompt the user to play again or to quit the game.

Status: Fixed!

#12)

Type: Reliability

Detail: On the second run of the game of the easy AI, the end game function is never called and the score is showing the wrong value (it could be due to the previous game score it is being added to). This bug is applied to both guest and registered user.

Status: Fixed!

#13)

Type: Reliability

Detail: Whenever guest mode is selected and player vs. player mode is selected, if the second user tries to login with a valid registered account, it will keep displaying the message of wrong username and password.

Status: Fixed!

#14)

Type: Reliability

Detail: After the second run of any game the score will display the wrong value and the end game function is never called.

Status: Fixed!

#15)

Type: Reliability

Detail: whenever the user signs up for a new account, the database will not verify if the username is taken properly, it registers the account anyway.

Status: Not fixed yet

#16)

Type: Reliability

Detail: When clicking on forgot password and inputting the information, the security question is not displaying, and the security answer is never verified, any input will be considered as a valid input.

Status: Fixed!

#17)

Type: Reliability

Detail: When canceling the request on password reset, it will still state that the password has been changed.

Status: Fixed!

#18)

Type: Reliability

Detail: When clicking on forgot password, the first input fields (first name, last name, and user name) are never verified, the user is still prompted to input the security answer.

Status: Fixed!