Football Higher or Lower

SCN 161873381

Alessandro Stoneham

TABLE OF CONTENTS

ANALYSIS

Description of the problem

I am creating a game similar to 'Higher or Lower' with fifa cards.

- Users are presented a card from the game EA FC 24, which contains a football player's name, image, overall rating, position, nation, league, club, and 6 other minor ratings for pace, shooting, passing, dribbling, defending, and physicality
- The program then randomly decides which rating will be used to compare with the next card and shows the name of the selected to the user
- The program then shows 3 hints (the league, position, and nation) to hint the player as to what the next card will be
- The user then guesses if the selected rating will be higher or lower on the next card
- Users are given 15 seconds to make their guess or the game ends
- If correct, one point is given to the user. The game keeps going until the user guesses wrong or runs out of time
- Users are able to save their high score in a database using a username and password and will be able to login to keep playing, or they can opt to play as a guest where their progress will not save

The terminology used in this project may be confusing, to help I have created some definitions:

- 'player/players/card/cards' refers to the card of the football player, which is stored in an array of records. An example photo is below:
- For more information about the array of records check the respective section of this document (Design>Data structure design>array of records: cardsArray[])

- 'User/users' refers to the human that is playing the game
- 'Stack' refers to the program

- ‘Card/Screen/Page’ refers to one specific page of the program that the user can see. Only one is open at once.

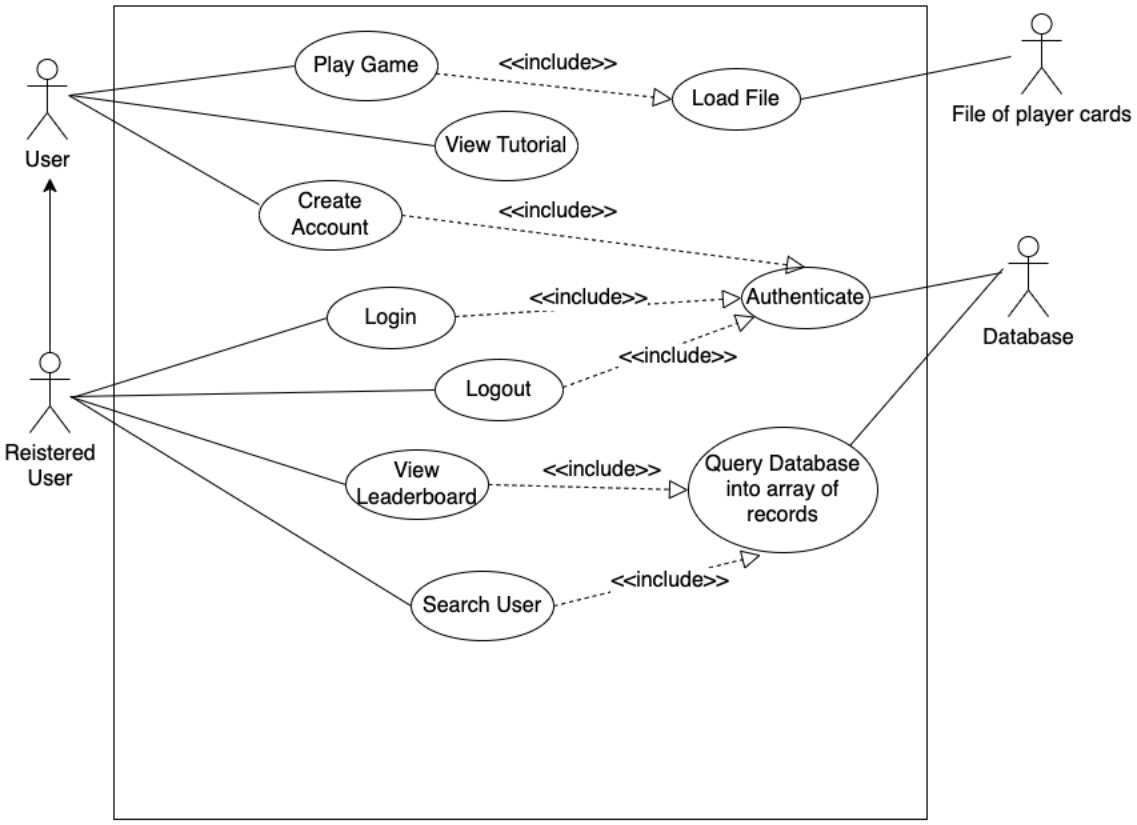
Advanced Higher Concepts

- My game will include an array of records, with 122 records, to store the information of each football player. Users login information (username and password) and high score will be kept in a single table database in SQLite.
- Users' information (username and highscore) will be queried from the database into another array of records and bubble sorted. The top 10 will be displayed on a leaderboard.
- Users will also be able to search for any user’s high score through a binary search on their username. The array of records will be bubble sorted alphabetically by username when the user wants to perform this action for the binary search to work.
- When a user achieves a new high score, the database will be updated to reflect the new high score.
- Upon account creation a new record will be created in the database with the user’s username, password, and a high score of 0.
- All inputs to my program will be validated

Constraints

- I have completed the Advanced Higher course which gives me the knowledge to create this project.
- I will be using Livecode 9.6.9 to create my project as I have multiple years of experience with the language and development environment.
- To view the contents of my database, I will use the free application DB Browser for SQLite
- I will be using Google Suite apps to document my progress and write this project report.
- Both Livecode and Google Suite are licensed by my school meaning there is no cost implication.
- I am using Mac to build my project meaning there are little performance or hardware issues and restrictions. Although built on Mac, my project will also run on Windows systems.
- My game will make use of football player card images from EA FC 24, which would have a legal implication if I was to commercially publish the game, however as it is for an academic purpose and the project will not be commercially published, the use of these images has no legal implications.
- In order for my game to not depend on EA FC 24’s image hosting servers, the images of the cards, as well as all other images used in my program will be stored on device in a folder with the program. This requires devices to have enough storage space (~30MB) to run the program with all required image files.
- My project takes into account GDPR laws but as it will not be commercially published all entries into my database are fictional and not in breach of GDPR.
- My project has to be completed by the 27th March to allow time for it to be sent to the SQA to meet the official deadline.
- It helps if the user playing has a basic understanding of EA FC 24 card stats, but if not there is an instructions page to explain the game.

UML Use Case Diagram



Requirements Specification

Functional Requirements

1. Login System

1.1 Login with account

1. The file of player cards will be read into an array of records. If this fails an error message will be shown.
2. A connection to the database will be opened.
3. If it does not already exist, a table in the database will be created.
4. Users will be prompted to enter their username. The username will be checked if it exists in the database. If no username was found the program will display an error.
5. If a username is found, a prompt to enter a password will appear. If the password entered by the user matches the related password in the database, the user will be sent to the home screen. If not, an error message will appear.
6. The connection to the database will be closed.

1.2 Play as guest

1. The file of player cards will be read into an array of records. If this fails an error message will be shown.
2. A variable “loggedinUsername” will be set to empty, this allows certain procedures to know they do not need to update the database
3. A variable “loggedinHighScore” will be set to 0
4. The user will be sent to the home screen

1.3 Create account

1. A connection to the database will be opened.
2. If it does not already exist, a table in the database will be created.
3. Users will be prompted to enter a username of length between 3 and 15 characters. The entered username will be validated to ensure it meets the length check and additionally that it is unique and does not already exist in the database. Error messages will be shown if the username is invalid/taken.
4. If the entered username is valid and unique, the user will be asked to enter a password. The password must be between 8-20 characters long and will be validated to ensure that is the case.
5. A new record will be inserted into the database with the username and password chosen by the user, the high score of the new account will be set to 0.
6. Once the account is created the user will then be prompted to login to start playing.
7. The connection to the database will be closed

1.4 Logout/save

1. User will be sent back to the login screen

2. Home Screen

1. Users are presented with buttons to Play Game, View Leaderboard, Search User, View Instructions, and Logout.

3. Gameplay

3.1 Start Game

1. The UI elements are reset to their starting position
2. Current score is set to 0, and countdown is not set as the first guess has no time limit. This is shown using an infinity symbol.
3. A starting card is randomly chosen from the array of records cardsArray[] and shown on screen
4. A rating is randomly chosen to be the comparison condition and is shown on screen
5. A next card is randomly chosen from the array of records cardsArray[]
6. 3 hints are then displayed for the next card

7. The user is then sent to the gameplay screen

3.2 User submits guess through button ‘Higher’ or button ‘Lower’

1. Check if a countdown is currently running, if so, stop the countdown
2. Set the countdown to 16 seconds (not 15: read Design>Project design>Gameplay screen for reason why)
3. Start countdown timer
4. If the guess is correct, a tick icon is shown to the user, the next card is placed into the current card, a new condition is chosen, a new next card is chosen, new hints are shown, and the current card is shown to the user. The users score is incremented by 1, and this is updated on the screen
5. If the guess is incorrect, a cross icon is displayed, followed by the image of the next card. The program checks if a new high score was achieved, if a user is logged in, the database is updated to reflect this. The game over screen is then shown with the user’s score and high score. If the user’s score was 0, the time runs out, or a new high score was achieved, a message is shown on the game over screen to reflect this.

4. Search Player

1. A connection to the database will be opened
2. The database will be queried and the username and high score will be inserted into an array of records called userScores[]
3. The connection to the database will be closed
4. This array of records will then be bubble sorted by username alphabetically
5. User will be prompted to enter a username to search
6. If a username is entered, a binary search will be carried out to find the user’s score.
7. If found, the username and high score will then be shown on screen, if not found the program will tell the user that it could not find that user in the database

5. View Leaderboard

1. A connection to the database will be opened
2. The database will be queried and the username and high score will be inserted into an array of records called userScores[]
3. The database connection will be closed
4. This array of records will then be bubble sorted by high score highest to lowest
5. This array of records will then be displayed in a table showing the first 10 records (the 10 highest scores and their respective usernames).
6. The user will be swapped to the leaderboard card
7. A button in the bottom left corner returns the user to the home screen

6. Tutorial

1. Users are taken to the tutorial screen where they can see instructions on how to play
2. A button in the bottom left corner returns the user to the home screen

7. Database

1. The database will be queried upon login, creating accounts, searching users, and viewing the leaderboard
2. New records will be inserted when accounts are created
3. Records will be updated to reflect users' new high scores

End User Requirements

1. Login System

1. Users can log in with an existing account using their credentials (username and password)
2. Users can create an account with a unique username and a password
3. Users can play as a guest in which progress will not be saved

2. Home Screen

1. Users can start a game, logout, view the leaderboard, search for a user's score, or view instructions

3. Gameplay

1. Users can click the 'Higher' or 'Lower' button to submit their guess
2. Users can view their current score and high score
3. Users can view the current selected player card and the 3 hints for the next card
4. Users should be able to see how long they have left to make their guess

4. Search Player

1. Users can enter a username and be shown the high score of the user whose name they entered

5. View Leaderboard

1. Users can view the top 10 highest scores along with the username of the user that achieved the score
2. Users can return to the home screen

6. Tutorial

1. Users can view the instructions on how to play the game
2. Users can return to the home screen

Project Plan

Task	Duration (~)	Start date	End date
Description of the problem	1 day	15th Dec 2023	15th Dec 2023
UML Case Diagram	1 day	16th Dec 2023	16th Dec 2023
Functional Requirements	2 days	17th Dec 2023	18th Dec 2023
End-User Requirements	1 day	19th Dec 2023	20th Dec 2023
Design: Data Structures	1 day	6th Feb 2024	6th Feb 2024
Design: Pseudocode for functional/end user requirements, advanced higher concepts, and input validation	12 days	7th Feb 2024	19th Feb 2024
Design of DDD: data dictionary, design of queries	2 days	20th Feb 2024	21st Feb 2024
Design: Wireframe design	3 days	22nd Feb 2024	24th Feb 2024
Create csv file of player cards	5 days	25th Feb 2024	29th Feb 2024
Implementation: Play as guest	1 day	1st Mar 2024	1st Mar 2024
Implementation: Create account	1 day	2nd Mar 2024	2nd Mar 2024
Implementation: Login	2 days	3rd Mar 2024	4th Mar 2024
Implementation: How-to screen	1 day	5th Mar 2024	5th Mar 2024
Implementation: Logout system	1 day	6th Mar 2024	6th Mar 2024
Implementation: Gameplay	4 days	7th Mar 2024	10th Mar 2024
Implementation: Leaderboard	1 day	11th Mar 2024	11th Mar 2024
Implementation: Search user	1 day	12th Mar 2024	12th Mar 2024
Implementation: New skills learned	1 day	13th Mar 2024	13th Mar 2024
Testing: Final Test Plan	2 days	14th Mar 2024	15th Mar 2024
Testing: Requirements Testing	4 days	16th Mar 2024	19th Mar 2024
Testing: Persona Testing	1 day	20th Mar 2024	20th Mar 2024
Evaluation	3 days	21st Mar 2024	23rd Mar 2024

Resources required at implementation: Livecode, DB Browser for SQlite, sqlite, google suite apps

DESIGN

Data structure design

Array of records: cardsArray

- This array is read into the program every time a user logs in or plays as a guest, and will not ever be modified. This array has to be kept as a global array so it can be accessed by other elements in the program, eg. the main game.

```
RECORD card {
   STRING name,
   STRING nation,
   STRING nationphoto,
   STRING league,
   STRING club,
   STRING clubphoto,
   STRING position,
   INTEGER ratingOVR,
   INTEGER ratingPAC,
   INTEGER ratingSHO,
   INTEGER ratingPAS,
   INTEGER ratingDRI,
   INTEGER ratingDEF,
   INTEGER ratingPHY,
   STRING playerphoto
}

DECLARE cardsArray[122] AS array of record card
```

Array of records: userScores

- This array will be created from the results queried from the database when a user searches for a player or clicks on the ‘Leaderboard’ button.

```
RECORD score {
   STRING username
   STRING score
}
```

DECLARE userScores[X] AS array of record score
(the array will be the size of the number of records (X) returned from the database)

Global variables

I will have three additional global variables:

1.) loggedinUsername: STRING
2.) loggedinHighScore: INTEGER

These variables act as session variables so they can be used throughout the program, and are updated every time the user logs in or plays as a guest

3.) numOfCards: INTEGER

This variable holds the amount of records in the cardsArray[] array. I could have just used the number 122 throughout my program but to improve maintainability I have set this number to a global variable, this way it can be accessed throughout the program and will work if I choose to add more cards.

Project design

Below are all the buttons in my program with the names of the procedures that they use, and below all the buttons are the pseudocode refinements for the procedures. These can be viewed as the top-level design for each function of my program.

Login page

Button: Create Account

(all procedures/functions below are called from this button but are written on the 'Login page' card)

- Open Database (see design 1)
- Create Table (see design 2)
- Create username (see design 3)
    - Validate username (see design 4)
    - Check username is unique (see design 5)
- Create password (see design 6)
    - Validate password (see design 4)
- Create new database entry (see design 7)
- Close database (see design 8)

Button: Log In

(all procedures/functions below are called from this button but are written on the 'Login page' card)

- Load file of players into array of records (see design 9)
  IF <number of elements of> cardsArray[] > 0 THEN // this ensures the file has been read in
    - Open Database (see design 1)
    - Create Table (see design 2)
    - Get Username to Login (see design 10)
    - Get Password to Login (see design 11)
    - Close Database (see design 8)
    - <Switch to home screen>

  ELSE
    - <Shows error message> (see wireframe XX)

  END IF

Button: Play As Guest

(all procedures/functions below are called from this button but are written on the 'Login page' card)

- Load file of players into array of records (see design 9)
- set loggedinUsername to ""
- set loggedinHighScore to 0
- set <the text of field 'title' of card 'home'> to "WELCOME GUEST"

  IF <number of elements of> cardsArray[] > 0 THEN *// this ensures the file has been read in*
    - <Switch to home screen>

  ELSE
    - <Shows error message> (see wireframe XX)

  END IF

Home screen

Button: Start Game

(all procedures/functions below are called from this button but are written on the 'gameplay' card)

- Initialise UI (see design 18)
- Reset Score (see design 19)
- Choose start card (see design 20)
- Choose condition (see design 21)
- Choose next card (see design 22)
- Choose hints (see design 23)
- <Switch to gameplay screen>

Button: Logout

- Empty global variable ‘loggedinUsername’
- Empty global variable ‘loggedinHighScore’
- <Switch to login screen>

Button: Leaderboard

Parameter passing:

Procedure	**IN:**	**OUT:**
Open Database		dbID
Query Database	dbID	userScores
Close Database	dbID	
Bubble Sort Scores	userScores	userScores
Display Leaderboard	userScores	

- Open Database (see design 1)
- Query Database for Usernames and High Scores (see design 12)
- Close database (see design 8)
- Bubble Sort on scores (see design 13)
- Display Leaderboard (see design 14)
- <Switch to leaderboard screen>

Button: Find Player

Parameter passing:

Procedure	**IN:**	**OUT:**
Open Database		dbID
Query Database	dbID	userScores
Close Database	dbID	
Bubble Sort Usernames	userScores	userScores
Get Target Username		targetUser
Binary Search	targetUser, userScores	

- Open Database (see design 1)
- Query Database for Usernames and High Scores (see design 12)
- Close Database (see design 8)
- Bubble Sort on Usernames (see design 15)
- Get target username (see design 16)
  IF targetUsername NOT “” THEN
    - Binary search (see design 17)
  END IF

Button: How to Play

- <Switch to tutorial page>

Gameplay screen

Button: Lower

(all procedures/functions below are called from this button but are written on the ‘gameplay’ card)

  IF countdown is running (see design 24) THEN
    - Stop countdown (see design 25)
  END IF
- Fill counter (see design 26)
- Start running counter (see design 27)
- Submit guess (see design 28)

Procedure	**IN:**	**OUT:**
Fill counter	pHowMany (16)	
Submit guess	Choice (“LOWER”)	

All other procedures make use of local card variables

Button: Higher

(all procedures/functions below are called from this button but are written on the ‘gameplay’ card)

  IF countdown is running (see design 24) THEN
    - Stop countdown (see design 25)
  END IF
- Fill counter (see design 26)
- Start running counter (see design 27)
- Submit guess (see design 28)

Procedure	**IN:**	**OUT:**

Fill counter	pHowMany (16)	
Submit guess	Choice (“HIGHER”)	

All other procedures make use of local card variables

N.B. Although users have 15 seconds to make their guess, we use 16 as the program has to complete other tasks, which takes roughly 1 second to do, before the user can actually submit their next guess

Tutorial screen

Button: Return to home

- <Go to home screen>

Leaderboard screen

Button: Return to home

- <Go to home screen>

Pseudocode design

N.B: the variables listed in each procedure are broken down into global variables, procedure-specific variables, and variables that are local to the card. Variables local to the card can be used in all procedures that are written on the card, which makes parameter passing between procedures on the same card unnecessary.

1. Opening connection to database
   Local to procedure variables:
   - dbPath: string, file path for database

   Local to card variables:
   - dbID: integer, identifier used for the database

   1.1 set dbPath to <file location/userDB.sqlite>
   1.2 set dbID to <open database(sqlite, dbPath)>

2. Creating Database Table
   Local to procedure variables:
   - SQL: string, sql code to be executed

   Local to card variables:
   - dbID: integer, identifier used for database

   2.1 set SQL to <database create statement (data dictionary)>

2.2 Execute SQL (dbID, SQL)

3. <u>Create username</u>

Local to procedure variables:

- valid: boolean, used to show if the username matches criteria
- unique: boolean, used to show if the username already exists in the database

Local to card variables:

- desiredUsername: string, the desired username for the user's account

3.1 set valid to false
3.2 set unique to false
3.3 RECEIVE desiredUsername from keyboard
3.4 set valid to checkValid(desiredUsername, 3, 15) *// calls function in design 4*
3.5 set unique to checkUnique(desiredUsername) *// calls function in design 5*
3.6 WHILE valid = false or unique = false DO
    3.7 IF desiredUsername = "" THEN
        3.8 EXIT PROCEDURE // this is if the user entered no name
    3.9 END IF
    3.10 SEND "Retry. Username must be between 3-15 characters and unique" TO DISPLAY
    3.11 RECEIVE desiredUsername from keyboard
    3.12 set valid to checkValid(desiredUsername, 3, 15)
    3.13 set unique to checkUnique(desiredUsername)
3.14 END WHILE

4. <u>Function to validate username/password</u>

Parameters IN:

- stringToCheck: STRING
- lowerLimit: INTEGER
- upperLimit: INTEGER

Function returns a boolean value

Local to function variables:

- stringLength: integer

4.1 set stringLength to length(stringToCheck)
4.2 IF stringLength >= lowerLimit AND stringLength <= upperLimit THEN
    4.3 return TRUE
4.4 ELSE
    4.5 return FALSE
4.6 END IF

5. <u>Function to check the username does not already exist in DB</u>

Parameters IN:

- desiredUsername

Function returns a boolean value
Local to function variables:

- SQL: string, sql code to be executed
- output: output from SQL query

5.1 set SQL to <Database design 1>
5.2 set output to Execute SQL(dbID,SQL)

5.3 IF output = “” THEN
    5.4 return TRUE
5.5 ELSE
    5.6 return FALSE
5.7 END IF

6. Create password

Local to procedure variables:

- valid: boolean, used to show if the password meets the criteria

Local to card variables:

- desiredPassword: string, the desired password for the user’s account

6.1 set valid to false
6.2 RECEIVE desiredPassword from keyboard
6.3 set valid to checkValid(desiredPassword, 8, 20) *// calls function in design 4*
6.4 WHILE valid = false DO
    6.5 SEND “Retry. Password must be between 8-20 characters” TO DISPLAY
    6.6 RECEIVE desiredPassword from keyboard
    6.7 set valid to checkValid(desiredPassword, 8, 20)
6.8 END WHILE

7. Create new database entry

Local to procedure variables:

- SQL: string, sql code to be executed

Local to card variables:

- desiredUsername: string, the desired username for the user’s account
- desiredPassword: string, the desired password for the user’s account
- dbID: integer, identifier used for the database

7.1 set SQL to <Database design 5>
7.2 Execute SQL (dbID, SQL)
7.3 SEND “Successfully created an account for" && desiredUsername & "! Please log in using the credentials you made” TO DISPLAY

8. Close Database

Local to card variables:

- dbID, integer, identifier used for the database

8.1 Run Close database function (dbID)

9. <u>Read file of players into array of records</u>

Global variables:
- cardsArray[]

Local to procedure variables:
- fileChoice: string, stores the location of the file containing player cards
- temp[]: 1D array of string, contains unfiltered data from the file read in
- arraySize: integer, stores the size of the array 'temp'

9.1 set arraySize to 0

9.2 set file choice to <location of folder/HLFUT24.csv>
9.3 open file for read
9.4 read file
9.5 set temp = <contents of file fileChoice>
9.6 split temp by return
9.7 close file

9.8 set arraySize = <number of elements of temp>
9.9 set each record of temp into its unique element in cardsArray[]

10. <u>Get Username to Login</u>

Local to procedure variables:
- SQL: string, sql code to execute
- output: string, output from sql query

Local to card variables:
- usernameToFind: string, target username entered by user

10.1 RECEIVE usernameToFind FROM KEYBOARD
10.2 IF usernameToFind = "" THEN
10.3 EXIT PROCEDURE *// assumes user accidentally clicked the button as they put nothing in the input box*
10.4 END IF

10.5 SET SQL TO <Database design 1>
10.6 SET output TO Execute SQL(dbID,sql)
10.7 IF output = "" THEN
10.8 SEND "Username not found" TO DISPLAY
10.9 <Get Username to Login> (recursive function)
10.10 END IF

11. Get Password to Login

Global variables:

- loggedinUsername: string, stores the name of the user currently logged in
- loggedinHighScore: integer, scores the high score of the user currently logged in

Local to procedure variables:

- SQL: string, SQL code to execute
- output: string, output from SQL query
- passwordToFind: string, password entered by the user

Local to card variables:

- dbID, integer, identifier used for the database

11.1 RECEIVE passwordToFind FROM KEYBOARD
11.2 set SQL to <Database design 2>
11.3 set output to Execute SQL(dbID,SQL)

11.4 set SQL to <Database design 3>
11.5 set loggedinHighScore to Execute SQL(dbID,SQL)

11.6 IF output = “” then
    11.7 SEND “Password incorrect” TO DISPLAY
    11.8 EXIT PROCEDURE
11.9 END IF
11.10 set loggedinUsername TO usernameToFind
11.11 set <text of field ‘title’ of card ‘home’> TO "WELCOME" && toUpper(loggedinUsername)

12. Query Database for Usernames and High Scores

Parameters:

- IN: dbID
- OUT: userScores[]

Local to procedure variables:

- sql: string, sql code to execute
- userScores[], array of records, holds the username and high score of every player
- output: string, stores the output of the executed SQL query
- arraySize: integer, stores the size of the array userScores[]

12.1 set sql to <Database design 4>
12.2 set output to Execute SQL(dbID,sql)
12.3 split output by return
12.4 set arraySize to <number of elements of array output>
12.5 set each record of output into its unique element in userScores[]

13. Bubble sort on leaderboard (scores highest to lowest)

Parameters:

- IN: userScores
- OUT: userScores

Local to procedure variables:

- swapped: boolean, indicates if the sort is still active
- limit: integer, an upper boundary to make the sort more efficient
- temp: integer, temp value to hold scores as they are being swapped in the array

13.1 set swapped to true
13.2 set limit to <length of array of records userScores>
13.3 while swapped = true and limit >=1 do
    13.4 set swapped to false
    13.5 for i = 1 to limit-1 do
        13.6 if userScores[i][score] < userScores[i+1][score] then
            13.7 set temp to userScores[i]
            13.8 set userScores[i] to userScores[i+1]
            13.9 set userScores[i+1] to temp
            13.10 set swapped to true
        13.11 end if
    13.12 end for
13.13 set limit to limit-1
13.14 end while

14. Display Leaderboard

Parameters:

- IN: userScores

No other variables are used in this procedure

14.1 set <the text of field 'lbField' of card 'leaderboard'> to be empty
14.2 FOR loop = 1 to 10 DO
    14.3 set <the text of field 'lbField' of card 'leaderboard'> to loop & tab & userScores[loop]["username"] & tab & userScores[loop]["score"] & cr
14.4 END FOR

15. Bubble Sort on usernames

Parameters:

- IN: userScores
- OUT: userScores

Local to procedure variables:

- swapped: boolean, indicates if the sort is still active
- limit: integer, an upper boundary to make the sort more efficient
- temp: integer, temp value to hold scores as they are being swapped in the array

13.1 set swapped to true
13.2 set limit to <length of array of records userScores>

13.3 while swapped = true and limit >=1 do
13.4 set swapped to false
13.5 for i = 1 to limit-1 do
13.6 if userScores[i][username] > userScores[i+1][username] then
13.7 set temp to userScores[i]
13.8 set userScores[i] to userScores[i+1]
13.9 set userScores[i+1] to temp
13.10 set swapped to true
13.11 end if
13.12 end for
13.13 set limit to limit-1
13.14 end while

16. Get target username

Parameters:

- OUT: targetUser

No other variables are used in this procedure

16.1 RECEIVE targetUser FROM KEYBOARD

17. Binary Search

Parameters:

- IN: targetUser, userScores

Local to procedure variables:

- highest: integer, stores the maximum index value
- lowest: integer, stores the minimum index value
- middleNum: integer, stores the middle index value
- found: boolean, used to show if the target has been found or not

17.1 set highest to <number of elements in userScores[] array>
17.2 set lowest to 1
17.3 set middleNum to 0
17.4 set found to false

17.5 WHILE found = false and lowest<=highest DO
17.6 set middleNum to trunc((lowest+highest)/2)
17.7 IF userScores[middleNum][username] = targetUser THEN
17.8 set found to true
17.9 SEND targetUser && "has a high score of" && userScores[middleNum]["highScore"] TO DISPLAY
17.10 ELSE IF userScores[middleNum][username] > targetUser THEN
17.11 set highest to middleNum-1
17.12 ELSE
17.13 set lowest to middleNum+1

17.14 END IF
17.15 END WHILE

17.16 IF found = false THEN
17.17 SEND targetUser && "not found in database" TO DISPLAY
17.18 END IF

18. Initialise UI

No variables are used in this procedure, this design ensures that all UI elements on the gameplay card are in the correct position and are showing or not.

18.1 <show field “conditionFld” of card “gameplay”>
18.2 <hide image “result” of card “gameplay”>
18.3 <show image "playerImage" of card "gameplay">
18.4 <show group "hintGroup" of card "gameplay">
18.5 <show group "timeGroup" of card "gameplay">
18.6 <show group "higherGroup" of card "gameplay">
18.7 <show group "lowerGroup" of card "gameplay">
18.8 <set the location of image "playerImage" of card "gameplay" to starting position>

19. Reset Score

Global variables:
- loggedinHighScore: integer, holds the currently logged in user’s high score

Local to card variables:
- currentScore: integer, value that holds the user’s score

19.1 set currentScore to 0
19.2 set <the text of field "scoreFld" of card "gameplay"> to "SCORE: 0"
19.3 set <the text of field "hsFld" of card "gameplay"> to "HIGH SCORE:" && loggedinHighScore
19.4 set <the text of field "timeFld" of card "gameplay"> to "∞"
19.5 set <the foreground color of field "timeFld" of card "gameplay"> to "white"

20. Choose Start Card

Global variables:
- numOfCards: integer, holds the number of records in the cardsArray[] array

Local to card variables:
- player1: integer, holds the index in cardsArray[] for the card that is currently shown to the user

20.1 set player1 to random(numOfCards)
20.2 set <the filename of image "playerImage" of card "gameplay"> to cardsArray[player1]["playerphoto"]

21. Choose Condition

Local to card variables:

- playerCondition: string, this holds the name of the record element that will be compared between player1 (currently shown card) and player2 (the card to be guessed by the user), and can be any of the 6 minor stats (PAC,SHO,DRI,PAS,DEF,PHY) or the card's overall rating (84-91)

Local to procedure variables:

- randNum: integer, holds a random number between 1 and 7
- fieldName: string, holds a more user friendly version of the playerCondition that can be shown on screen

21.1 set randNum to random(7)
21.2 IF randNum = 1 THEN
    21.3 set fieldName to “OVERALL RATING”
    21.4 set playerCondition to “ratingOVR”
21.5 ELSE IF randNum = 2 THEN
    21.6 set fieldName to “PAC”
    21.7 set playerCondition to “ratingPAC”
21.8 ELSE IF randNum = 3 THEN
    21.9 set fieldName to “SHO”
    21.10 set playerCondition to “ratingSHO”
21.11 ELSE IF randNum = 4 THEN
    21.12 set fieldName to “PAS”
    21.13 set playerCondition to “ratingPAS”
21.14 ELSE IF randNum = 5 THEN
    21.15 set fieldName to “DRI”
    21.16 set playerCondition to “ratingDRI”
21.17 ELSE IF randNum = 6 THEN
    21.18 set fieldName to “DEF”
    21.19 set fieldname to “ratingDEF”
21.20 ELSE
    21.21 set fieldName to “PHY”
    21.22 set playerCondition to “ratingPHY”
21.23 END IF

21.24 set <set the htmltext of fld "conditionFld" of card "gameplay"> to "WILL THE <u>" & fieldName & "</u> ON THE NEXT CARD BE HIGHER OR LOWER?"

// this allows the condition to be formatted separately to the rest of the field

22. Choose Next Card

Global variables:

- cardsArray[], array of records that is read in from a csv file, contains information about 122 cards
- numOfCards: integer, holds the number of records in the cardsArray[] array

Local to card variables:

- player2: integer, holds the index in cardsArray[] for the card that is not shown to the user that they have to guess if the rating chosen is higher/lower
- playerCondition: string, this holds the name of the record element that will be compared between player1 (currently shown card) and player2 (the card to be guessed by the user), and can be any of the 6 minor stats (PAC,SHO,DRI,PAS,DEF,PHY) or the card's overall rating (84-91)

Local to procedure variables:

- randNum: integer, holds a random number that is generated

22.1 set randNum to random(numofCards)

22.2 WHILE cardsArray[player1][playerCondition] = cardsArray[randNum][playerCondition] DO *// this line ensures that the stat on each card is not the same, otherwise higher/lower would not work if they have the same value*

22.3 set randNum to random(numOfCards)

22.4 END WHILE

22.5 set player2 to randNum

23. Choose Hints

Global variables:

- cardsArray[], array of records that is read in from a csv file, contains information about 122 cards

Local to card variables:

- player2: integer, holds the index in cardsArray[] for the card that is not shown to the user that they have to guess if the rating chosen is higher/lower

23.1 set <the filename of image "hint1" of card "gameplay"> to cardsArray[player2]["nationphoto"]

23.2 set <the filename of image "hint2" of card "gameplay"> to cardsArray[player2]["clubphoto"]

23.3 set <the text of field "hint3" of card "gameplay"> to cardsArray[player2]["position"]

24. Function to check countdown is running

No parameters are taken in to this function

Function returns a boolean value

Local to card variables:

- sRunning: boolean, stores if the countdown is currently running

24.1 return sRunning

25. Stop Countdown

Local to card variables:

- sRunning: boolean, stores if the countdown is currently running
- sCounter: integer, stores the number of seconds the countdown is currently at

25.1 set sRunning to false
25.2 set sCounter to 0
25.3 Stop counting down

26. Fill Counter
    Parameters:
    - IN: pHowMany

    Local to card variables:
    - sCounter: integer, stores the number of seconds the countdown is currently at

    26.1 set sCounter to pHowMany

27. Start running counter
    Local to card variables:
    - sRunning: boolean, stores if the countdown is currently running
    - sCounter: integer, stores the number of seconds the countdown is currently at

    Local to procedure variables:
    - tWhenToSend: integer, holds the amount of time between counting down each second

    27.1 IF sRunning = true THEN
        27.2 IF sCounter > 0 THEN
            27.3 set sCounter to sCounter-1
            27.4 IF sCounter <=3 THEN
                27.5 set <the foreground color of field “timeFld”> to ‘red’
            27.6 ELSE
                27.7 set <the foreground color of field “timeFld”> to ‘white’
            27.8 END IF
            27.9 set <the text of field “timeFld”> to sCounter
            27.10 set tWhenToSend to 1000
            27.11 IF “doCountDown” <is not in the pendingMessages> THEN
                27.12 SEND “doCountDown” to me in tWhenTosSend milliseconds
            27.13 END IF
        27.14 ELSE
            27.15 hide field "conditionFld"
            27.16 hide group "hintGroup"
            27.17 hide group "higherGroup"
            27.18 hide group "lowerGroup"
            27.19 hide group "timeGroup"
            27.20 hide image "playerImage"
            27.21 endGame "YOU RAN OUT OF TIME", "lightblue" (see design 29)
        27.22 END IF
    27.23 END IF

28. Submit Guess

Parameters:

- IN: choice (either “HIGHER” or “LOWER”)

Global variables:

- cardsArray[], array of records that is read in from a csv file, contains information about 122 cards

Local to card variables:

- player1: integer, holds the index in cardsArray[] for the card that is currently shown to the user
- player2: integer, holds the index in cardsArray[] for the card that is not shown to the user that they have to guess if the rating chosen is higher/lower
- playerCondition: string, this holds the name of the record element that will be compared between player1 (currently shown card) and player2 (the card to be guessed by the user), and can be any of the 6 minor stats (PAC,SHO,DRI,PAS,DEF,PHY) or the card’s overall rating (84-91)
- currentScore: integer, value that holds the user’s score

Local to procedure variables:

- correct: boolean, used to show if the users guess was correct or not

28.1 IF choice = “HIGHER’ AND cardsArray[player2][playerCondition] > cardsArray[player1][playerCondition] THEN

28.2 set correct to true

28.3 ELSE IF choice = "LOWER" and cardsArray[player2][playerCondition] < cardsArray[player1][playerCondition] THEN

28.4 set correct to true

28.5 ELSE

28.6 set correct to false

28.7 END IF

28.8 hide field "conditionFld"
28.9 hide group "hintGroup"
28.10 hide group "higherGroup"
28.11 hide group "lowerGroup"
28.12 hide group "timeGroup"
28.13 hide image "playerImage"

28.14 IF correct = true THEN

28.15 set <the filename of image "result"> to "images/tick.png"
28.16 show image "result"

28.17 set player1 to player2
28.18 Choose Condition (see design 21)

28.19 Choose Next Card (see design 22)
28.20 Choose Hints (see design 23)
28.21 set currentScore to currentScore+1
28.22 set <the text of field "scoreFld"> to "SCORE:" && currentScore

28.23 show image "playerImage"
28.24 show group "hintGroup"
28.25 show group "higherGroup"
28.26 show group "lowerGroup"
28.27 show group "timeGroup"
28.28 show field "conditionFld"
28.29 hide image "result"
28.30 ELSE
28.31 IF currentScore = 0 THEN
28.32 endGame "NICE TRY!!!", "lightblue" (see design 29)
28.33 ELSE
28.34 endGame (see design 29)
28.35 END IF
28.36 END IF

29. End Game

Parameters:

- IN: message, msgColor

Global variables:

- cardsArray[], array of records that is read in from a csv file, contains information about 122 cards
- loggedinUsername: string, stores the name of the user currently logged in
- loggedinHighScore: integer, scores the high score of the user currently logged in

Local to card variables:

- player2: integer, holds the index in cardsArray[] for the card that is not shown to the user that they have to guess if the rating chosen is higher/lower
- currentScore: integer, value that holds the user's score

29.1 IF <Check if countdown is running (see design 24)> THEN
29.2 Stop countdown (see design 25)
29.3 END IF

29.4 set <the filename of image "result"> to "images/cross.png"
29.5 show image "result"
29.6 wait .5 seconds
29.7 hide image "result"
29.8 set <the location of image "playerImage"> to 223,285
29.9 set <the filename of image "playerImage"> to cardsArray[player2]["playerphoto"]

29.10 show image "playerImage"
29.11 wait 2 seconds

29.12 set <the foreground color of field "messageFld" of card "gameOver"> to msgColor
29.13 set <the text of field "messageFld" of card "gameOver"> to message

29.14 IF currentScore > loggedinHighScore THEN
29.15 set <the foreground color of field "messageFld"> of card "gameOver" to "green"
29.16 set <the text of field "messageFld" of card "gameOver"> to "NEW HIGH SCORE ACHIEVED"
29.17 set loggedinHighScore to currentScore
29.18 IF loggedinUsername NOT “” THEN
29.19 Open Database (see design 1)
29.20 Update Highscore in Database (see design 30)
29.21 Close database (see design 8)
29.22 END IF
29.23 END IF

29.24 set <the text of field "scoreFld" of card "gameOver"> to "SCORE:" && currentScore & cr & "HIGH SCORE:" && loggedinHighScore
29.25 <Switch to game over screen>

30. Update Highscore in database

Global variables:

- loggedinUsername: string, stores the name of the user currently logged in
- loggedinHighScore: integer, scores the high score of the user currently logged in

Local to procedure variables:

- dbID: integer, identifier used for database
- sql: string, sql code to be executed

30.1 set sql to “UPDATE players SET highscore = " & loggedinHighScore && "WHERE username ='" & loggedinUsername & "';"
30.2 revExecuteSQL dbID, sql

Database design

Data dictionary

The database table initially has zero records

Entity: Players					
Attribute	**Key**	**Type**	**Required**	**Size**	**Validation**
username	PK	varchar(25)	yes	20	Length >= 3 and length =< 15
password		varchar(25)	yes	25	Length >= 8 and length =< 20
highscore		integer	no	3	

Query design

Values between angle brackets <> are variables from the program

1. Check if username is in database
   Used in pseudocode design 5 and 10

Field(s)	username
Table(s)	players
Search Criteria	Username = <usernameToFind/desiredUsername variable>
Order	

2. Check if username matches password
   Used in pseudocode design 11

Field(s)	password
Table(s)	players
Search Criteria	Username = <usernameToFind

	variable> and password = <passwordToFind> variable
Order	

3. Retrieve high score from database
   Used in pseudocode design 11

Field(s)	highscore
Table(s)	players
Search Criteria	Username = <usernameToFind variable>
Order	

4. Retrieve usernames and highscores from database
   Used in pseudocode design 12

Field(s)	username, highscore
Table(s)	players
Search Criteria	
Order	

5. Insert new record into database
   Used in pseudocode design 7

Field(s)	username, password, highscore
Table(s)	players
Value(s)	<username>,<password>,0

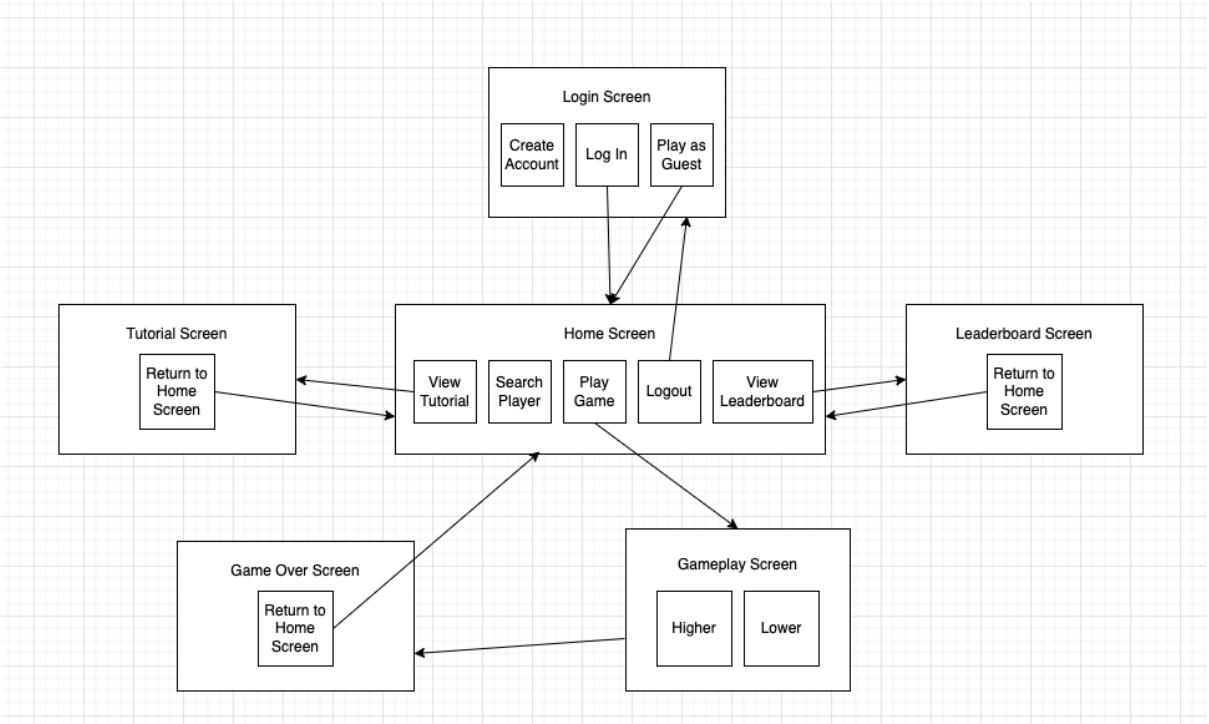
6. Update high score in database
   Used in design 30

Field(s)	highscore
Table(s)	players

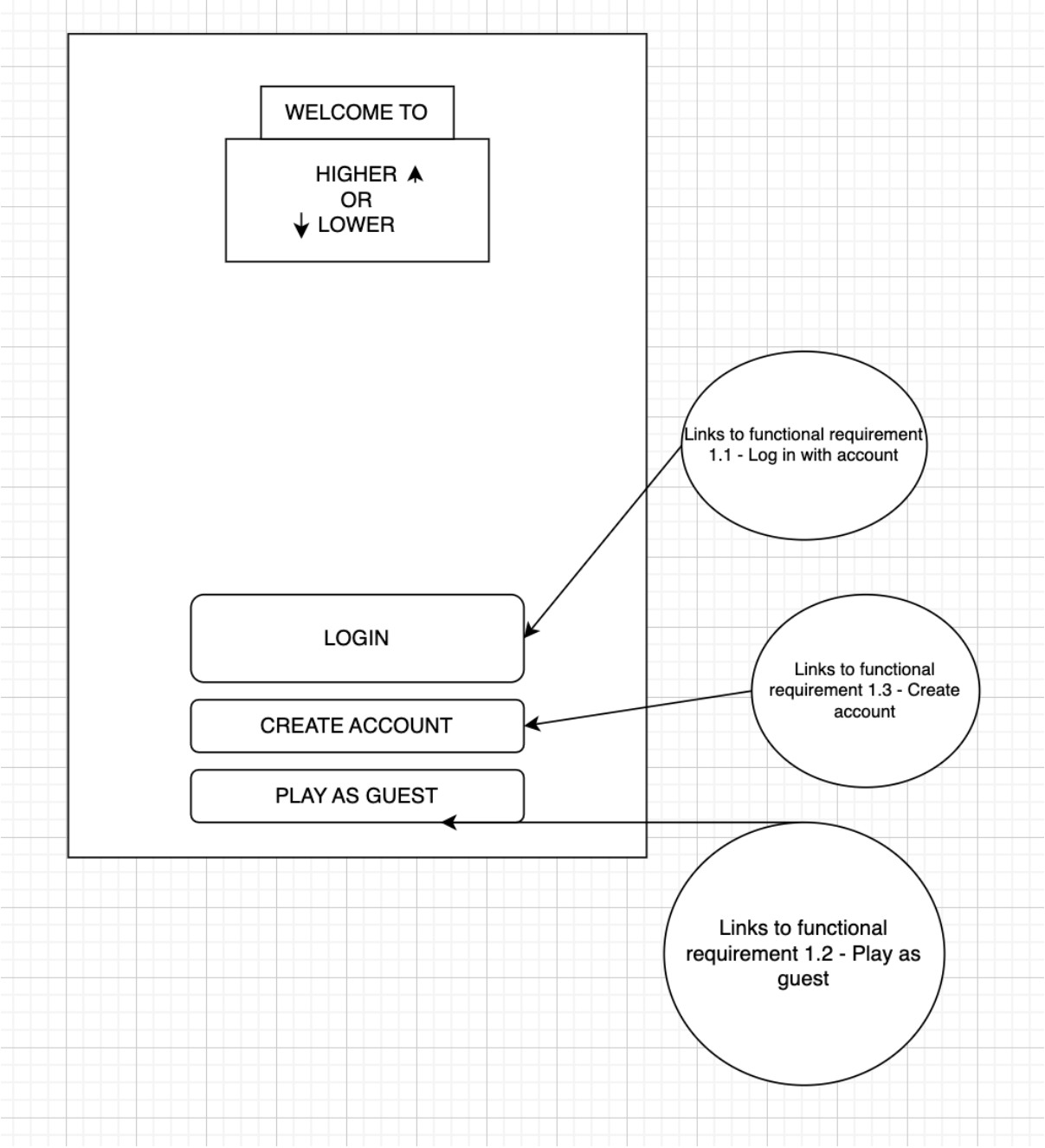
Search Criteria	Username = <username>
Value(s)	Highscore = <new highscore>

User interface design

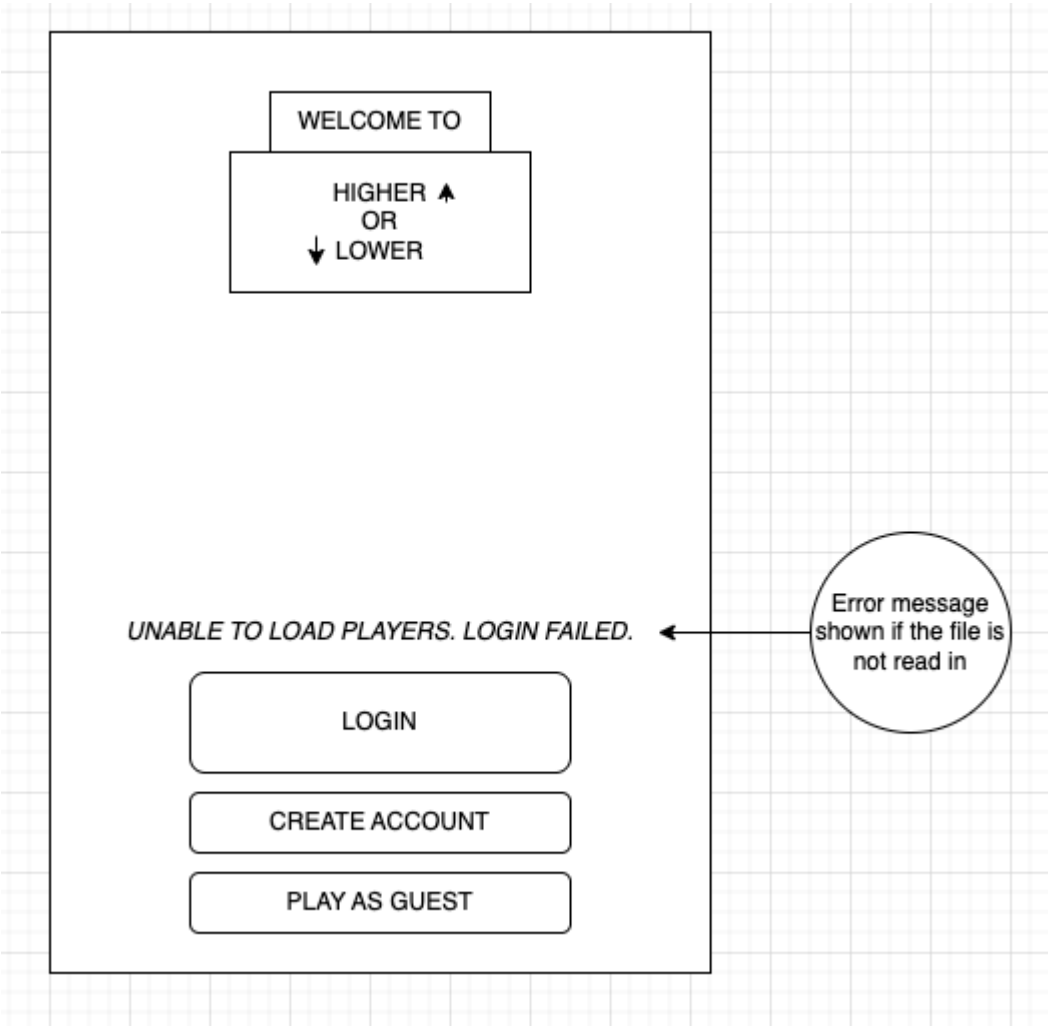
Map of the game



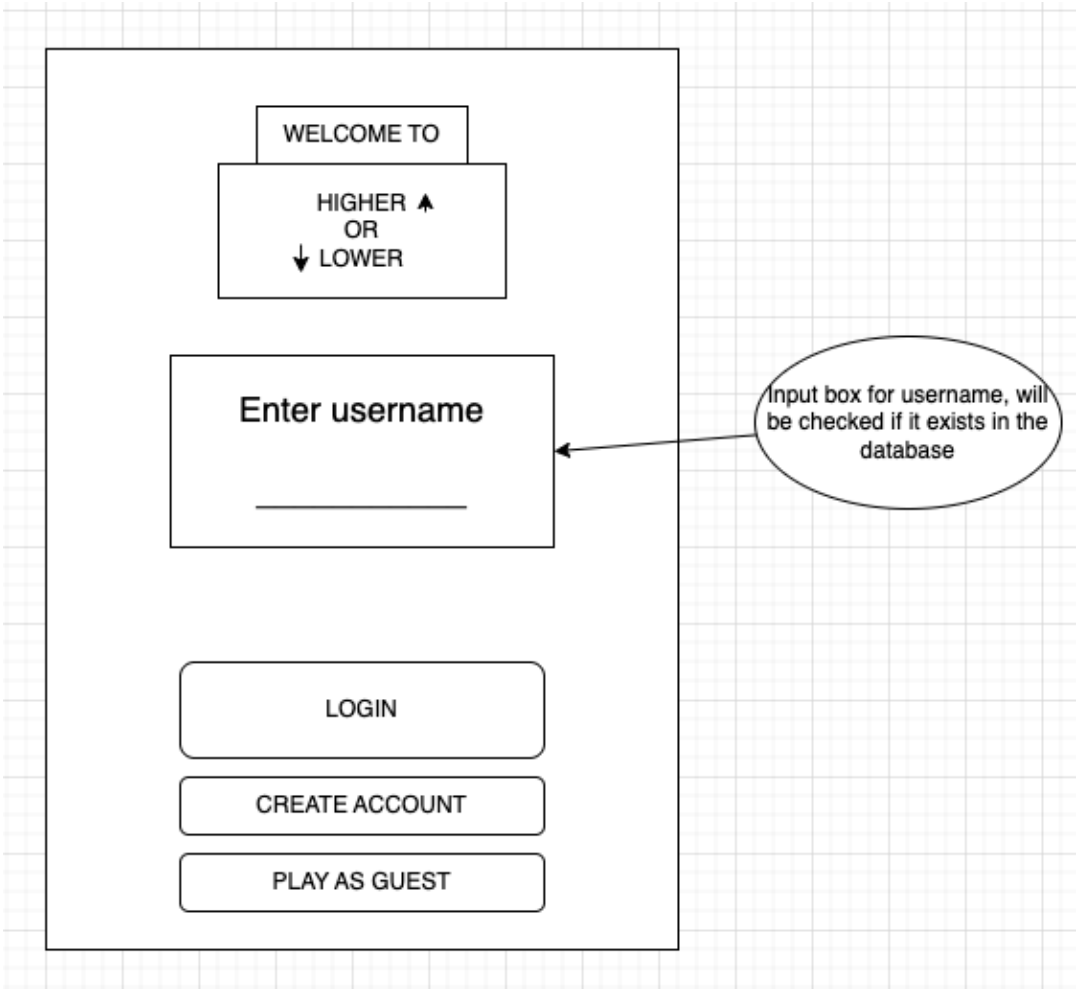
Login page

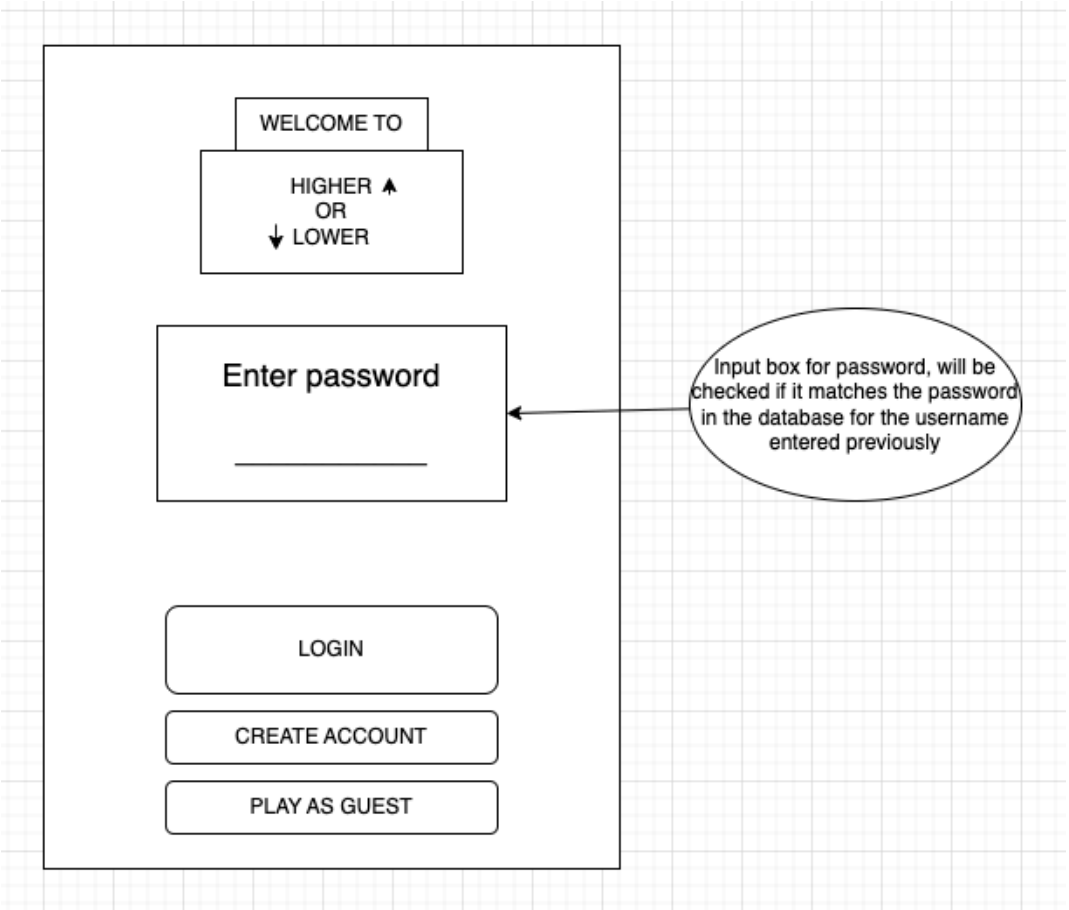

WELCOME TO
HIGHER
OR
LOWER
LOGIN
CREATE ACCOUNT
PLAY AS GUEST
Links to functional requirement 1.1 - Log in with account
Links to functional requirement 1.3 - Create account
Links to functional requirement 1.2 - Play as guest

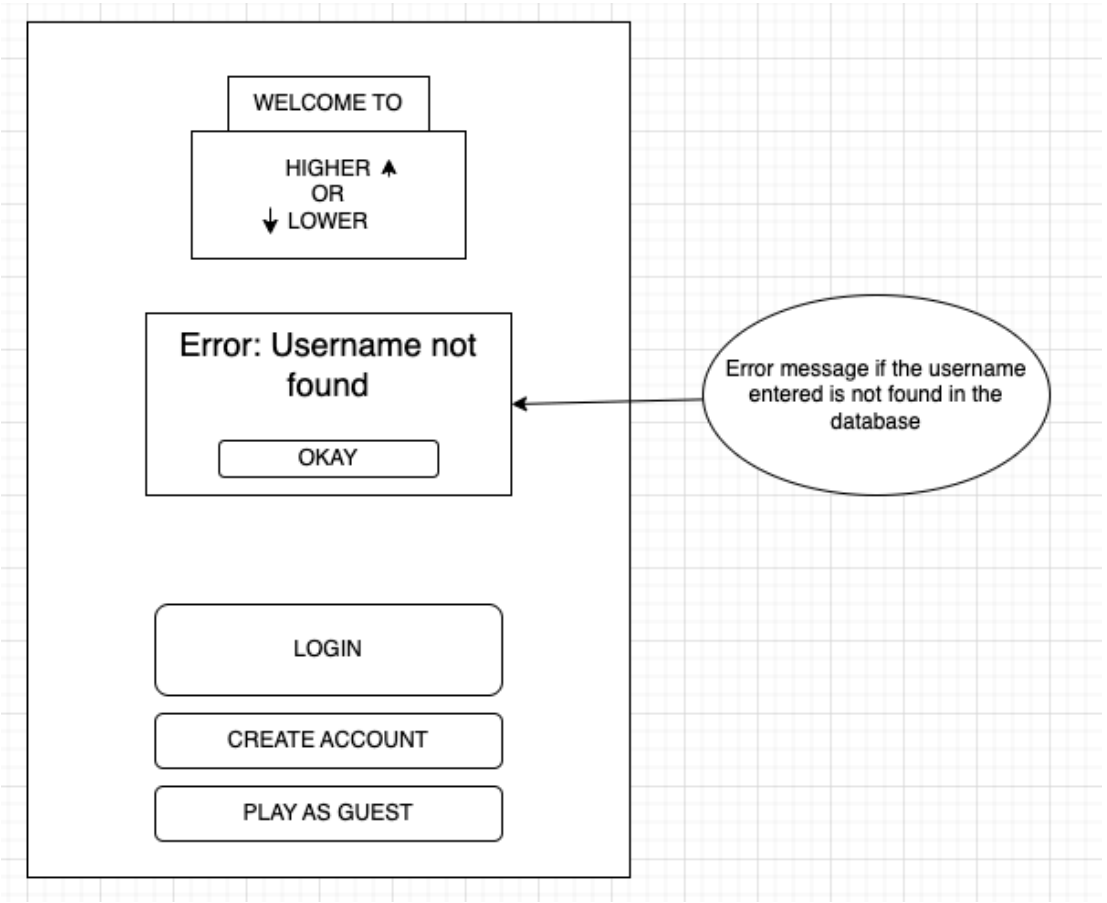
Error message if file fails to load

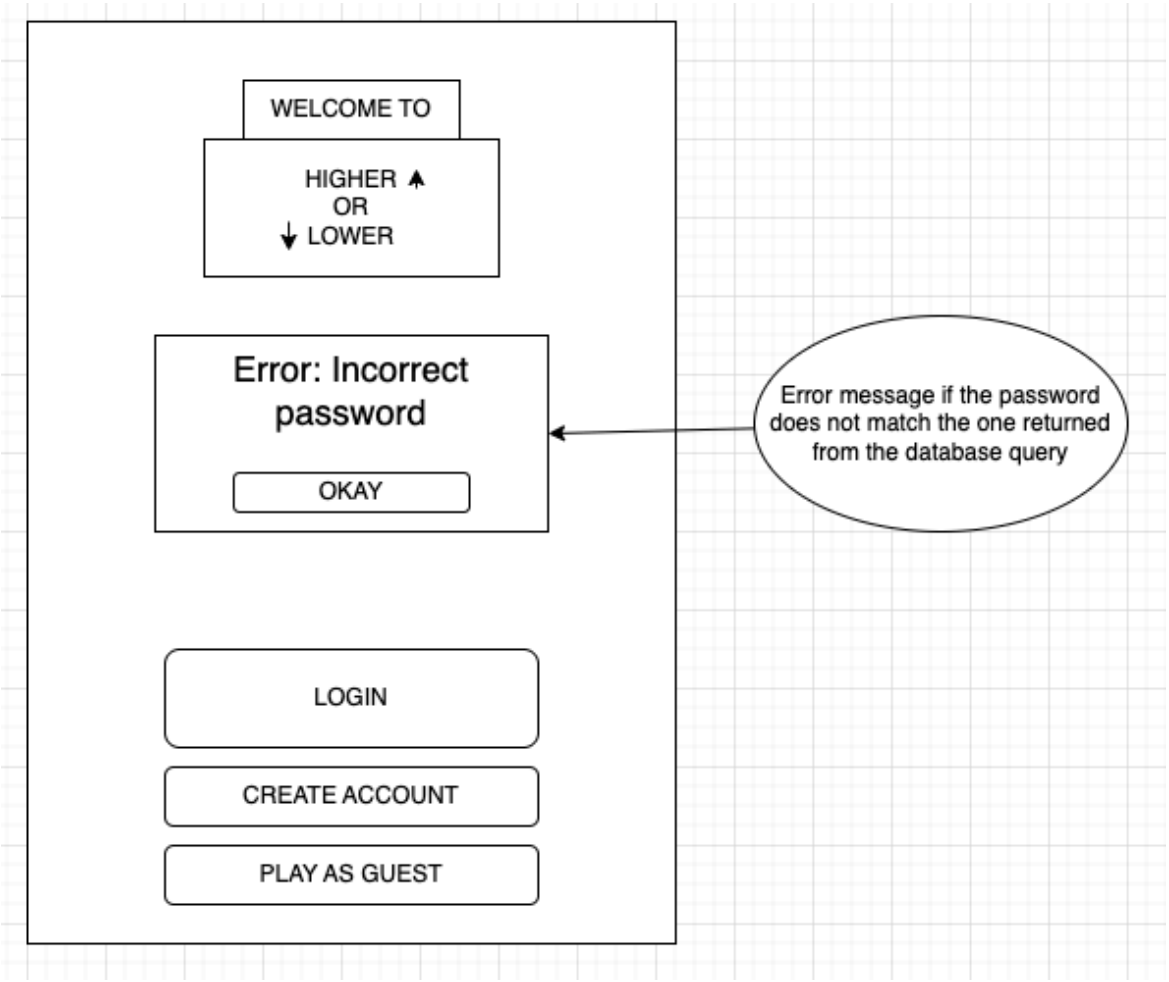


When ‘Login’ button is pressed:

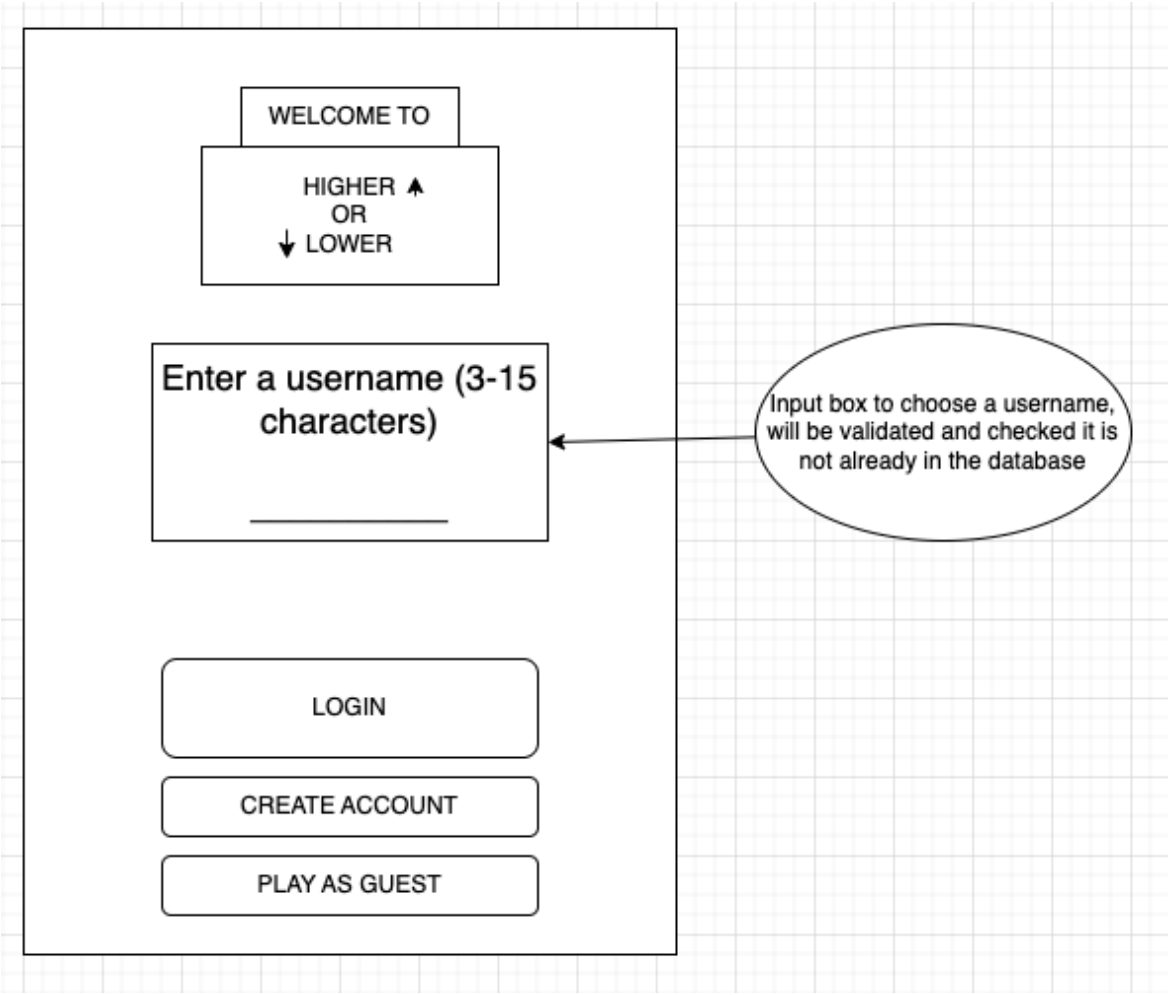


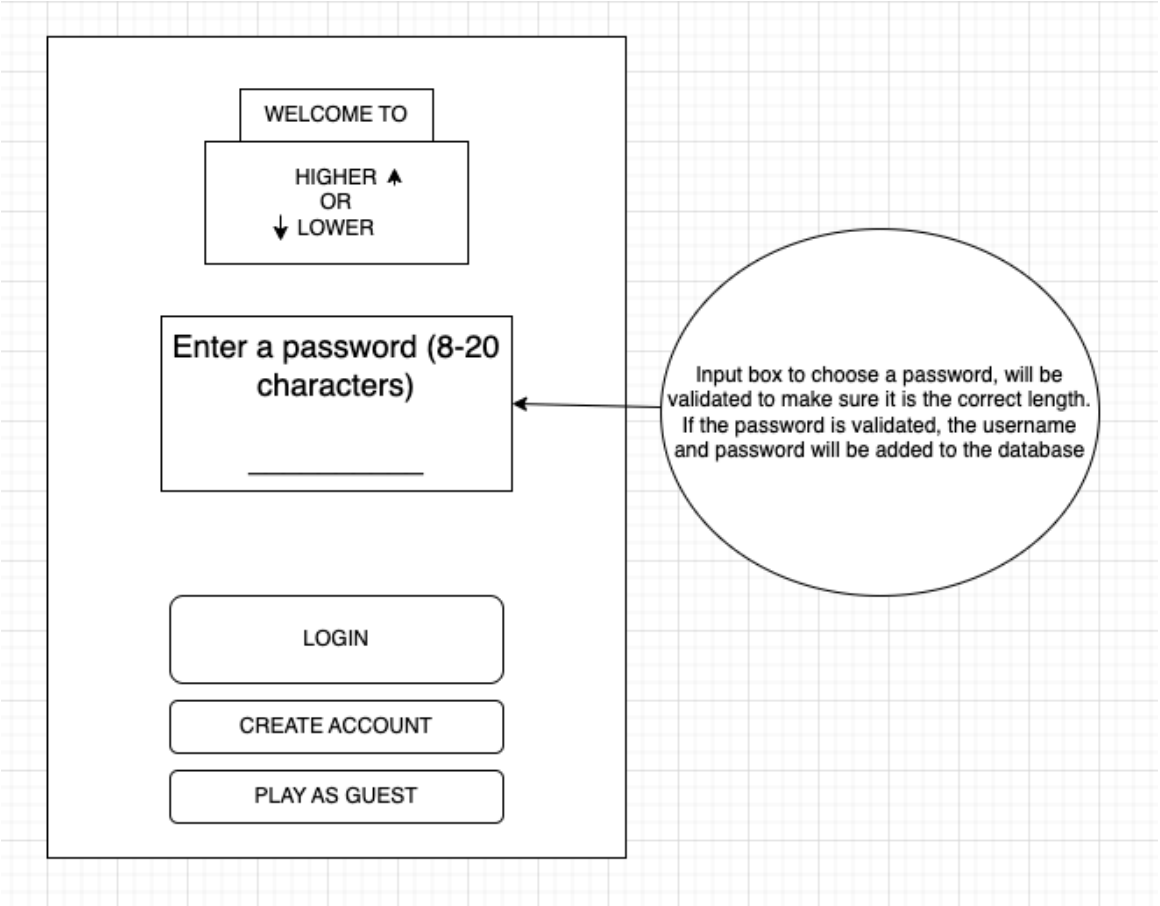

WELCOME TO
HIGHER
OR
LOWER
Enter password
Input box for password, will be checked if it matches the password in the database for the username entered previously
LOGIN
CREATE ACCOUNT
PLAY AS GUEST

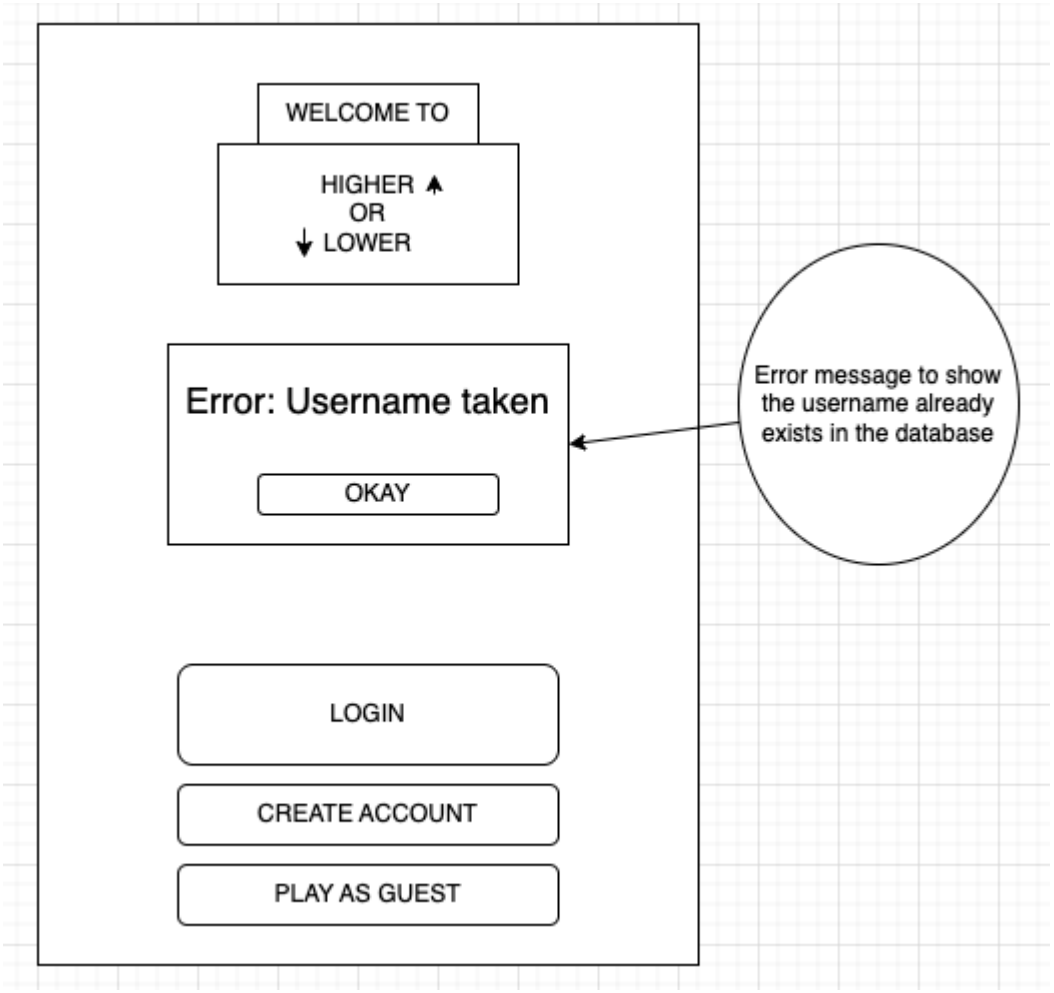

WELCOME TO
HIGHER
OR
LOWER
Error: Username not found
OKAY
Error message if the username entered is not found in the database
LOGIN
CREATE ACCOUNT
PLAY AS GUEST

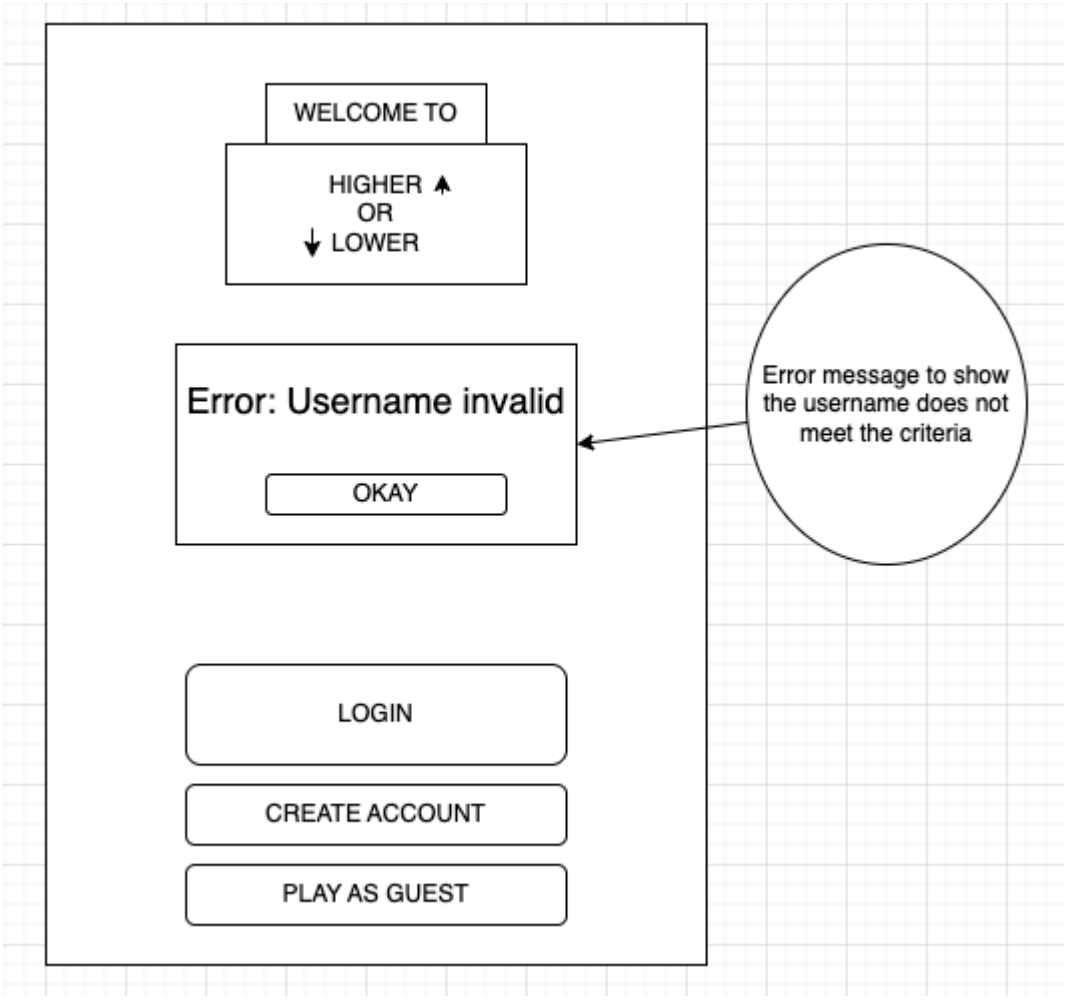

WELCOME TO
HIGHER
OR
LOWER
Error: Incorrect password
OKAY
Error message if the password does not match the one returned from the database query
LOGIN
CREATE ACCOUNT
PLAY AS GUEST

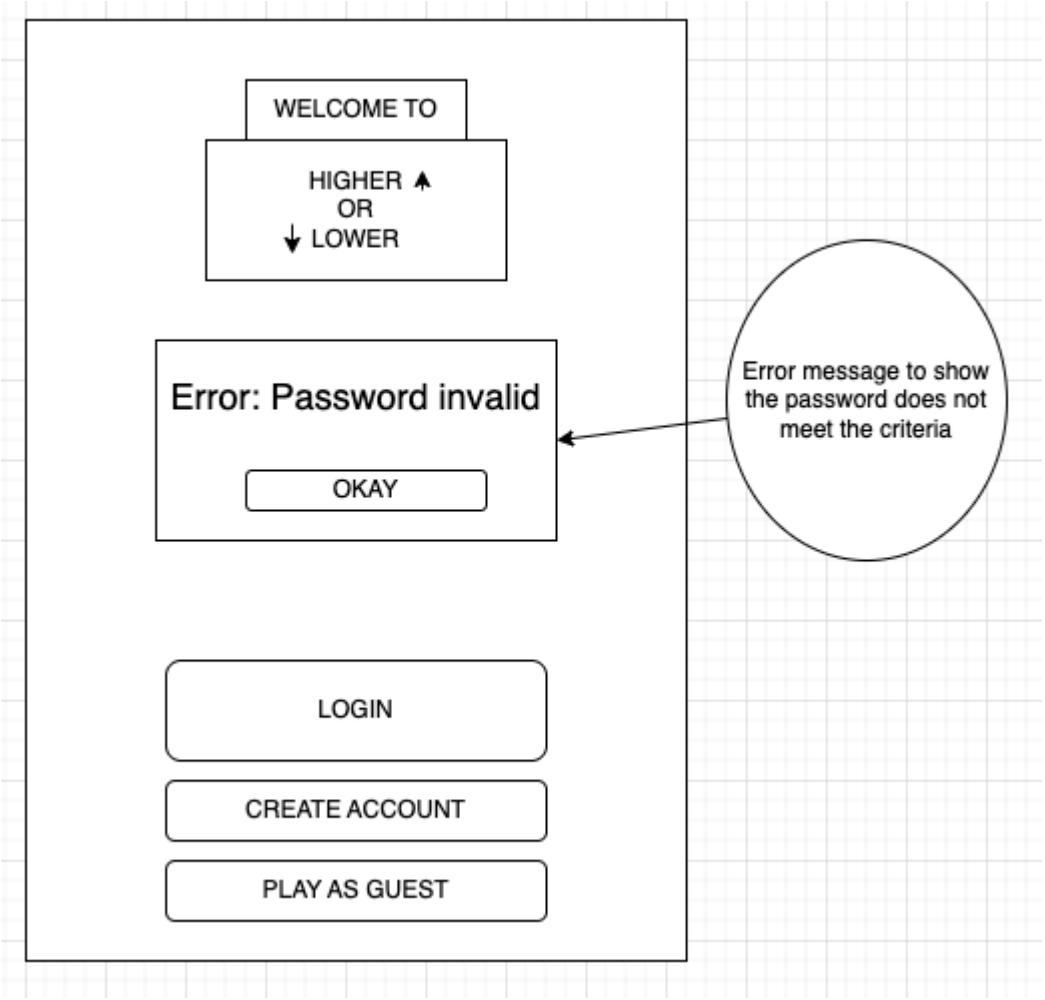
When ‘Create Account’ button is pressed:

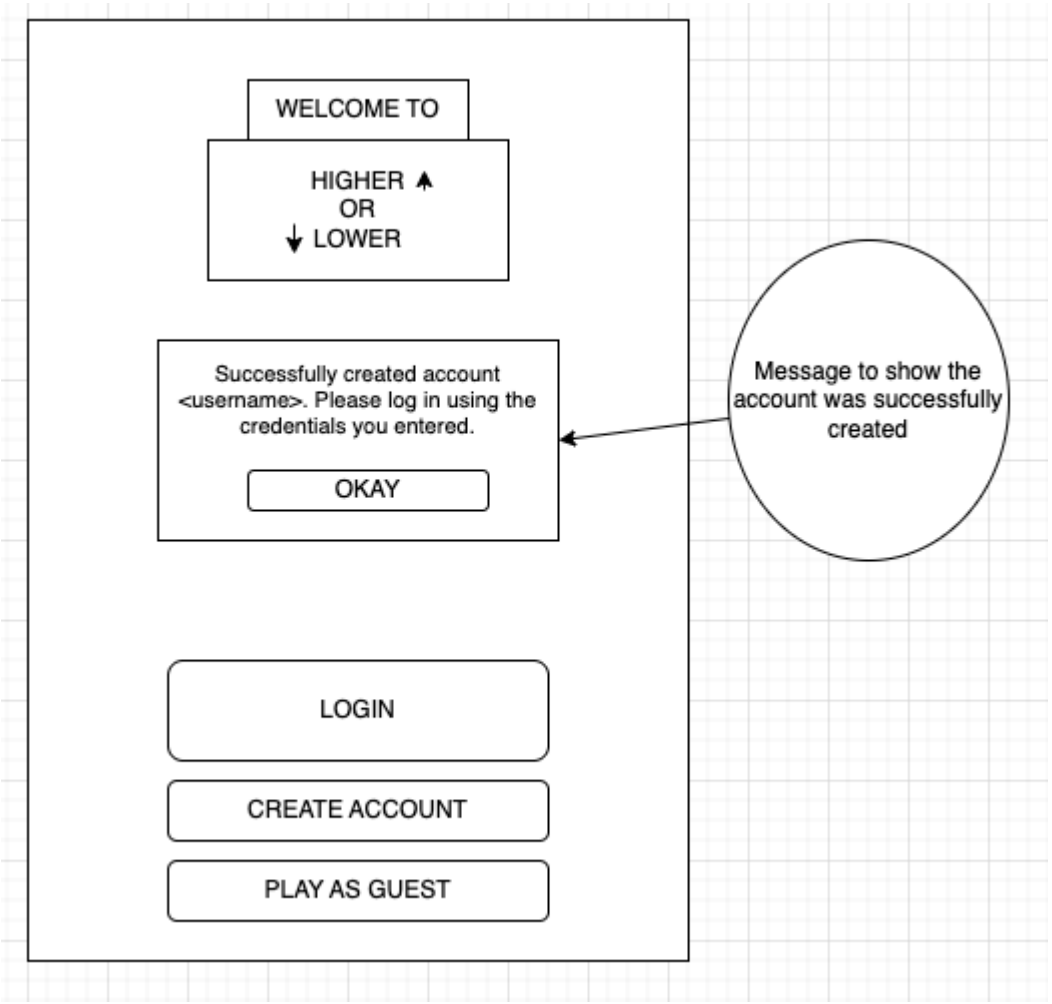



WELCOME TO
HIGHER
OR
LOWER
Enter a password (8-20 characters)
LOGIN
CREATE ACCOUNT
PLAY AS GUEST
Input box to choose a password, will be validated to make sure it is the correct length. If the password is validated, the username and password will be added to the database

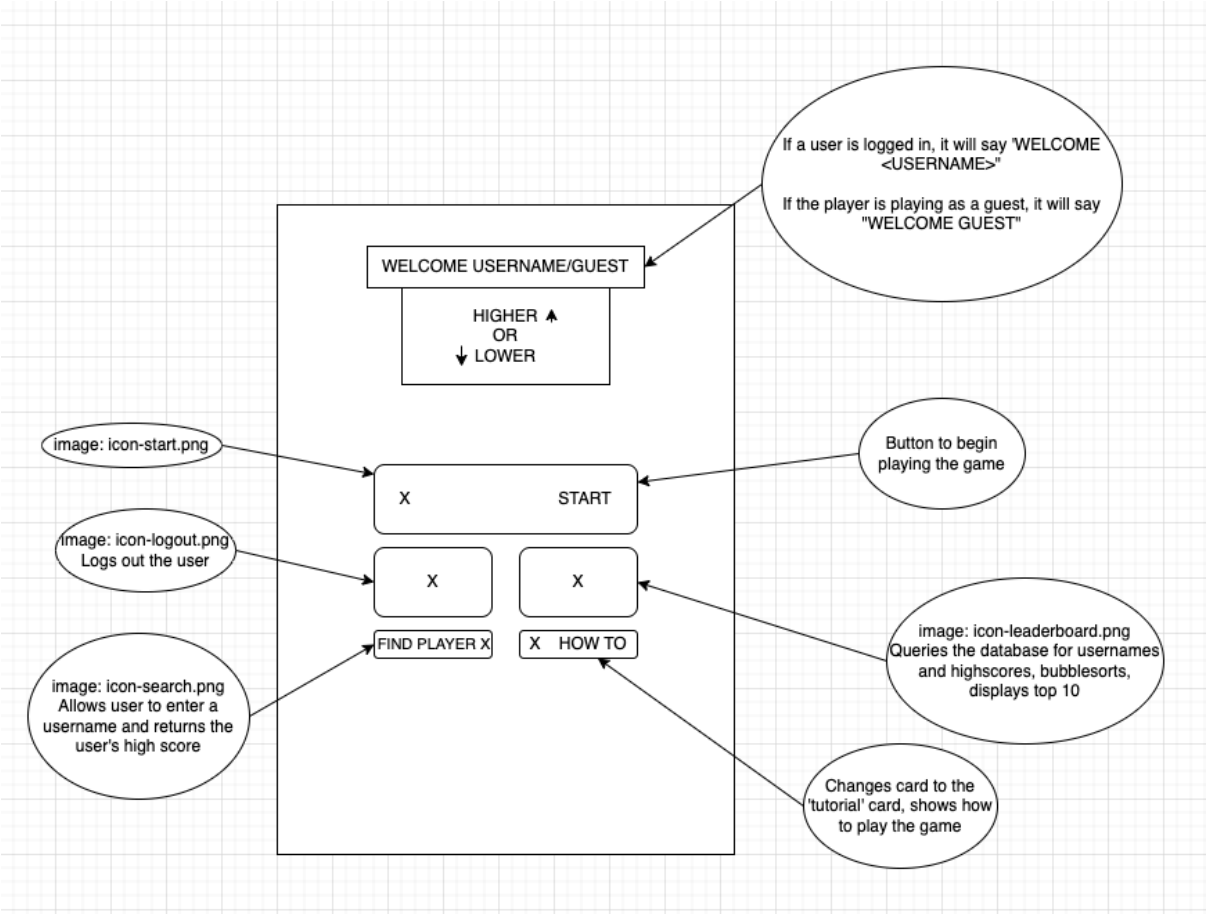

WELCOME TO
HIGHER
OR
LOWER
Error: Username taken
OKAY
Error message to show the username already exists in the database
LOGIN
CREATE ACCOUNT
PLAY AS GUEST


WELCOME TO
HIGHER
OR
LOWER
Error: Username invalid
OKAY
Error message to show the username does not meet the criteria
LOGIN
CREATE ACCOUNT
PLAY AS GUEST

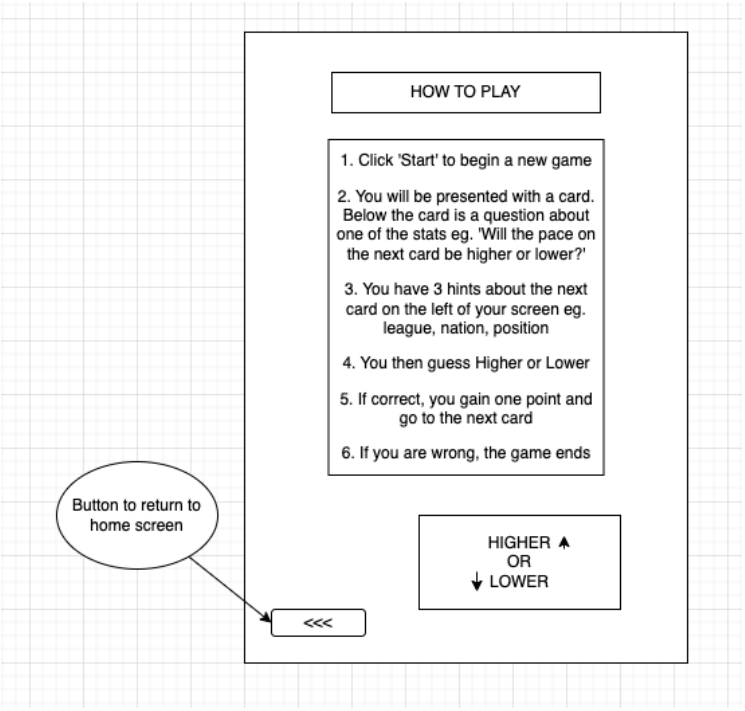

WELCOME TO
HIGHER
OR
LOWER
Error: Password invalid
OKAY
Error message to show the password does not meet the criteria
LOGIN
CREATE ACCOUNT
PLAY AS GUEST


WELCOME TO
HIGHER
OR
LOWER
Successfully created account <username>. Please log in using the credentials you entered.
OKAY
Message to show the account was successfully created
LOGIN
CREATE ACCOUNT
PLAY AS GUEST

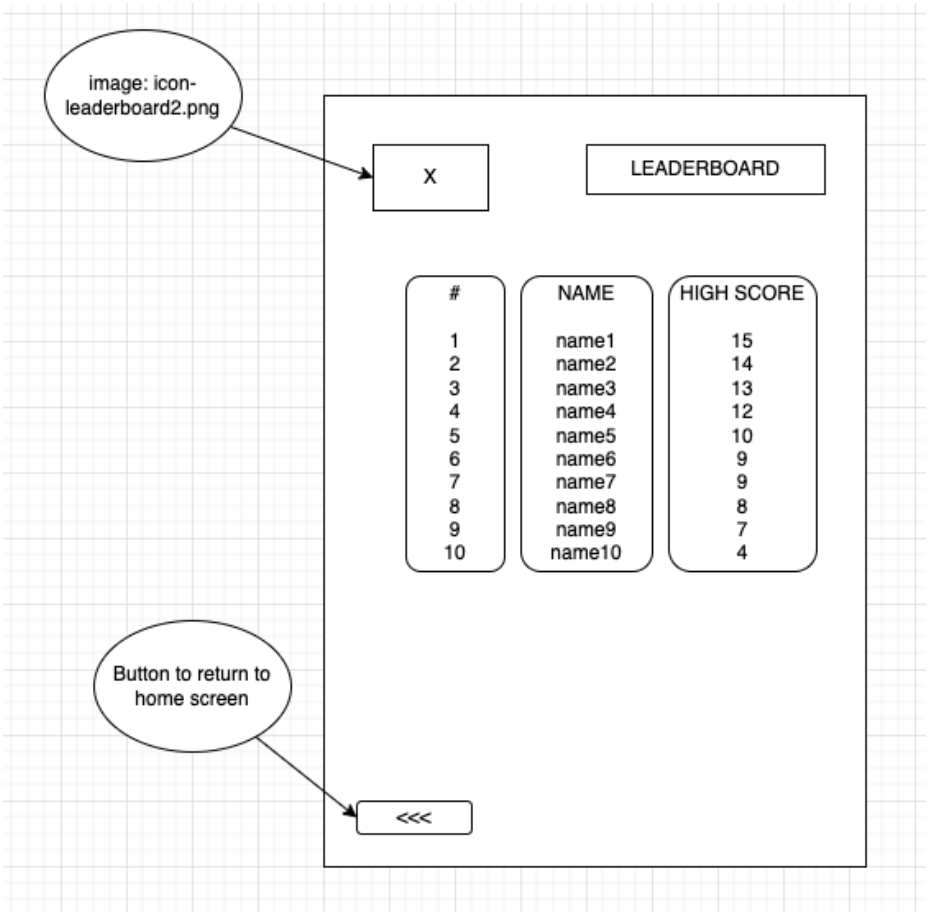
Home page



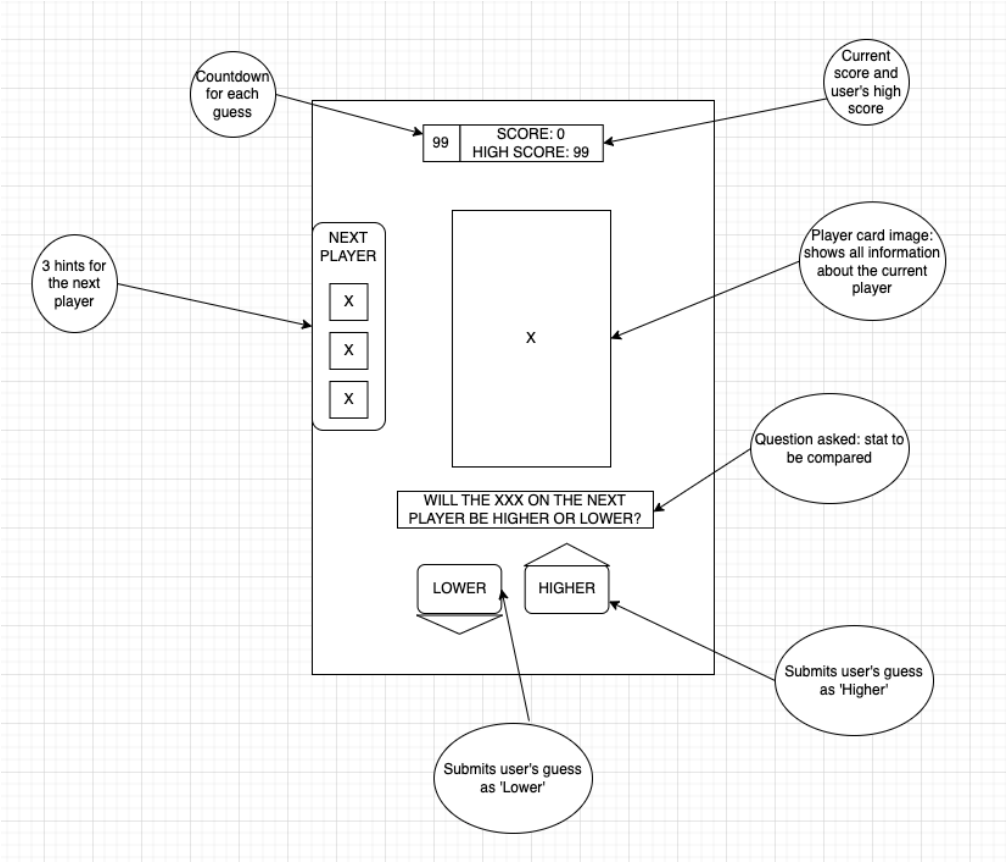
Tutorial page

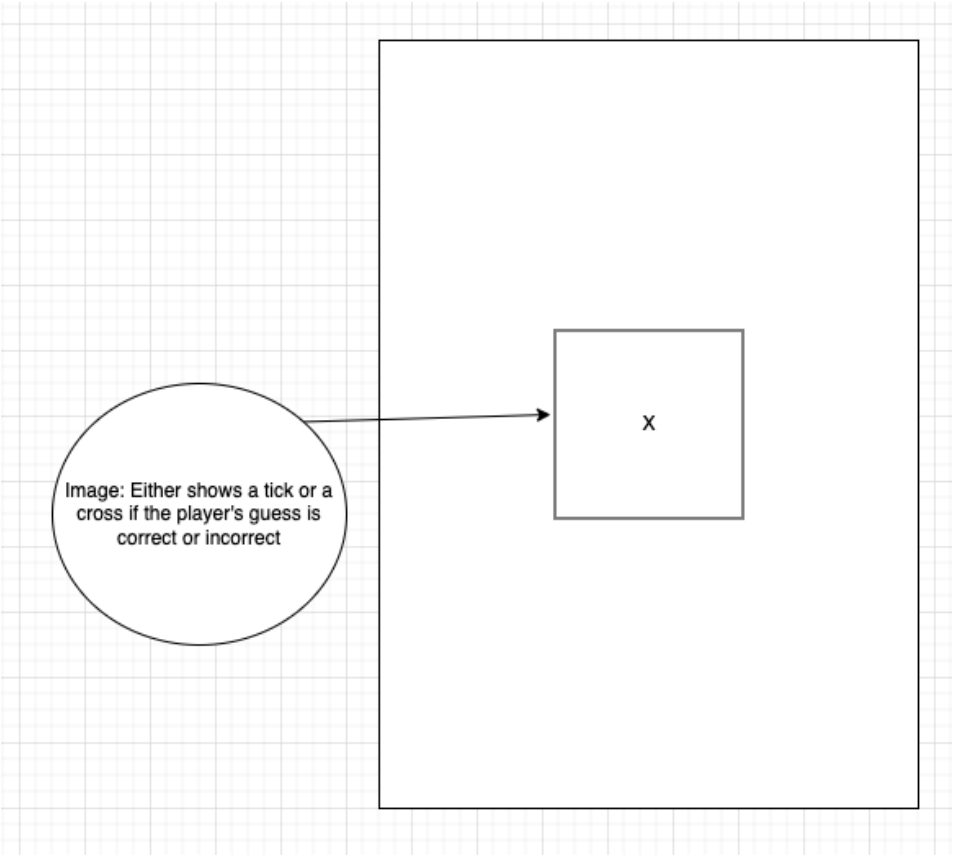


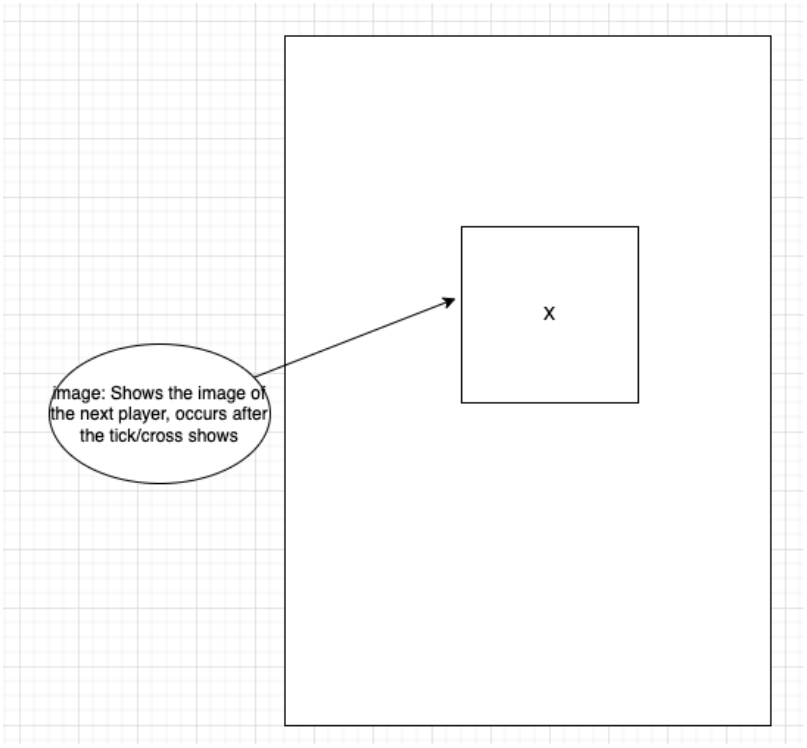
Leaderboard page



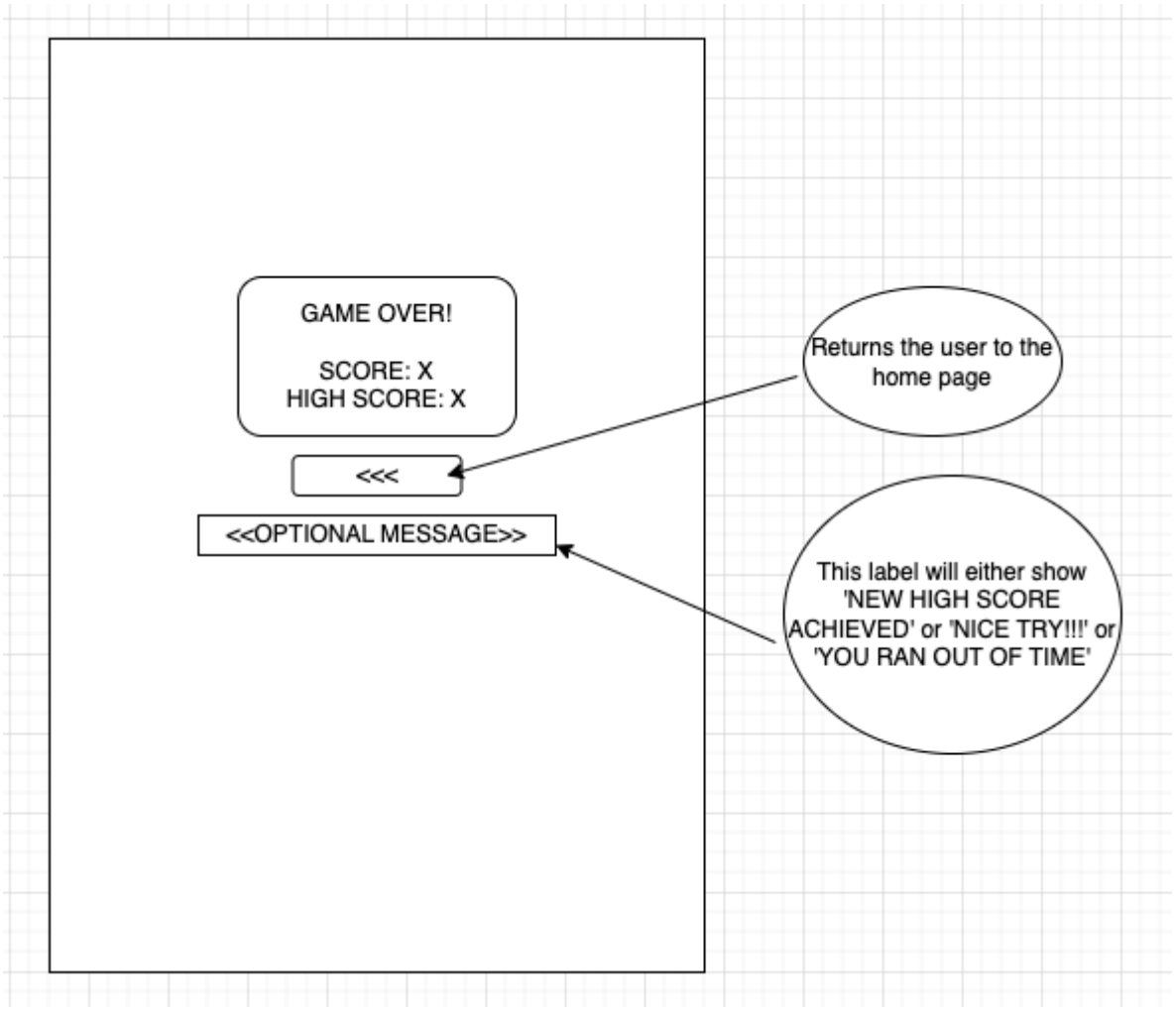
Main game page






X
image: Shows the image of the next player, occurs after the tick/cross shows

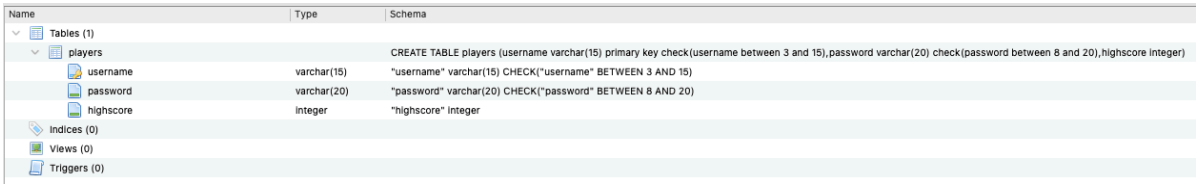
Game over screen



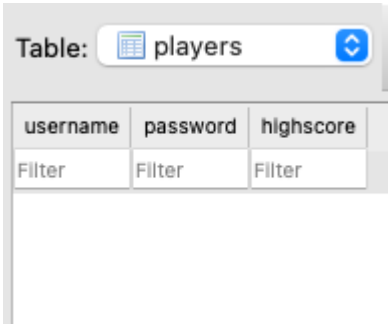
IMPLEMENTATION

Program code can be found in appendix 1

Database Structure

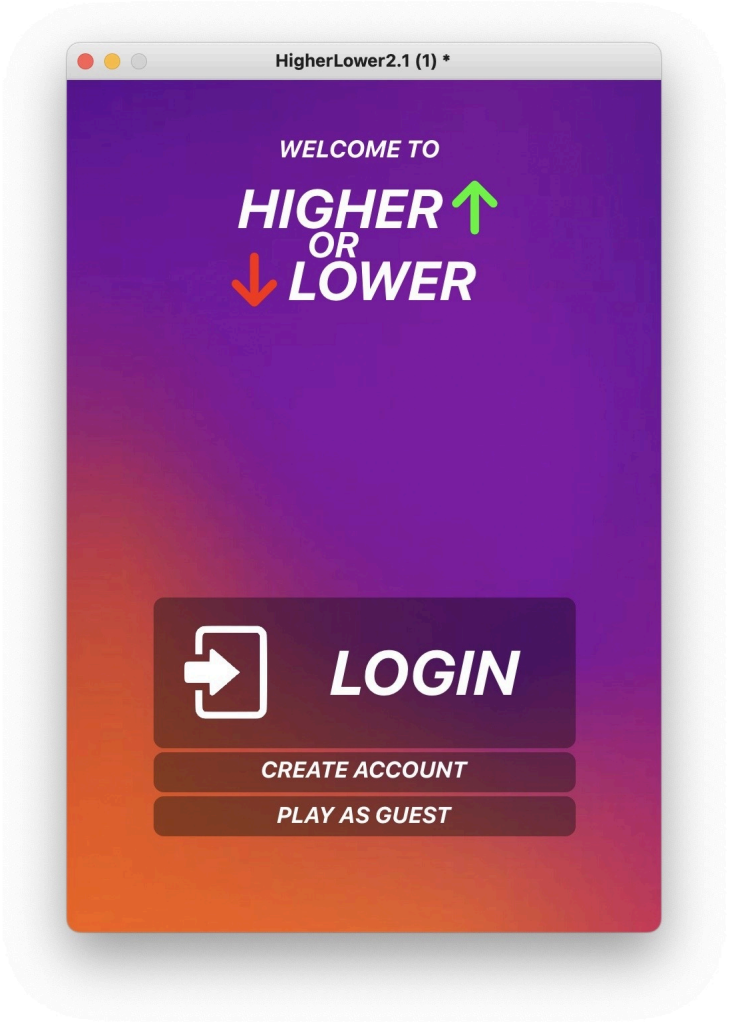


Initial database data

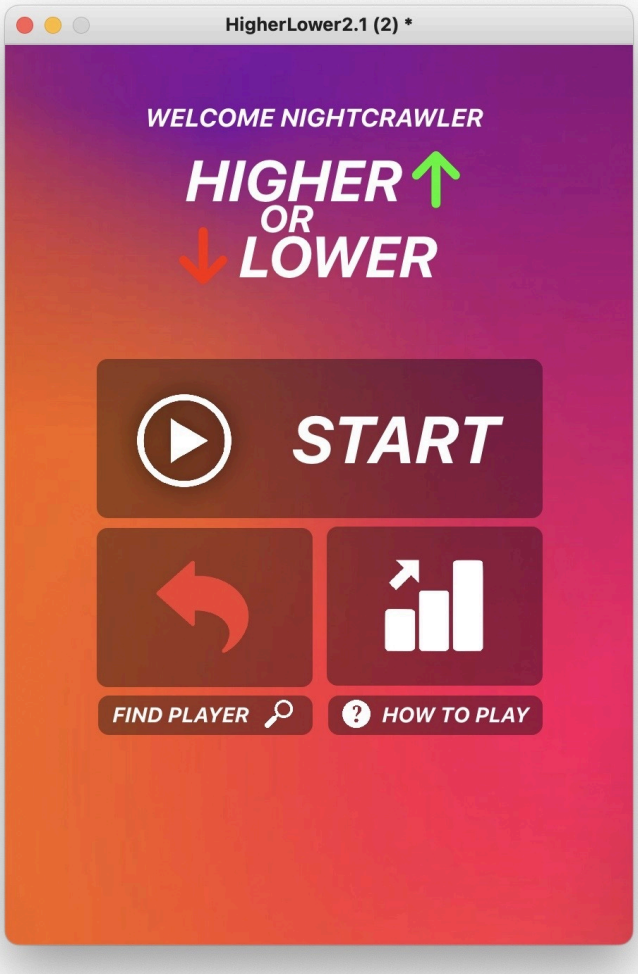


Initially, the database is empty however I had some test data in my database

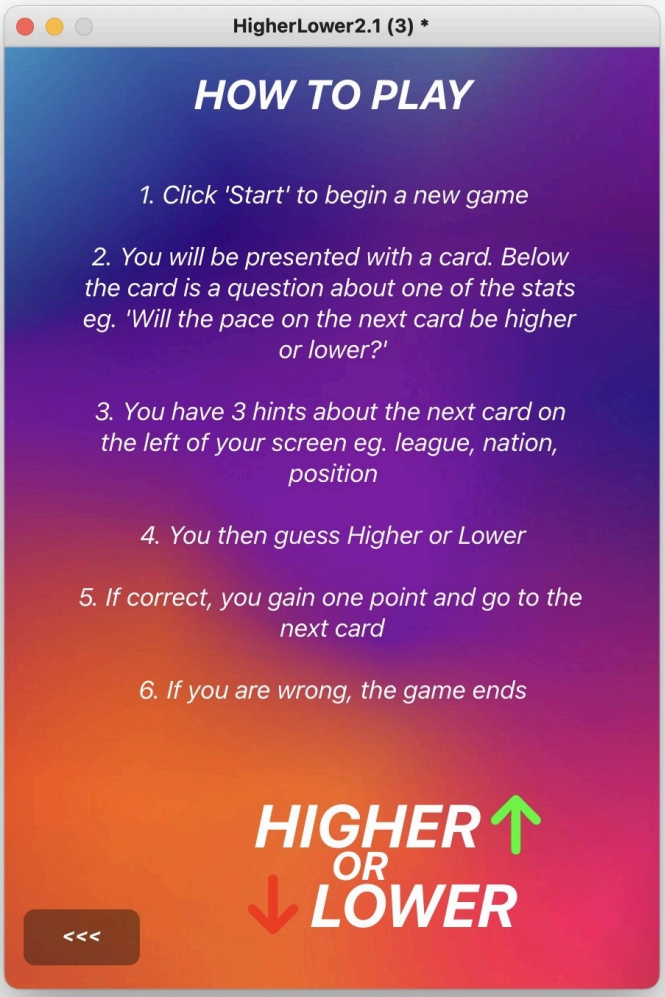
Login Screen



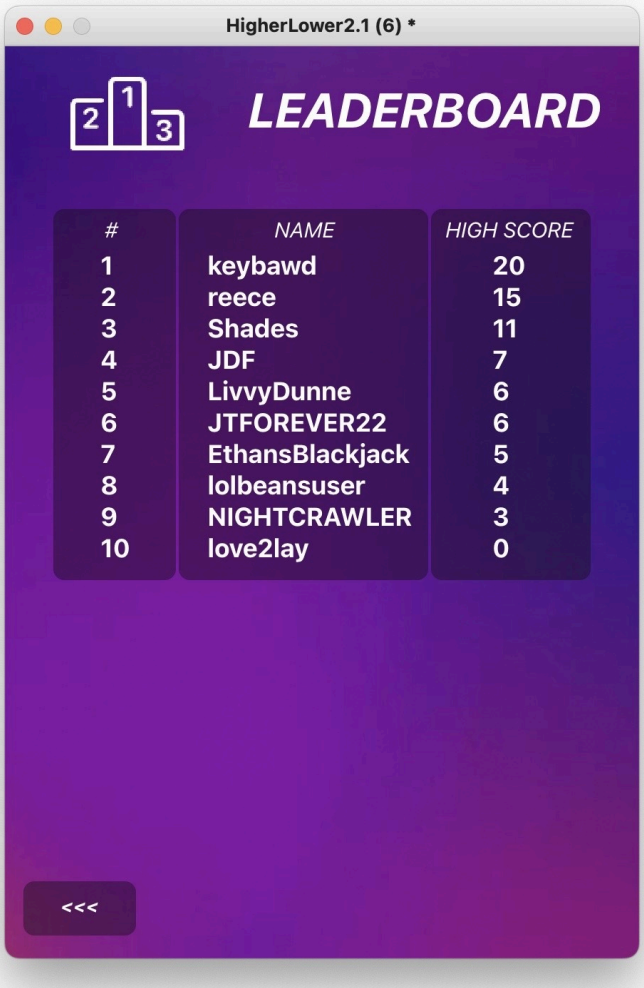
Home Screen


HigherLower2.1 (2) *
WELCOME NIGHTCRAWLER
HIGHER
OR
LOWER
START
FIND PLAYER
HOW TO PLAY

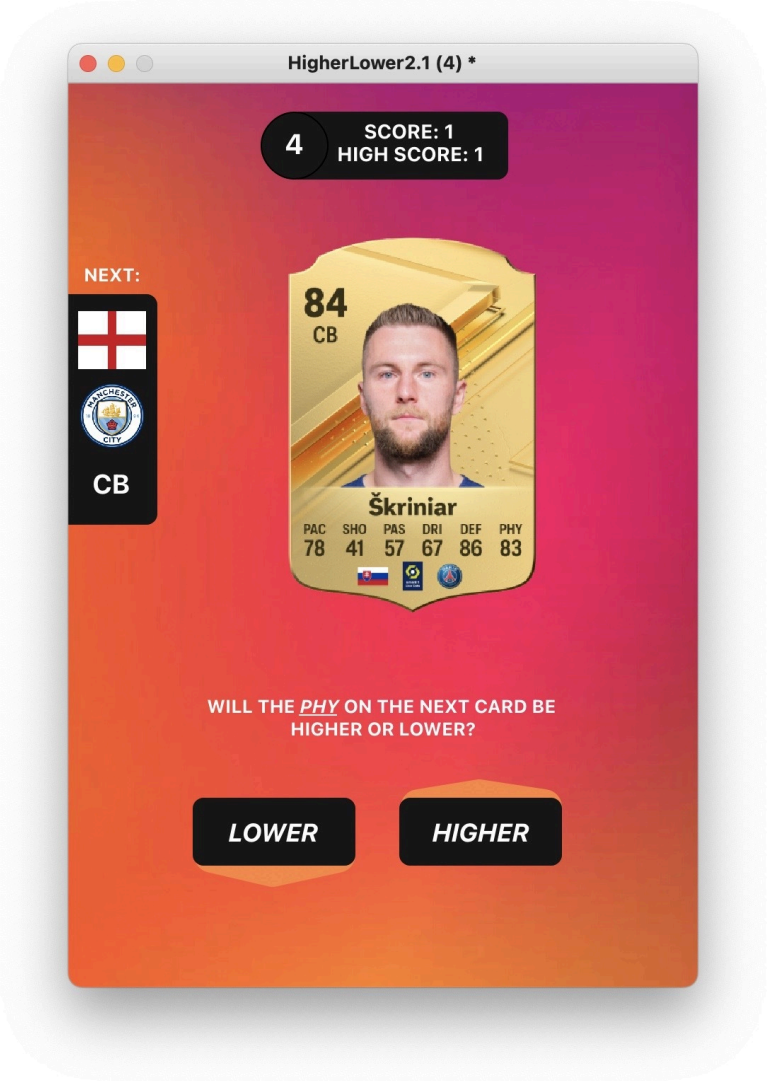
Tutorial Screen



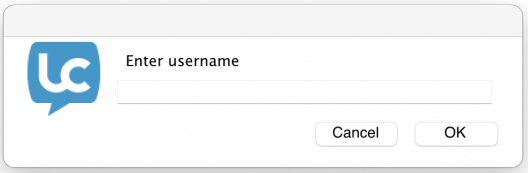
Leaderboard Screen with sample data



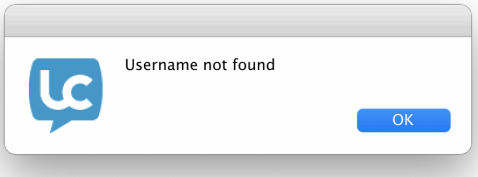
Gameplay Screen



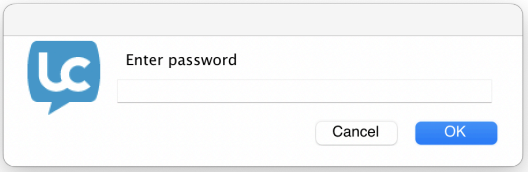
Input to enter username



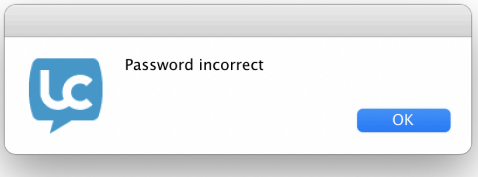
Error for no such username



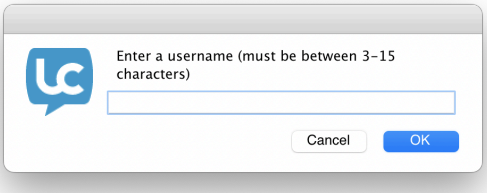
Input to enter password



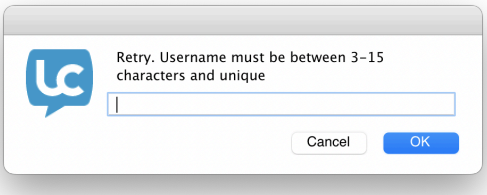
Error message for incorrect password



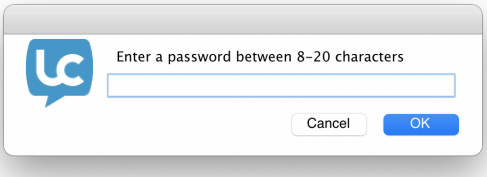
Input to choose a desired username



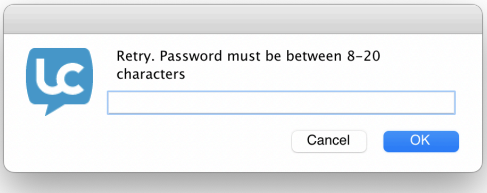
Error/re-input of desired username



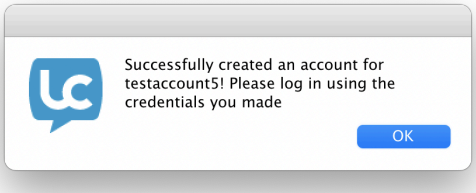
Input to choose a desired password



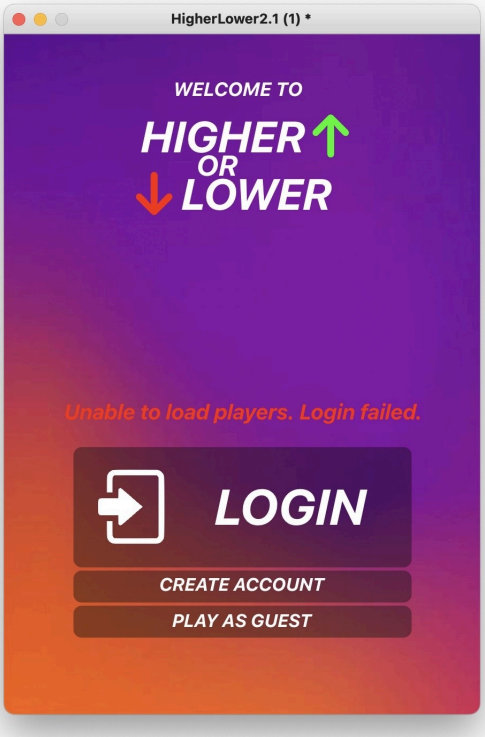
Error/re-input of desired password



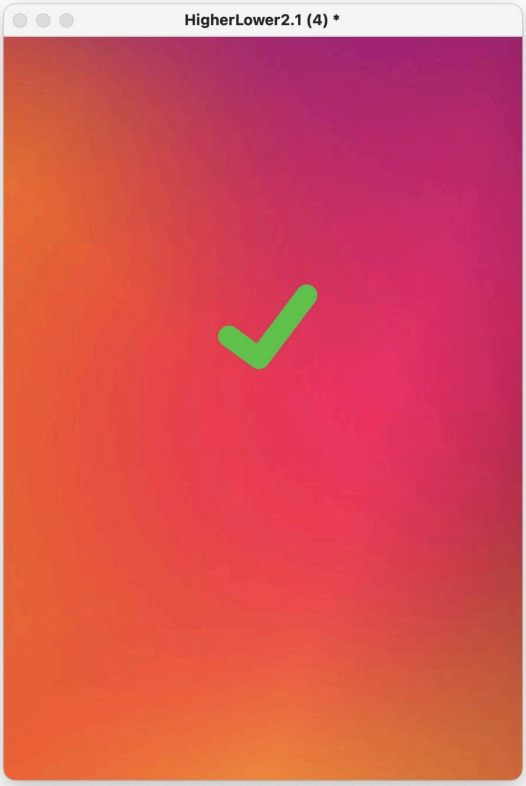
Output for successful account creation



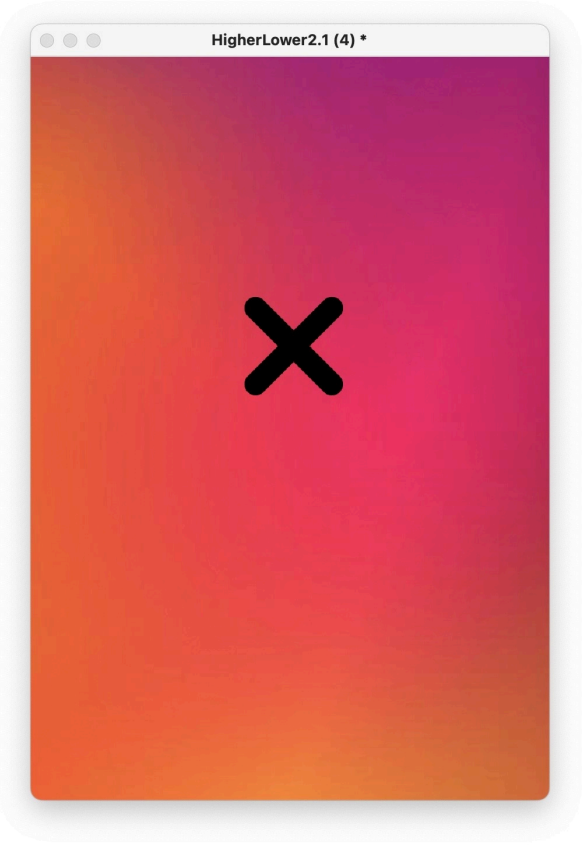
Error message if file fails to load



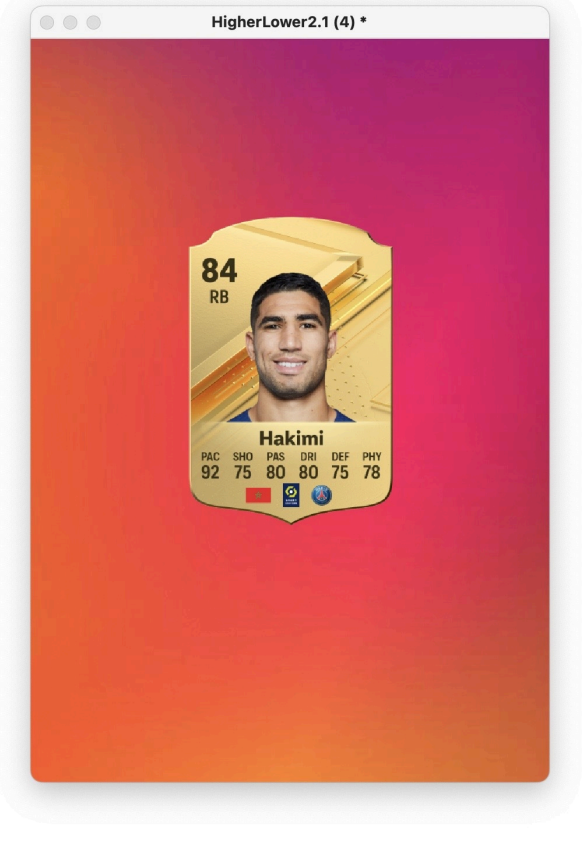
Correct guess in gameplay



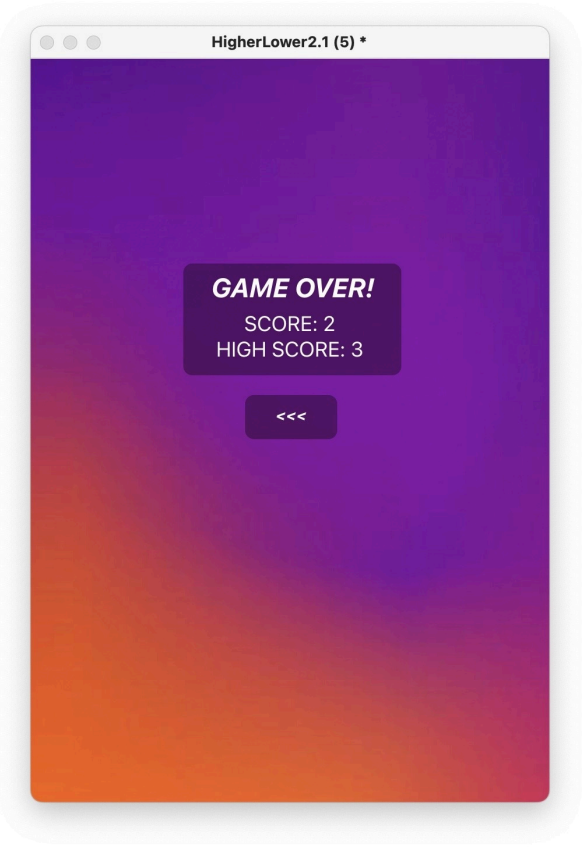
Incorrect guess in gameplay



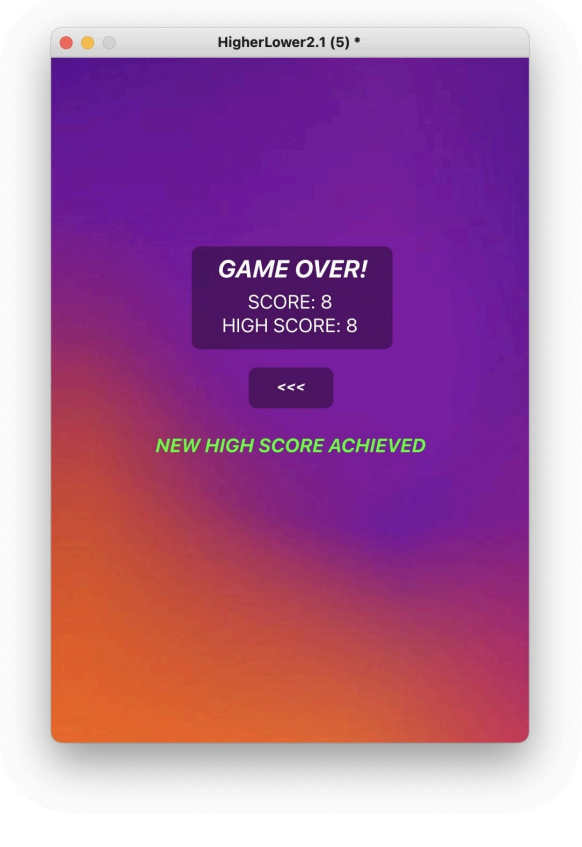
Player briefly shown when game ends



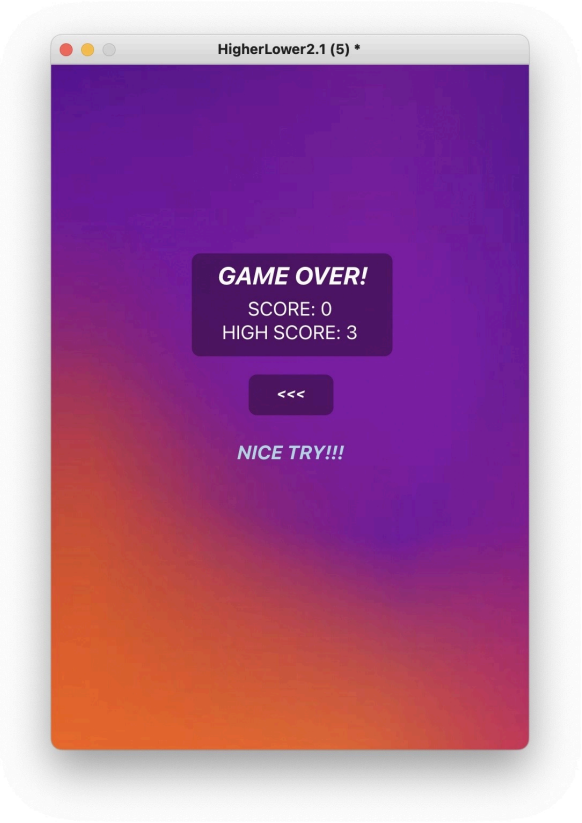
Game Over screen with no message



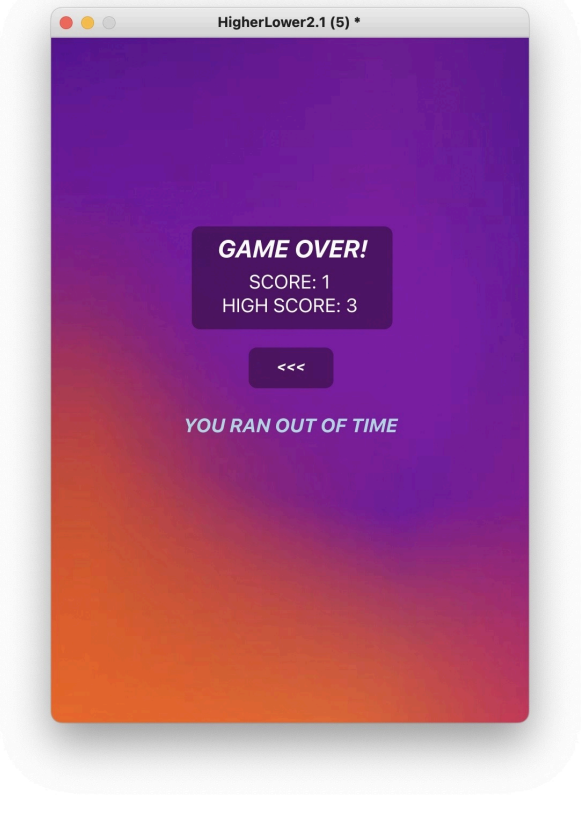
Game Over screen when a new high score is achieved



Game Over screen when the score is 0



Game Over screen when the timer runs out



New Skills Learned

Countdown
I had to research how to make the countdown timer, as this was something not covered in the Advanced Higher course. This uses a livecode service/library called 'Pending Messages' and which is totally unfamiliar to me and I have never had to use it apart from this.
Source Used: https://forums.livecode.com/viewtopic.php?t=20758

Animations
My program makes use of Livecode Visual Effects to fade out UI elements, and move between cards seamlessly. This was relatively simple to do but required me to research Livecode Wiki to discover the types of animations and different speeds I could use. This made little difference to my game aside from an improved user experience.
Source Used: https://livecode.fandom.com/wiki/Visual_effect

Ongoing Testing

Component being tested	Issues encountered	How the issue was resolved	References used
Logging in with account (design 11)	A password of any account could be used to log in to one account. The query used simply checked if the password existed in the table and didn't check it matched the username	The query was adjusted to resolve the issue by including a check for the username	/
Binary search (design 17)	My original designs used variables 'mid' and 'middle' but these words were reserved by the language	I had to change to a new variable name 'middleNum' and update my designs to reflect this	Livecode documentation: reserved keywords
Choose condition (design 21)	The line to ensure a card with the same rating was not chosen proved to not be working as on many occasions two cards	In the 'Submit Guess' procedure (see design 28), the procedure 'Choose Condition' is moved to come before 'Choose Next	/

	had the same rating value. The reason for this is that the 'Choose Next Card' procedure was being called before 'Choose Condition', so it checked that the previously selected condition was not the same value.	Card'.	

TESTING

Final test plan

Persona Cases

Case 1

Name	Ethan Hamilton
Age	17
Gender	Male
Occupation	Student
Description	Ethan is a bright 17 year old who enjoys playing games such as Ethan's Blackjack and Shane's Matching Pairs. He is confident using a computer and has good knowledge of EA FC cards as he plays the game at home.

Case 2

Name	Jack Douglas Ford
Age	59
Gender	Male
Occupation	Retired
Description	A retired blacksmith, Jack enjoys playing board games in his free time and has trouble with shaking hands so often moves pieces of board games accidentally. He is hopeful playing on a computer will be impacted by this less.

Test Cases:

The user should try to create an account
The user should try to log in with their account
The user should try and view the tutorial
The user should try to start and play the game

The user should try to view the leaderboard
The user should try to search for a player
The user should try to log out

Requirements Testing Plan

Requirement	How it will be tested
FUNCTIONAL REQUIREMENTS	
1. Login System	
1.1 Login with account	
1.1.1: The file of player cards will be read into an array of records. If this fails an error message will be shown.	A breakpoint will be used in the code to check the array of records has been filled properly. Incorrect data will purposely be entered to test the error message works.
1.1.2: A connection to the database will be opened.	A breakpoint will be used to check that the variable 'dbID' has been filled, this shows us the connection has been established.
1.1.3: If it does not already exist, a table in the database will be created.	This will first be tested where the table does not exist, to make sure it creates it correctly. Then it will be tested again to make sure it does not attempt to create it again.
1.1.4: Users will be prompted to enter their username. The username will be checked if it exists in the database. If no username was found the program will display an error.	An existing username will be entered and an SQL query will be executed to search for that username in the database, an empty response from the query shows no such username exists, if the query returns the username then it exists. An existing and non-existing username will be entered. An error message should appear for the non-existing, and for the existing it should move to functional requirement 1.1.5
1.1.5: If a username is found, a prompt to enter a password will appear. If the password entered by the user matches the related password in the database, the user will be sent to the home screen. If not, an error message will appear.	The password input box should show. A correct password will be entered and the program should switch to the home screen and have the username at the top of the screen. An incorrect password will also be entered and the program should display an error message.
1.1.6: The connection to the database will be	A temporary line of code will be inputted after

closed	the database is closed which tries to query data, executing this code should give an error if the database has been properly closed. After testing this line of code will be omitted.
1.2 Play as guest	
1.2.1: The file of player cards will be read into an array of records. If this fails an error message will be shown.	A breakpoint will be used in the code to check the array of records has been filled properly. Incorrect data will purposely be entered to test the error message works.
1.2.2: A variable “loggedinUsername” will be set to empty, this allows certain procedures to know they do not need to update the database	A breakpoint will be used to ensure the value is set properly by this line of code
1.2.3: A variable “loggedinHighScore” will be set to 0	A breakpoint will be used to ensure the value is set properly by this line of code
1.2.4: The user will be sent to the home screen	The program should switch screens to the home screen
1.3 Create account	
1.3.1: A connection to the database will be opened	A breakpoint will be used to check that the variable ‘dbID’ has been filled, this shows us the connection has been established.
1.3.2: If it does not already exist, a table in the database will be created.	This will first be tested where the table does not exist, to make sure it creates it correctly. Then it will be tested again to make sure it does not attempt to create it again.
1.3.3: Users will be prompted to enter a username of length between 3 and 15 characters. The entered username will be validated to ensure it meets the length check and additionally that it is unique and does not already exist in the database. Error messages will be shown if the username is invalid/taken.	Normal (9), extreme (15), and exceptional (2) test data will be used to ensure the input validation works as expected. A new username will be used to test if the unique function works, and then an already taken username will be used. We should expect to see error messages if the desired username does not meet the validation criteria or if it already exists in the database.
1.3.4: If the entered username is valid and unique, the user will be asked to enter a password. The password must be between 8-20 characters long and will be validated to ensure that is the case.	Normal (15), extreme (8), and exceptional (25) test data will be used to ensure the input validation works as expected. We should expect to see an error message if the desired password does not meet the validation criteria.
1.3.5: A new record will be inserted into the	The database will be checked to see if it

database with the username and password chosen by the user, the high score of the new account will be set to 0.	contains this newly added record, and that the data in the record matches the username and password entered by the user and the high score is set to 0.
1.3.6: Once the account is created the user will then be prompted to login to start playing.	Ensure the message shown to the user contains the correct username and that the message appears on screen.
1.3.7: The connection to the database will be closed	A temporary line of code will be inputted after the database is closed which tries to query data, executing this code should give an error if the database has been properly closed. After testing this line of code will be omitted.
1.4 Log out	
1.4.1: User will be sent back to the login screen	The program should switch screens to the login screen
2. Home Screen	
2.1: Users are presented with buttons to Play Game, View Leaderboard, Search User, View Instructions, and Logout.	The buttons should be shown on screen
3. Gameplay	
3.1 Start Game	
3.1.1: The UI elements are reset to their starting position	A breakpoint will be used at the start of the code and stepped through to ensure the elements are moved to the correct position
3.1.2: The current score is set to 0, and countdown is not set as the first guess has no time limit. This is shown using an infinity symbol.	A breakpoint will be used to ensure the current score variable is set to 0. The timer at the top should show an infinity symbol.
3.1.3: A starting card is randomly chosen from the array of records cardsArray[] and shown on screen	A breakpoint will be used and the module will be stepped through to ensure a random number is chosen. The selected card should appear on screen.
3.1.4: A rating is randomly chosen to be the comparison condition and is shown on screen	A breakpoint will be used and the module will be stepped through to ensure a random number is chosen, and that it goes through the if statement and sets the field name and playerCondition variable. The selected rating should appear on screen.

3.1.5: A next card is randomly chosen from the array of records cardsArray[]	A breakpoint will be used and the module will be stepped through to ensure a random number is chosen.
3.1.6: 3 hints are then displayed for the next card	The module should display the card's position, nation image, and club image onto the screen on the left.
3.1.7: The user is then sent to the gameplay screen	The game should switch to the gameplay screen
3.2 User submits guess through button 'Higher' or button 'Lower'	
3.2.1: Check if a countdown is currently running, if so, stop the countdown	A breakpoint will be used and the code will be stepped through to ensure the program checks if a countdown exists and stops it if it does
3.2.2: Set the countdown to 16 seconds (not 15: read Design>Project design>Gameplay screen for reason why)	A breakpoint will be used to check that the variable sCounter is at 16
3.2.3: Start countdown timer	The timer should begin counting down from 16
3.2.4 If the guess is correct, a tick icon is shown to the user, the next card is placed into the current card, a new condition is chosen, a new next card is chosen, new hints are shown, and the current card is shown to the user. The users score is incremented by 1, and this is updated on the screen	A correct answer will be chosen - this should then hide all the UI elements and fade out the image of the card, and show a tick. A breakpoint will be used and the code will be stepped through to ensure the variable player1 is set to player2, a new condition gets chosen by correctly updating the playerCondition variable, a random number is generated to be the next card, and that the 3 hints for the next card are shown on screen properly. The currentScore variable will be checked to ensure it has been increased by 1 and that this is reflected on the screen.
3.2.5: If the guess is incorrect or the timer reaches 0, a cross icon is displayed, followed by the image of the next card. The program checks if a new high score was achieved, if a user is logged in, the database is updated to reflect this. The game over screen is then shown with the user's score and high score. If the user's score was 0 or a new high score was achieved, a message is shown on the game over screen to reflect this.	An incorrect answer will be chosen - this should then hide all the UI elements and fade out the image of the card, and show a cross. A few seconds later the image of the next card should appear. A breakpoint will be used and the code will be stepped through. If the current score is greater than the high score, the program should check if the loggedinUsername variable is empty or not. If

	it is empty, move to the game over screen. If the logged in username variable is not empty, it should open a connection to the database. This will be checked by seeing if the variable databaseID has a value. The line of SQL to update the database will be executed and the database will be checked to ensure the new high score has been written to the database and to the correct record. The game should then switch to the game over screen. On the game over screen, the currentScore and highScore should be shown on screen, as well as a message if there is a new high score, time runs out, or the score is 0. All 3 circumstances will be tested to ensure the message is correctly displayed.
4. Search Player	
4.1: A connection to the database will be opened	A breakpoint will be used to check that the variable 'dbID' has been filled, this shows us the connection has been established.
4.2: The database will be queried and the username and high score will be inserted into an array of records called userScores[]	A breakpoint will be used to check the values inside the array of records are formatted properly and that they all exist from the database
4.3: The connection to the database will be closed	A temporary line of code will be inputted after the database is closed which tries to query data, executing this code should give an error if the database has been properly closed. After testing this line of code will be omitted.
4.4: This array of records will then be bubble sorted by username alphabetically	A breakpoint will be set at the end of the algorithm and the data will be observed to see if it is in alphabetical order of username.
4.5: User will be prompted to enter a username to search	A box should appear to enter a username, and it should be assigned to the variable targetUser
4.6: If a username is entered, a binary search will be carried out to find the user's score.	If the target username is empty, cancel the operation. If the target username is not empty, continue. A breakpoint will be used to traverse the binary search making sure it works. A

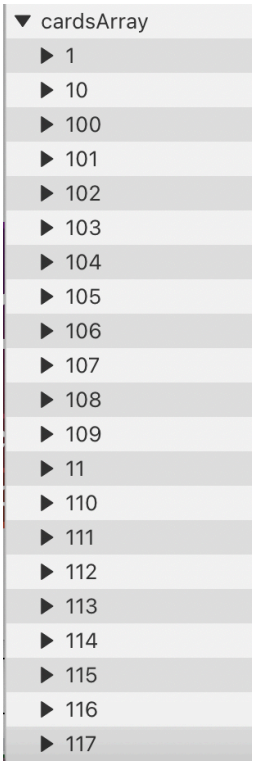
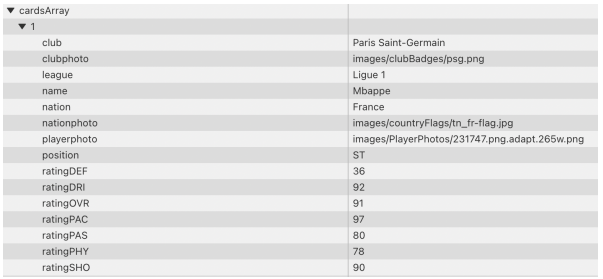
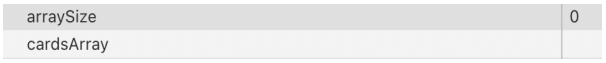
	username that exists will be used as well as a username that does not exist.
4.7: If found, the username and high score will then be shown on screen, if not found the program will tell the user that it could not find that user in the database	The username that does exist should have the high score and name shown on screen. The username that does not exist should show an error saying it was not found in the database.
5. View Leaderboard	
5.1 A connection to the database will be opened	A breakpoint will be used to check that the variable 'dbID' has been filled, this shows us the connection has been established.
5.2: The database will be queried and the username and high score will be inserted into an array of records called userScores[]	A breakpoint will be used to check the values inside the array of records are formatted properly and that they all exist from the database
5.3: The connection to the database will be closed	A temporary line of code will be inputted after the database is closed which tries to query data, executing this code should give an error if the database has been properly closed. After testing this line of code will be omitted.
5.4: This array of records will then be bubble sorted by high score highest to lowest	A breakpoint will be set at the end of the algorithm and the data will be observed to see if it is in highest to lowest of highscore.
5.5: This array of records will then be displayed in a table showing the first 10 records (the 10 highest scores and their respective usernames).	The table should display the first 10 records after sorting. The data in the table will be compared to the data in the array to make sure it is displaying the correct information.
5.6: The user will be swapped to the leaderboard card	The screen should change to the leaderboard page
5.7: A button in the bottom left corner returns the user to the home screen	This button will be clicked and it should return the user to the home screen
6. Tutorial	
6.1: Users are taken to the tutorial screen where they can see instructions on how to play	The screen should change to the tutorial page and instructions should be visible to the user
6.2: A button in the bottom left corner returns the user to the home screen	This button will be clicked and it should return the user to the home screen
7. Database	

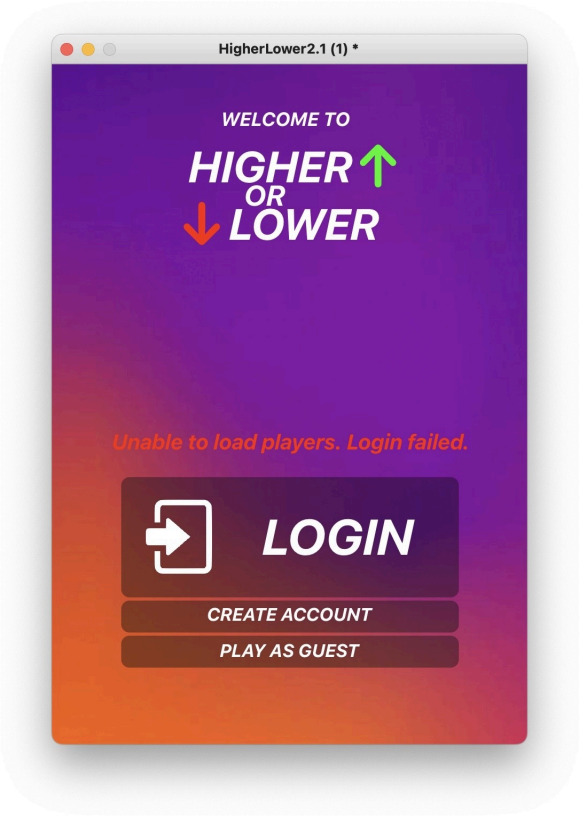
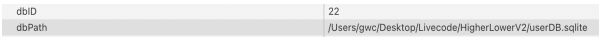
7.1: The database will be queried upon login, creating accounts, searching users, and viewing the leaderboard	Breakpoints will be used whenever data is to be queried, and the variables set to the output of the SQL will be checked to ensure they return the expected data.
7.2: New records will be inserted when accounts are created	A new account will be created and the database will be checked to see if the record was properly added in DB Browser
7.3: Records will be updated to reflect users' new high scores	A new high score will be achieved and the database will be checked to see if the record was properly updated in DB Browser
END USER REQUIREMENTS	
1. Login System	
1.1: Users can log in with an existing account using their credentials (username and password)	An existing username and password will be entered into the program and should successfully sign in
1.2: Users can create an account with a unique username and a password	A new account with a unique username and a password will be created and should be successful
1.3: Users can play as a guest in which progress will not be saved	The play as guest button should take users straight to the home screen, and any high score they achieve is not saved to the database.
2. Home Screen	
2.1: Users can start a game, logout, view the leaderboard, search for a user's score, or view instructions	The buttons listed should be visible to the user and should be clickable - Start Game - Logout - View Leaderboard - Search User - View Instructions
3. Gameplay	
3.1: Users can click the 'Higher' or 'Lower' button to submit their guess	The higher and lower button should be clickable by the user and will submit their guess to the program
3.2: Users can view their current score and high score	The current score the user has as well as their high score (if logged in/session high score) should be visible at the top of the screen while

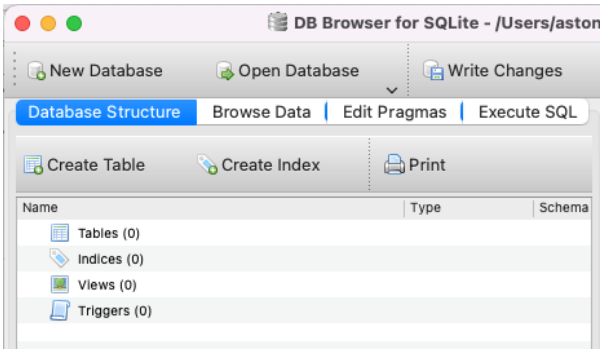
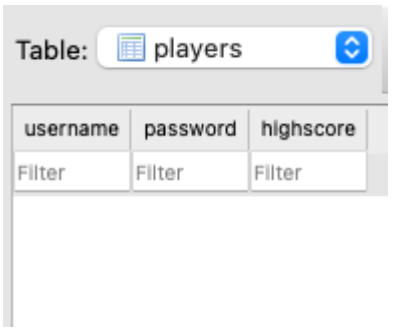
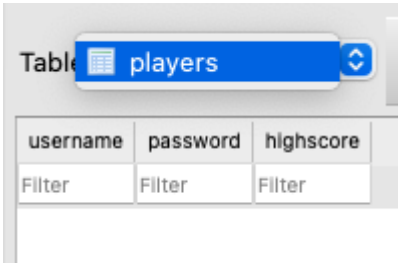
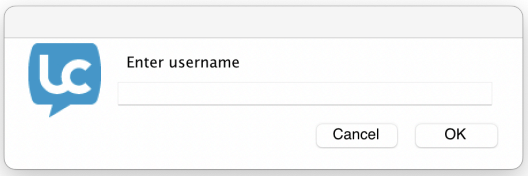
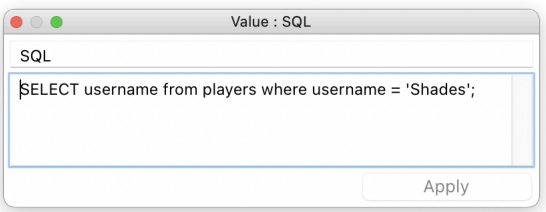
	playing
3.3: Users can view the current selected player card and the 3 hints for the next card	The current player should be shown in the middle of the screen and hints for the next card visible on the left
3.4: Users should be able to see how long they have left to make their guess	A countdown timer should be visible at the top of the screen next to the score and high score, and it should be counting down every second.
4. Search Player	
4.1: Users can enter a username and be shown the high score of the user whose name they entered	An input box should appear to enter a username. If an existing username is entered an output should show with that username and the respective high score, otherwise throw an error.
5. View Leaderboard	
5.1: Users can view the top 10 highest scores along with the username of the user that achieved the score	The usernames and scores of the top 10 highest scores should be shown on screen.
5.2: Users can return to the home screen	A 'back' button should be pressed and will take the user back to the home screen
6. Tutorial	
6.1: Users can view the instructions on how to play the game	A paragraph containing instructions of how to play the game should be shown on screen to the user
6.2: Users can return to the home screen	A 'back' button should be pressed and will take the user back to the home screen

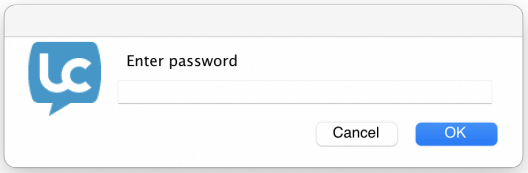
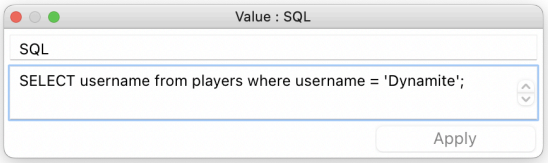
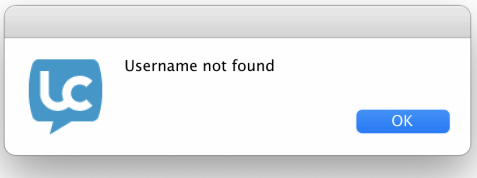
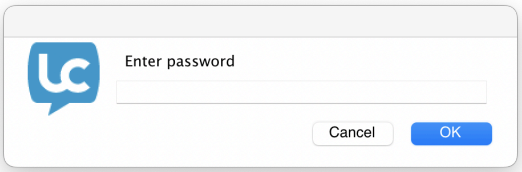
Requirements Testing

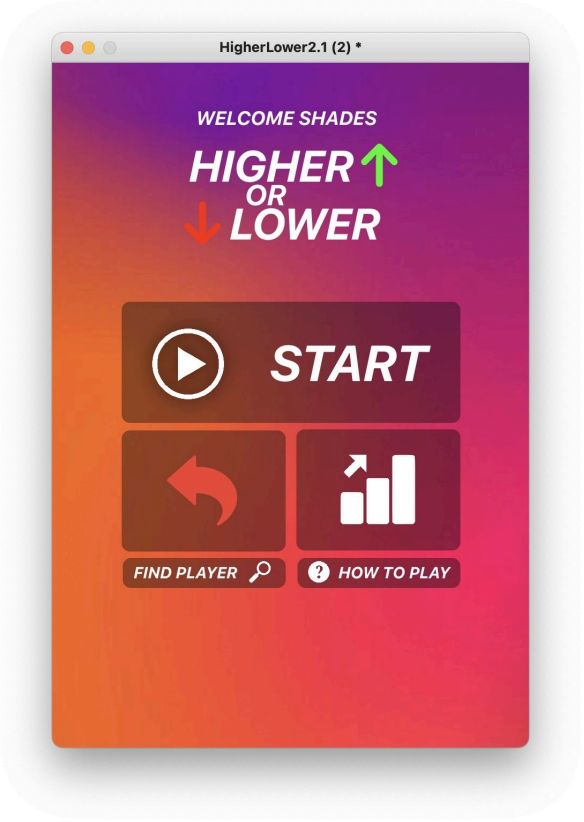
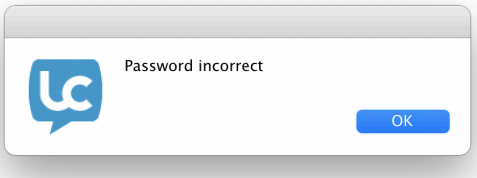
Requirement	**Evidence**	**Comments**
FUNCTIONAL REQUIREMENTS		
Functional Requirement 1.1.1 and 1.2.1: The file of player cards will be read into an	Screenshot 1:	Screenshot 1: All 122 records are loaded

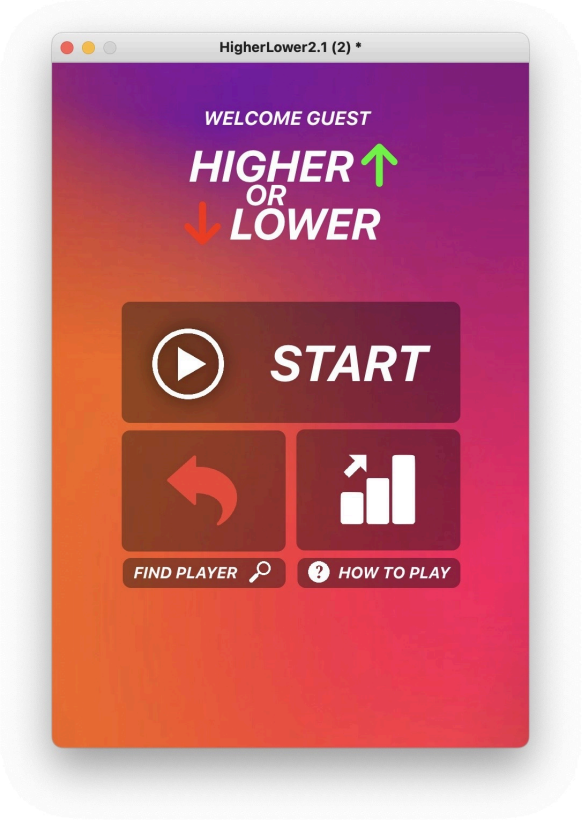
array of records. If this fails an error message will be shown. *Test Method:* A breakpoint will be used in the code to check the array of records has been filled properly. Incorrect data will purposely be entered to test the error message works.	 Screenshot 2:  Screenshot 3:  Screenshot 4:  Screenshot 5:	Screenshot 2: First record successfully read into array of records Screenshot 3: last record successfully read into array of records Screenshot 4: No file is given to the program so the array is not filled Screenshot 5: Error message shown when file is not read properly

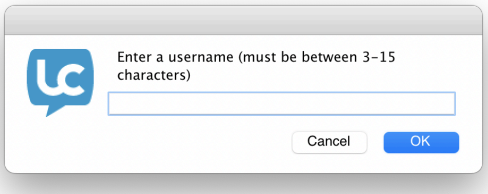
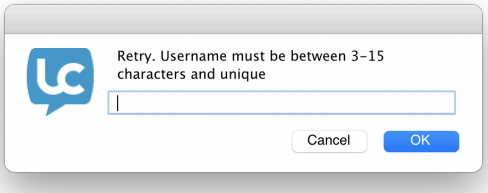
		
Functional Requirements: - 1.1.2 - 1.3.1 - 4.1 - 5.1 A connection to the database will be opened. *Test Method:* A breakpoint will be used to check that the variable 'dbID' has been filled, this shows us the connection has been established.		The dbID variable has been filled and the path to the database was filled successfully.
Functional Requirement 1.1.3 and 1.3.2: If it does not already exist, a table in the database will be created. *Test Method:* This will first be tested where the table does not exist, to make sure it creates it correctly. Then it will be	Screenshot 1:	Screenshot 1: Shows an empty database with no table Screenshot 2: Shows the table being created with the correct validation

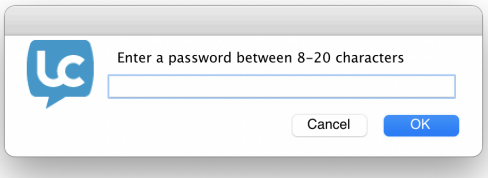
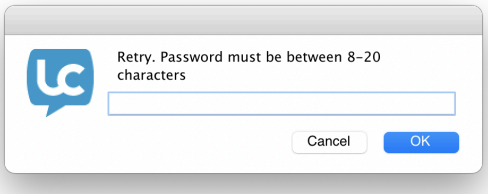
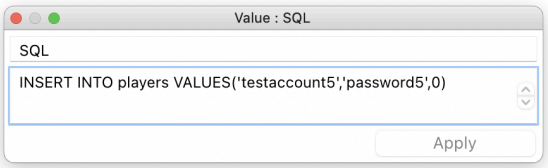
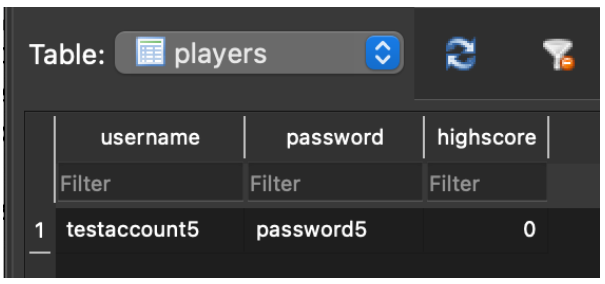
tested again to make sure it does not attempt to create it again.	 Screenshot 2:  Screenshot 3:  Screenshot 4: 	Screenshot 3: Shows the empty table with no starting data Screenshot 4: After the query is ran when the table already exists, it shows there is no duplicate table created
Functional Requirement 1.1.4 and 7.1: Users will be prompted to enter their username. The username will be checked if it exists in the database. If no username was found the program will display an error. *Test Method:* An existing username will be entered and an SQL query will be executed to search for that username in the database, an empty response from the query shows no such	Screenshot 1:  Screenshot 2: 	Screenshot 1: Shows the input box to enter the username to log in with Screenshot 2: Shows the SQL statement to be executed with our desired username that does exist Screenshot 3: Shows the output variable from SQL has been set to our

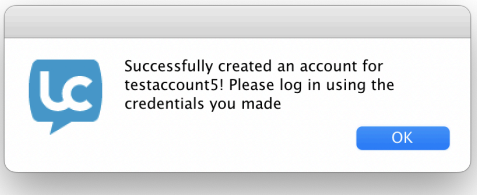
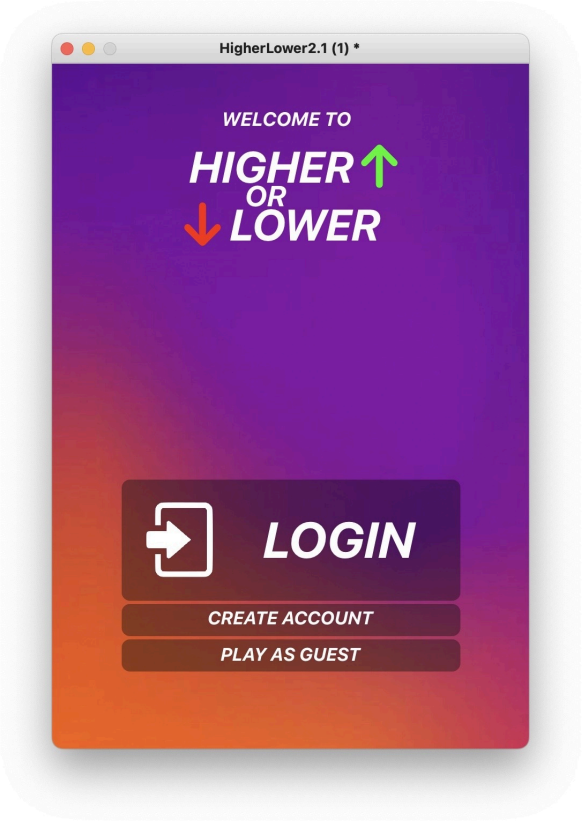
username exists, if the query returns the username then it exists. An existing and non-existing username will be entered. An error message should appear for the non-existing, and for the existing it should move to functional requirement 1.1.5	Screenshot 3:  Screenshot 4:  Screenshot 5:  Screenshot 6:  Screenshot 7: 	username. This indicates the username exists in the database Screenshot 4: Shows the program has moved onto the next functional requirement Screenshot 5: Shows our SQL statement to execute where we inputted a name that is not in the database Screenshot 6: Our returned value from the SQL statement is empty as the username does not exist in DB Screenshot 7: Shows the error message for a non-existing username
Functional Requirement 1.1.5 and 7.1: If a username is found, a prompt to enter a password will appear. If the password entered by the user matches the related password in the database, the user will be sent to the home screen. If not, an error message will appear. *Test Method:* The password input box should show. A correct password will be entered and the program should switch to the home screen and have the	Screenshot 1:  Screenshot 2:	Screenshot 1: Input box for password Screenshot 2: Shows the program has switched to the home screen and has the username we entered at the top of the screen Screenshot 3: Shows the error message when an incorrect password is inputted

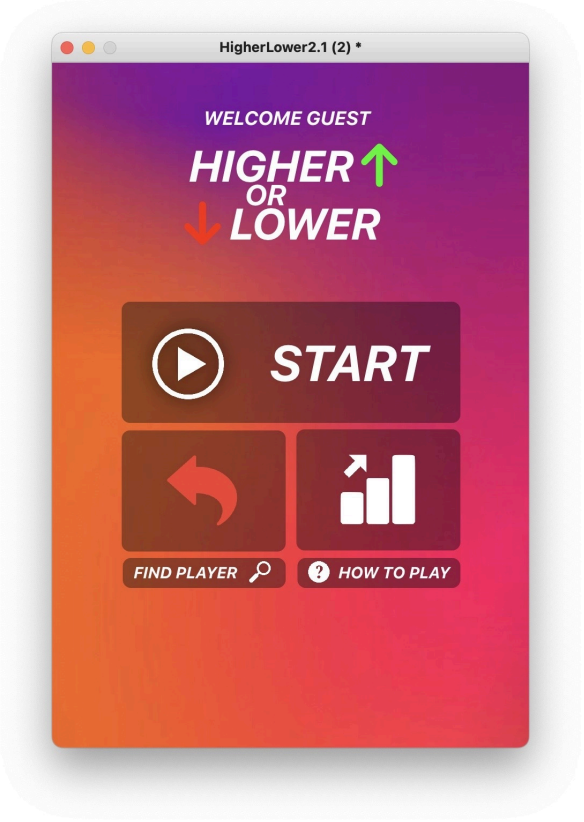
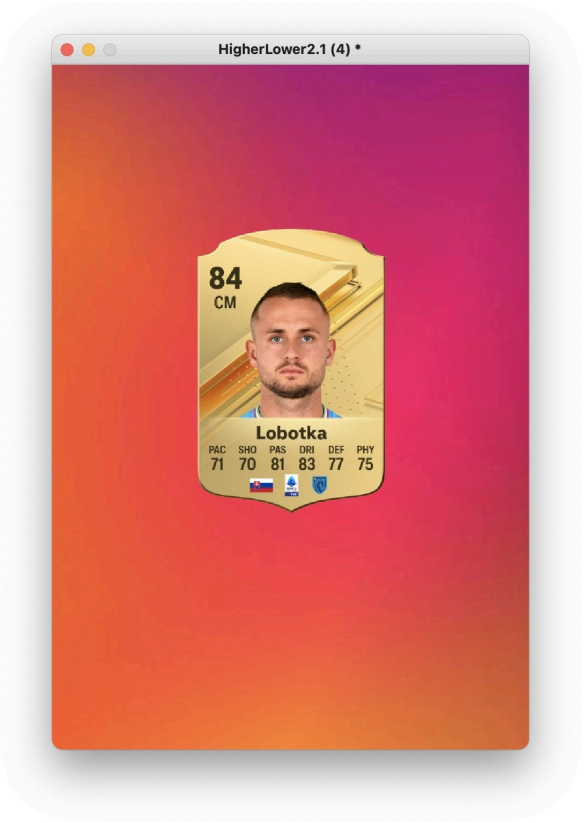
username at the top of the screen. An incorrect password will also be entered and the program should display an error message.	 Screenshot 3: 	
Functional Requirements: - 1.1.6 - 1.3.7 - 4.3 - 5.3 The connection to the database will be closed *Test Method:* A temporary line of code will be inputted after the database is closed which tries to query data, executing this code should give an error	Screenshot 1: 12 on closeDB 13 revCloseDatabase dbID 14 15 local SQL, output 16 put "SELECT * from players;" into SQL 17 put revDataFromQuery(,,dbID,SQL) into output 18 end closeDB Screenshot 2: card "loginScreen": execution error at line n/a (External handler execution error: revdberr,invalid connection id) near "re	Screenshot 1: Shows the additional lines (15-17) added for test purposes Screenshot 2: Error produced by the program as there is an invalid connection ID. This shows us the connection was successfully closed.
Functional Requirement 1.2.2: A variable	Screenshot 1:	Screenshot 1: Shows the value of

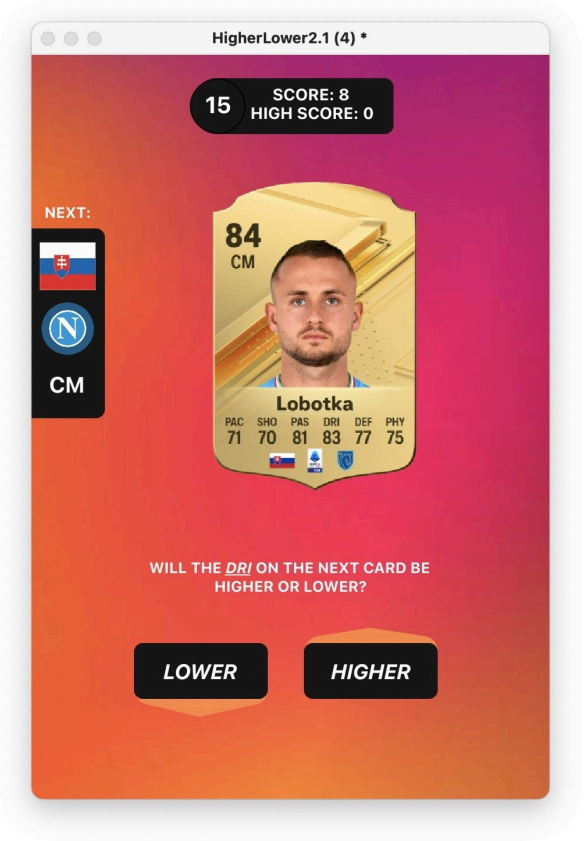
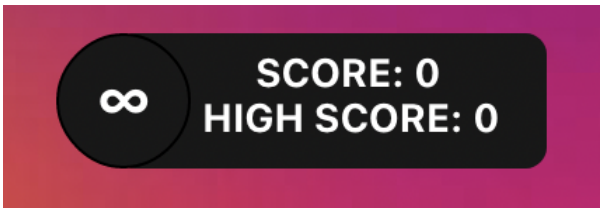
"loggedinUsername" will be set to empty, this allows certain procedures to know they do not need to update the database *Test Method:* A breakpoint will be used to ensure the value is set properly by this line of code	loggedinUsername	the variable is empty
Functional Requirement 1.2.3: A variable "loggedinHighScore" will be set to 0 *Test Method:* A breakpoint will be used to ensure the value is set properly by this line of code	Screenshot 1: loggedinHighScore 0	Screenshot 1: Shows the variable's value has been set to 0
Functional Requirement 1.2.4: The user will be sent to the home screen *Test Method:* The program should switch screens to the home screen	Screenshot 1: 	Screenshot 1: Shows that the user has been switched to the home screen and 'WELCOME GUEST' at the top is set.
Functional Requirement 1.3.3: Users will be prompted to enter a username of length between 3 and 15 characters.	Screenshot 1:	Screenshot 1: Shows input box for creating a username

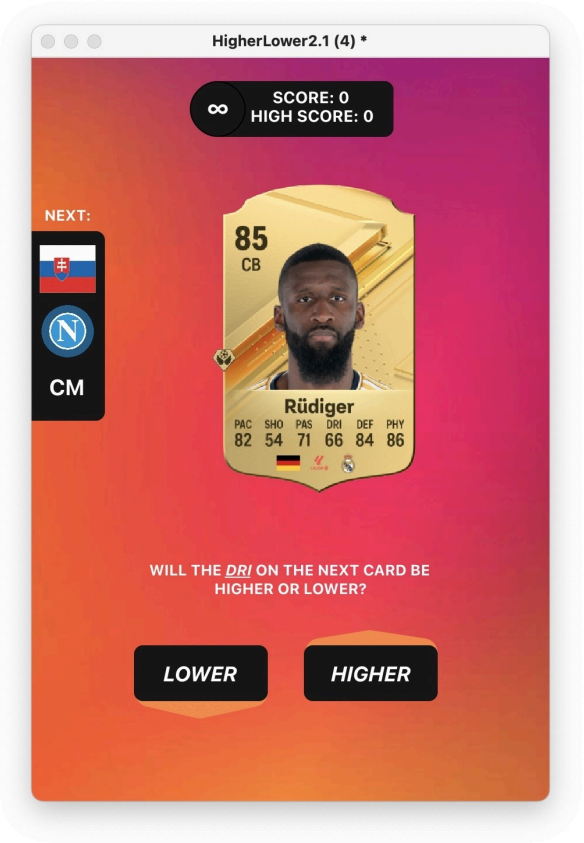
The entered username will be validated to ensure it meets the length check and additionally that it is unique and does not already exist in the database. Error messages will be shown if the username is invalid/taken. *Test Method:* Normal (9), extreme (15), and exceptional (2) test data will be used to ensure the input validation works as expected. A new username will be used to test if the unique function works, and then an already taken username will be used. We should expect to see error messages if the desired username does not meet the validation criteria or if it already exists in the database.	 Screenshot 2: valid true Screenshot 3: valid true Screenshot 4: valid false Screenshot 5:  Screenshot 6: unique true Screenshot 7: unique false	Screenshot 2: A 9 character name “ichijo-04” was entered and valid (length check) returned true. Screenshot 3: A 15 character name “fifteenchars999” was entered and valid (length check) returned true. Screenshot 4: A 2 character name “YO” was entered and valid (length check) returned false. Screenshot 5: Shows error message for an invalid or already taken username. Screenshot 6: ‘pos1’ was entered as the desired username, as it does not exist in the database the unique function returned true. Screenshot 7: ‘pos’ was entered as the desired username, it already exists in the database so the unique function returned false.
Functional Requirement 1.3.4: If the entered username is valid and unique, the user will be asked to enter a password.	Screenshot 1:	Screenshot 1: Shows input box for password

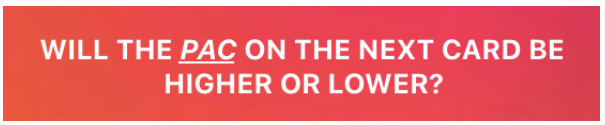
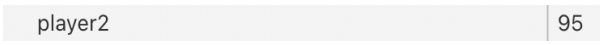
The password must be between 8-20 characters long and will be validated to ensure that is the case. *Test Method:* Normal (15), extreme (8), and exceptional (25) test data will be used to ensure the input validation works as expected. We should expect to see an error message if the desired password does not meet the validation criteria.	 Screenshot 2: valid | true Screenshot 3: valid | true Screenshot 4: valid | false Screenshot 5: 	Screenshot 2: An 8 character name "8charsss" was entered and valid (length check) returned true. Screenshot 3: A 15 character name "fifteencharact_" was entered and valid (length check) returned true. Screenshot 4: A 25 character username "twentyfivefivecharacters" was tnered and valid (length check) returned false Screenshot 5: Shows error message for a password of incorrect length
Functional Requirement 1.3.5 and 7.2: A new record will be inserted into the database with the username and password chosen by the user, the high score of the new account will be set to 0. *Test Method:* The database will be checked to see if it contains this newly added record, and that the data in the record matches the username and password entered by the user and the high score is set to 0.	Screenshot 1: desiredPassword | password5 desiredUsername | testaccount5 Screenshot 2:  Screenshot 3: 	Screenshot 1: Shows the test data Screenshot 2: Shows the SQL query to be executed with the test data Screenshot 3: Shows the record has been added to the database correctly

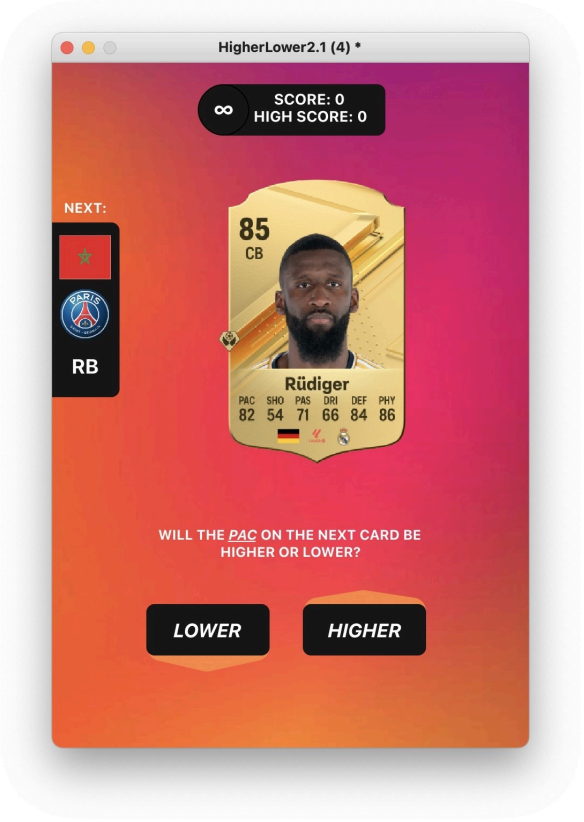
Functional Requirement 1.3.6: Once the account is created the user will then be prompted to login to start playing. *Test Method:* Ensure the message shown to the user contains the correct username and that the message appears on screen.	Screenshot 1:  	Screenshot 1: Shows the message is displayed with the correct username of the new account.
Functional Requirement 1.4.1; User will be sent back to the login screen *Test Method:* The program should switch screens to the login screen	Screenshot 1:  	Screenshot 1: Shows us the user has been returned to the login screen when the logout button was pressed.
Functional Requirement 2.1: Users are presented with buttons to Play Game, View Leaderboard, Search User, View Instructions, and Logout. *Test Method:* The buttons should be shown on screen	Screenshot 1:	Screenshot 1: All required buttons are visible to the user on the home screen

		
Functional Requirement 3.1.1: The UI elements are reset to their starting position *Test Method:* A breakpoint will be used at the start of the code and stepped through to ensure the elements are moved to the correct position	Screenshot 1:  Screenshot 2:	Screenshot 1: Shows all UI elements in their state at the end of the previous game Screenshot 2: Shows breakpoint set in code Screenshot 3: Shows the UI elements now returned to the correct position to start the game

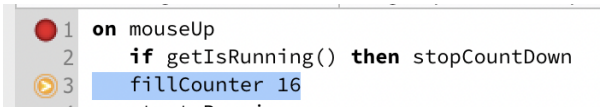
	`` on initialiseUI show fld "conditionFld" of card "gameplay" hide image "result" of card "gameplay" show image "playerImage" of card "gameplay" show group "hintGroup" of card "gameplay" show group "timeGroup" of card "gameplay" show group "higherGroup" of card "gameplay" show group "lowerGroup" of card "gameplay" set the location of image "playerImage" of card "gameplay" to 243,241 end initialiseUI ``` Screenshot 3: 	
Functional Requirement 3.1.2: The current score is set to 0, and countdown is not set as the first guess has no time limit. This is shown using an infinity symbol. *Test Method:* A breakpoint will be used to ensure the current score variable is set to 0. The timer at the top should show an infinity symbol.	Screenshot 1: currentScore | 0 Screenshot 2: 	Screenshot 1: Shows the currentScore variable has been set to 0. Screenshot 2: Shows the timer has been set to show an infinity symbol
Functional Requirement 3.1.3: A starting card is randomly chosen from the array of records cardsArray[] and	Screenshot 1: player1 | 70 Screenshot 2:	Screenshot 1: Shows the random number generated by the program

shown on screen *Test Method:* A breakpoint will be used and the module will be stepped through to ensure a random number is chosen. The selected card should appear on screen.		within the numOfCards range Screenshot 2: The newly selected card is now shown on screen
Functional Requirement 3.1.4: A rating is randomly chosen to be the comparison condition and is shown on screen *Test Method:* A breakpoint will be used and the module will be stepped through to ensure a random number is chosen, and that it goes through the if statement and sets the field name and playerCondition variable. The selected rating should appear on screen.	Screenshot 1: randNum | 2 Screenshot 2: `on chooseCondition` `  local randNum, fieldName` `  put random(7) into randNum` `  if randNum = 1 then` `  put "OVERALL RATING" into fiel` `  put "ratingOVR" into playerCon` `  else if randNum = 2 then` `  put "PAC" into fieldName` `  put "ratingPAC" into playerCon` `  else if randNum = 3 then` `  put "SHO" into fieldName` `  put "ratingSHO" into playerCon` `  else if randNum = 4 then` `  put "PAS" into fieldName` `  put "ratingPAS" into playerCon` Screenshot 3: fieldName | PAC Screenshot 4: playerCondition | ratingPAC Screenshot 5:	Screenshot 1: Shows the random number generated by the program Screenshot 2: Shows the program has stepped into the correct if statement Screenshot 3: Shows the fieldName variable has been set correctly Screenshot 4: Shows the playerCondition variable has been set correctly Screenshot 5: Shows the selected

		rating has been displayed on screen
Functional Requirement 3.1.5: A next card is randomly chosen from the array of records cardsArray[] *Test Method:* A breakpoint will be used and the module will be stepped through to ensure a random number is chosen.	Screenshot 1: 	Screenshot 1: The player 2 variable has been set to a random number within the numOfCards range
Functional Requirement 3.1.6: 3 hints are then displayed for the next card *Test Method:* The module should display the card's position, nation image, and club image onto the screen on the left.	Screenshot 1: 	Screenshot 1: Shows that the program has displayed the hints for the next card (95)
Functional Requirement 3.1.7: The user is then sent to the gameplay screen *Test Method:* The game should switch to the gameplay screen	Screenshot 1:	Screenshot 1: Shows the program has been switched to the gameplay screen and all the correct information is displayed

<table>
<tr>
<td></td>
<td> 
</td>
<td></td>
</tr>
<tr>
<td><u>Functional Requirement 3.2.1:</u>
Check if a countdown is currently running, if so, stop the countdown

<u>Test Method:</u>
A breakpoint will be used and the code will be stepped through to ensure the program checks if a countdown exists and stops it if it does</td>
<td>Screenshot 1:
 

Screenshot 2:
 

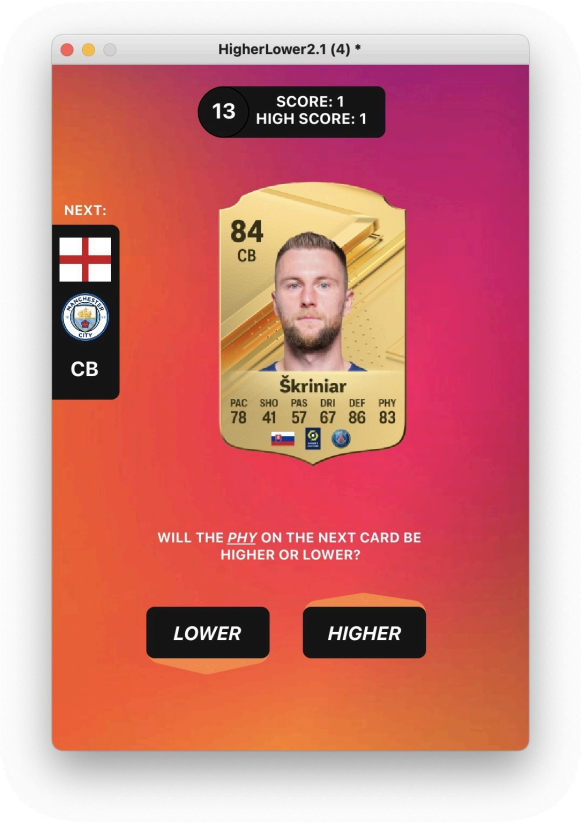
Screenshot 3:
 

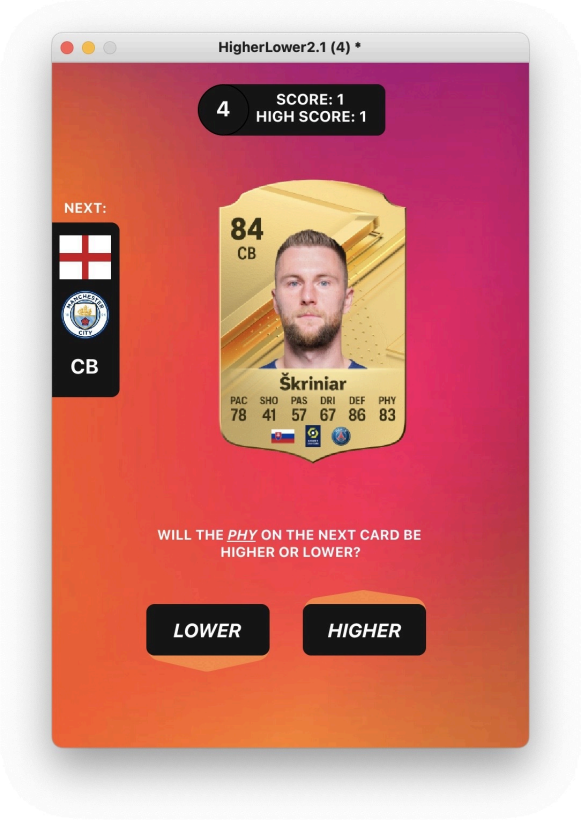
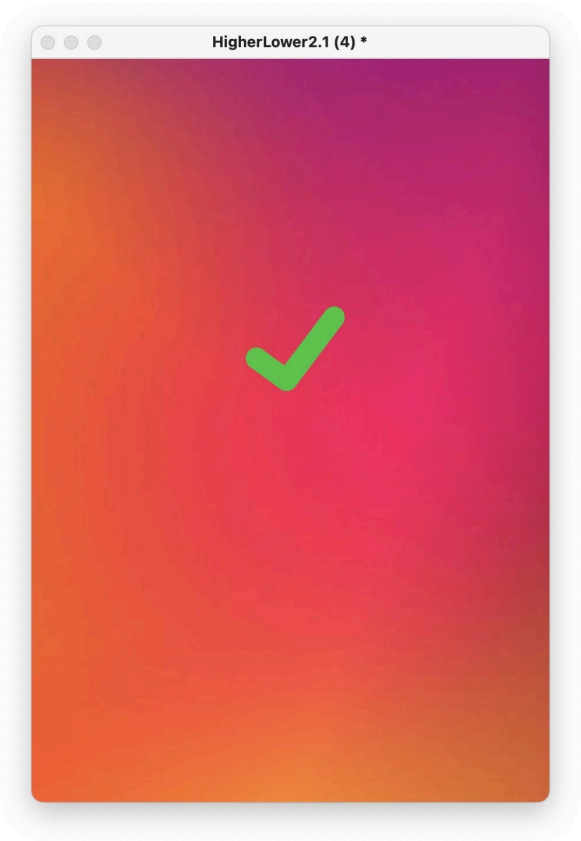
Screenshot 4:
 
</td>
<td>Screenshot 1: Countdown is not running, the program should skip over the if statement

Screenshot 2: Shows the stopCountDown procedure was not called as the program skipped the IF statement

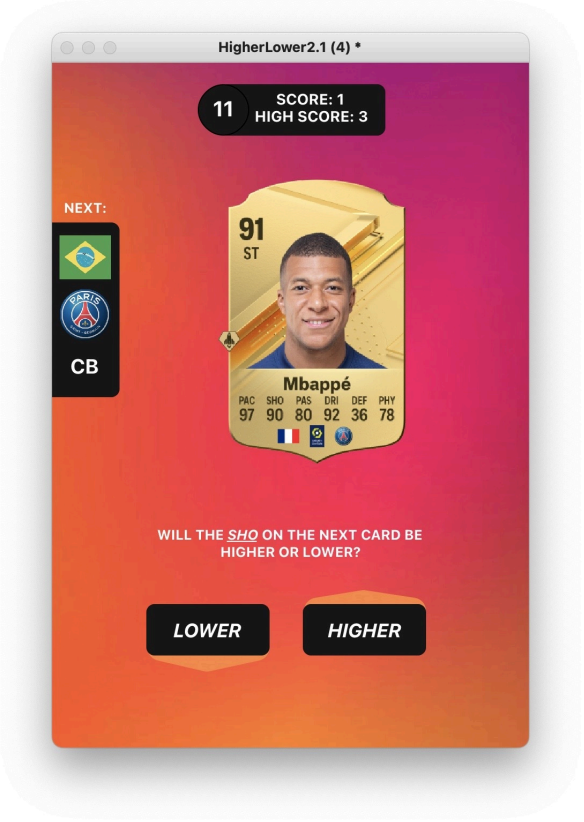
Screenshot 3: Shows the countdown is running

Screenshot 4: The countdown is stopped and set to 0 by the stopCountDown</td>
</tr>
</table>

		procedure
Functional Requirement 3.2.2: Set the countdown to 16 seconds *Test Method:* A breakpoint will be used to check that the variable sCounter is at 16	Screenshot 1: sCounter 16	Screenshot 1: The sCounter variable has been set to 16
Functional Requirement 3.2.3: Start countdown timer *Test Method:* The timer should begin counting down from 16	Screenshot 1:  Screenshot 2:	Screenshot 1: Shows the countdown timer at 13 seconds Screenshot 2: Shows the countdown timer at 4 seconds

		
Functional Requirement 3.2.4: If the guess is correct, a tick icon is shown to the user, the next card is placed into the current card, a new condition is chosen, a new next card is chosen, new hints are shown, and the current card is shown to the user. The users score is incremented by 1, and this is updated on the screen *Test Method:* A correct answer will be chosen - this should then hide all the UI elements and fade out the image of the card, and show a tick. A breakpoint will be used and the code will be stepped through to ensure the variable player1 is set to player2, a new condition gets chosen by correctly updating the	Screenshot 1:  Screenshot 2:	Screenshot 1: When we submit our guess and it is correct, all current UI elements are hidden and we are briefly shown this 'tick' icon. Screenshot 2: Shows the current player (player1) and the next player (player2) variables before player 1 is updated to be player 2's value. Screenshot 3: Shows the values are now the same (both are player2's value) before player 2 receives a new value in the next

playerCondition variable, a random number is generated to be the next card, and that the 3 hints for the next card are shown on screen properly. The currentScore variable will be checked to ensure it has been increased by 1 and that this is reflected on the screen.	player1 70 player2 1 Screenshot 3: player1 1 player2 1 Screenshot 4: randNum 3 Screenshot 5: `on chooseCondition` `  local randNum, fieldName` `  put random(7) into randNum` `  if randNum = 1 then` `  put "OVERALL RATING" into fieldName` `  put "ratingOVR" into playerCondition` `  else if randNum = 2 then` `  put "PAC" into fieldName` `  put "ratingPAC" into playerCondition` `  else if randNum = 3 then` `  put "SHO" into fieldName` `  put "ratingSHO" into playerCondition` `  else if randNum = 4 then` Screenshot 6: fieldName SHO Screenshot 7: playerCondition ratingSHO Screenshot 8: player1 1 player2 23 Screenshot 9: currentScore 0 Screenshot 10: currentScore 1 Screenshot 11:	step Screenshot 4: Shows the random number generated for the higher/lower condition Screenshot 5: The IF statement has been stepped into at the correct line Screenshot 6: The fieldName variable has been set correctly Screenshot 7: The playerCondition variable has been set correctly Screenshot 8: Shows the new random number generated for player2's value that is not the same value as player1. Screenshot 9: Shows the current score before the point is added Screenshot 10: Shows the new value for current score Screenshot 11: Shows the new card on screen, as well as the hints for the next card, new player condition for the next card, and the updated

		current score at the top.
Functional Requirement 3.2.5 and 7.3: If the guess is incorrect, a cross icon is displayed, followed by the image of the next card. The program checks if a new high score was achieved, if a user is logged in, the database is updated to reflect this. The game over screen is then shown with the user's score and high score. If the user's score was 0 or a new high score was achieved, a message is shown on the game over screen to reflect this. *Test Method:* An incorrect answer will be chosen - this should then hide all the UI elements and fade out the image of the card, and show a cross. A few seconds later the image of the next	Screenshot 1:  Screenshot 2:	Screenshot 1: An incorrect guess was submitted, all present UI elements are faded away and the cross is shown briefly on screen Screenshot 2: After a .5 second wait, the image of the next card is briefly shown on screen Screenshot 3: The current score is not greater than the high score, so the program should skip over the IF statement Screenshot 4: The program correctly

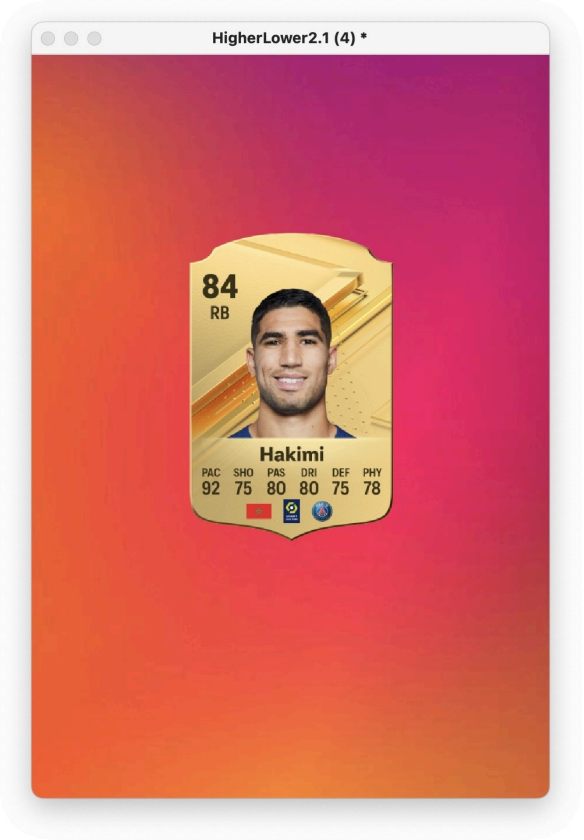
card should appear.

A breakpoint will be used and the code will be stepped through.

If the current score is greater than the high score, the program should check if the loggedinUsername variable is empty or not. If it is empty, move to the game over screen.

If the logged in username variable is not empty, it should open a connection to the database. This will be checked by seeing if the variable databaseID has a value. The line of SQL to update the database will be executed and the database will be checked to ensure the new high score has been written to the database and to the correct record. The game should then switch to the game over screen.

On the game over screen, the currentScore and highScore should be shown on screen, as well as a message if there is a new high score, time runs out, or the score is 0. All 3 circumstances will be tested to ensure the message is correctly displayed.



Screenshot 3:

currentScore	2
loggedinHighScore	3

Screenshot 4:

```
210     if currentScore > loggedinHighScore then
211        set the foregroundcolor of fld "message
212        put "NEW HIGH SCORE ACHIEVED" into fld
213        put currentScore into loggedinHighScore
214        if loggedinUsername <> "" then
215           updateDatabase
216        end if
217     end if
218
19      put "SCORE:" && currentScore & cr & "HIGH
```

Screenshot 5:

currentScore	8
loggedinHighScore	3

Screenshot 6:

```
210   if currentScore > loggedinHighScore then
211      set the foregroundcolor of fld "messageFld" of card "gameOver" to "green"
212      put "NEW HIGH SCORE ACHIEVED" into fld "messageFld" of card "gameOver"
213      put currentScore into loggedinHighScore
214      if loggedinUsername <> "" then
215         updateDatabase
216      end if
217   end if
```

Screenshot 7:

skips over the IF statement

Screenshot 5: Current score is greater than high score, the IF statement should be stepped into.

Screenshot 6: Shows the IF statement correctly being stepped into.

Screenshot 7: Session high score variable updated.

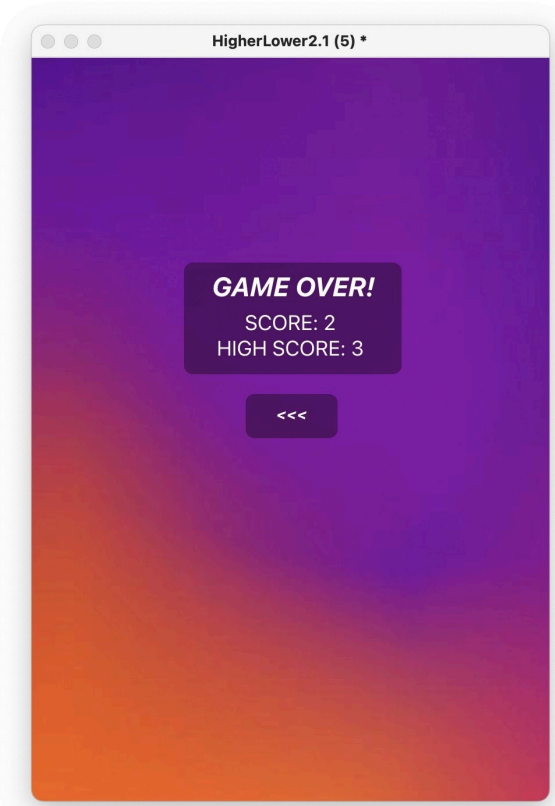
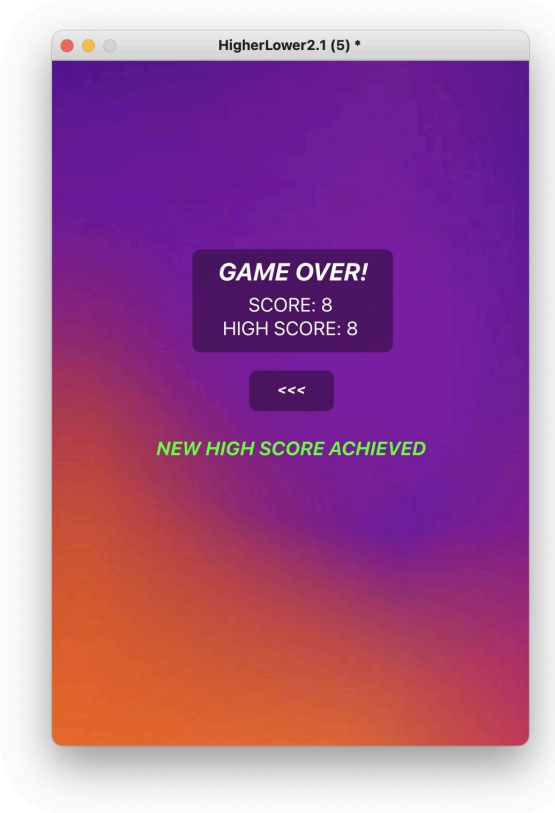
Screenshot 8: Shows the session username variable is empty. The program should not attempt to update the database because of this.

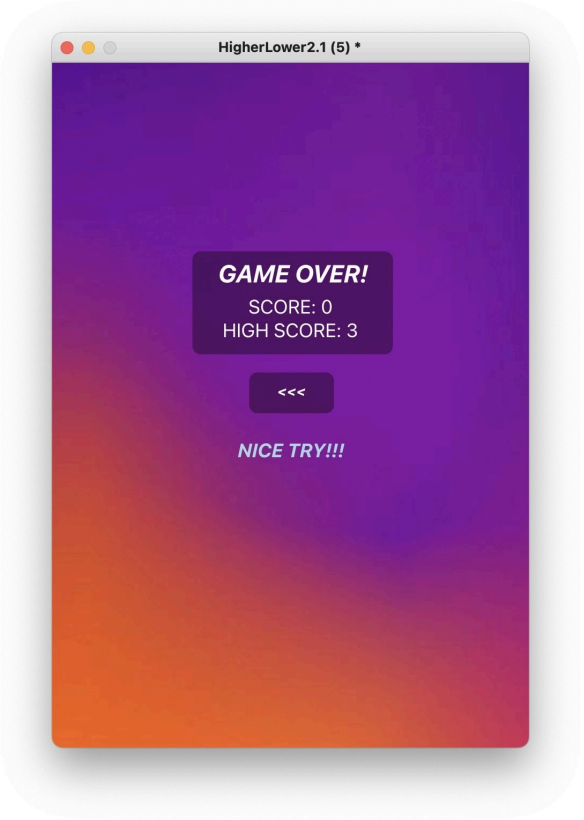
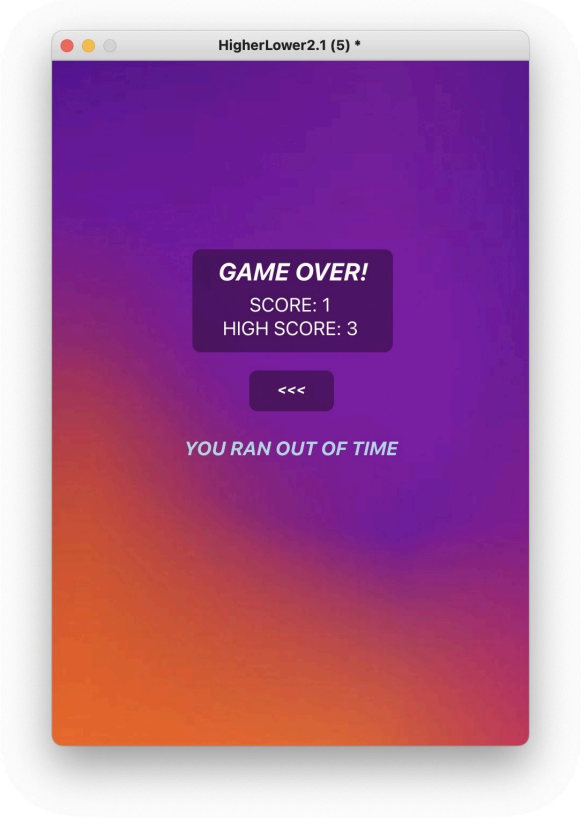
Screenshot 9: The program correctly skips over the IF statement as the session username variable is empty.

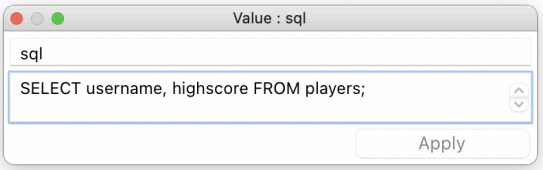
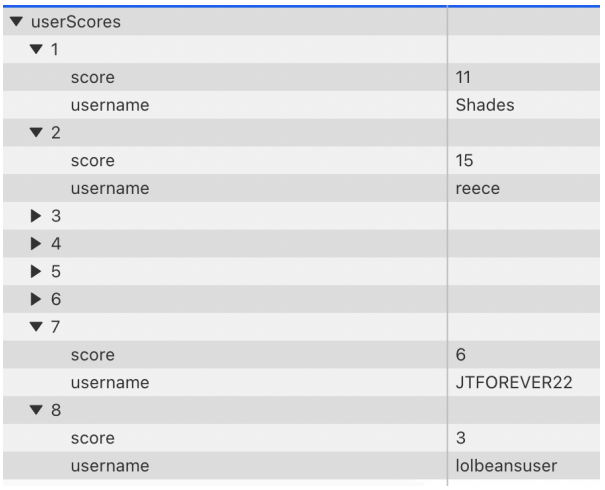
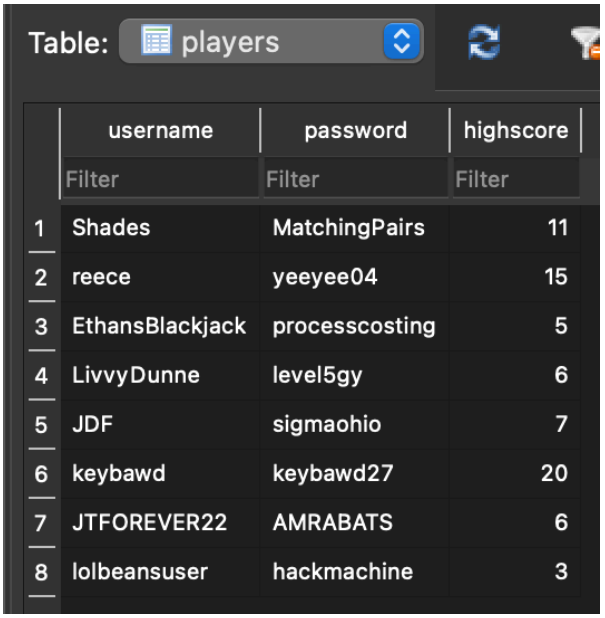
Screenshot 10: In this instance we are logged in so our session variable is filled with our username.

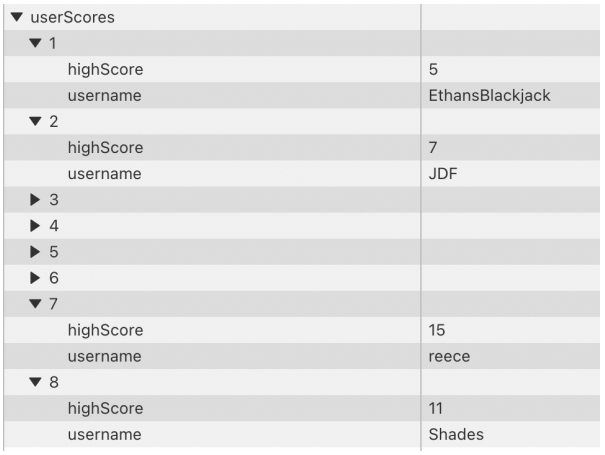
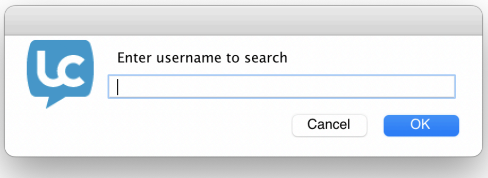
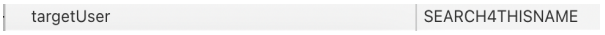
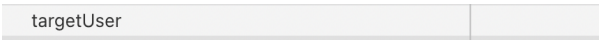
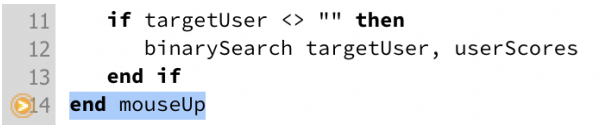
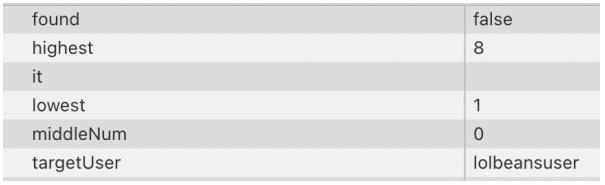
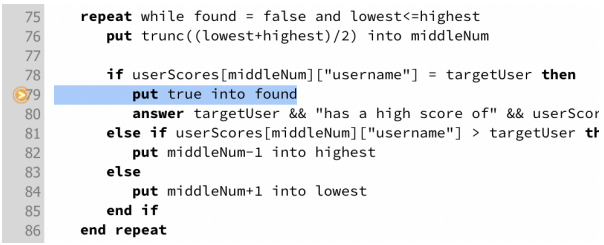
Screenshot 11: Correctly, the program steps into the IF statement and will now start the process of updating the

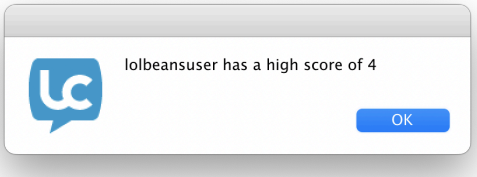
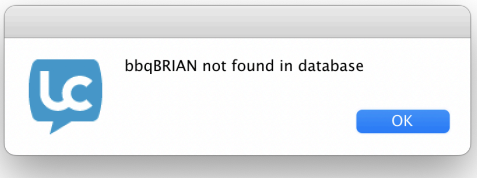
	loggedinHighScore | 8 Screenshot 8: loggedinUsername | Screenshot 9: `if loggedinUsername <> "" then` `updateDatabase` `end if` `end if` `put "SCORE:" && currentScore & cr & "HIGH` Screenshot 10: loggedinUsername | lolbeansuser Screenshot 11: `if loggedinUsername <> "" then` `updateDatabase` `end if` Screenshot 12: dbID | 30 dbPath | /Users/gwc/Desktop/Livecode/HigherLowerV2/userD... Screenshot 13: Value : sql sql UPDATE players SET highscore = 3 WHERE username ='lolbeansuser'; Apply Screenshot 14: Table: players username | password | highscore Filter | Filter | Filter 1 | lolbeansuser | hackmachine | 3 Screenshot 15: `on closeDB` `revCloseDatabase dbID` `local SQL, output` `put "SELECT * from players;" into SQL` `put revDataFromQuery(,,dbID,SQL) into output` `end closeDB` Screenshot 16: card "loginScreen": execution error at line n/a (External handler execution error: revdberr,invalid connection id) near "re Screenshot 17:	database. Screenshot 12: A database path is found and the dbID value is set Screenshot 13: The SQL statement to be executed, with the username and new high score Screenshot 14: Shows the database has been updated with the new high score Screenshot 15: Shows the additional lines (15-17) added for test purposes Screenshot 16: Error produced by the program as there is an invalid connection ID. This shows us the connection was successfully closed. Screenshot 17: The program switches to the end screen showing the score achieved, the user's high score (or session high score if playing as a guest), and a button to return to the menu. No end screen message is shown in this instance. Screenshot 18: End

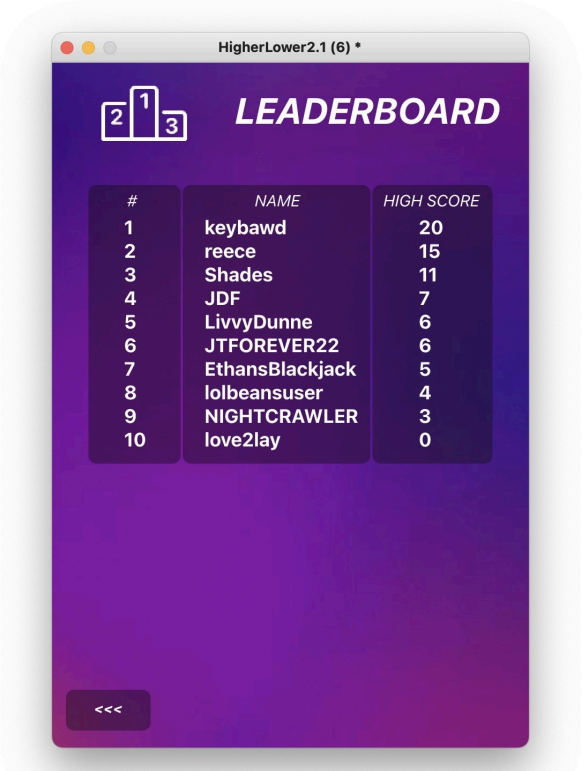
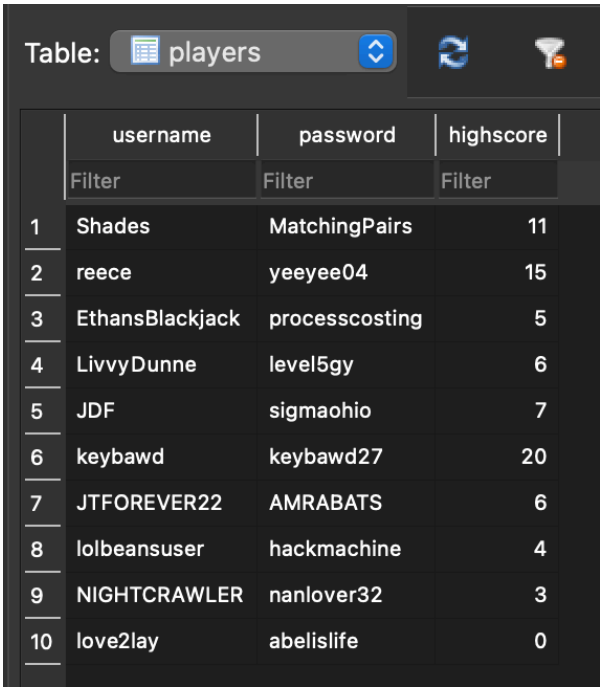
	 Screenshot 18:  Screenshot 19:	screen with message for when a user achieves a new high score Screenshot 19: End screen with message for when a user's score is 0 and they lose Screenshot 20: End screen with message for when a user runs out of time

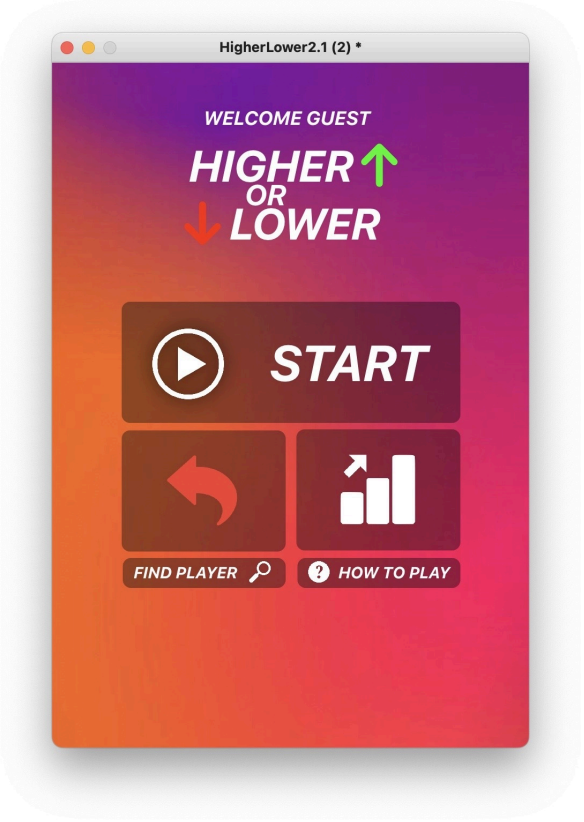
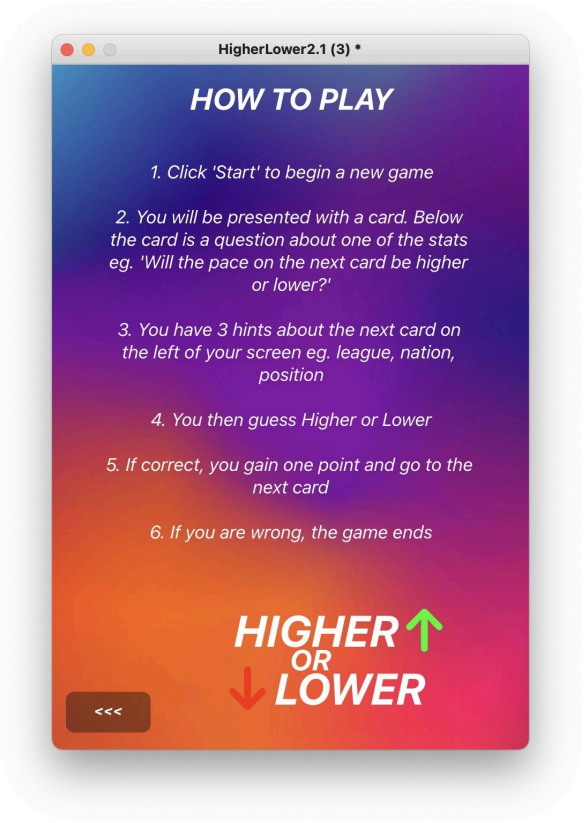
	 Screenshot 20: 	
Functional Requirement 4.2	Screenshot 1:	Screenshot 1:

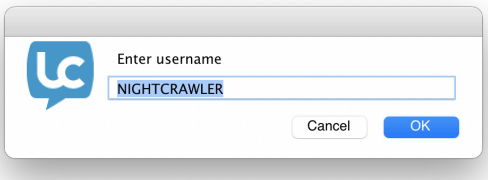
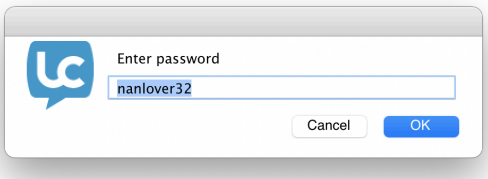
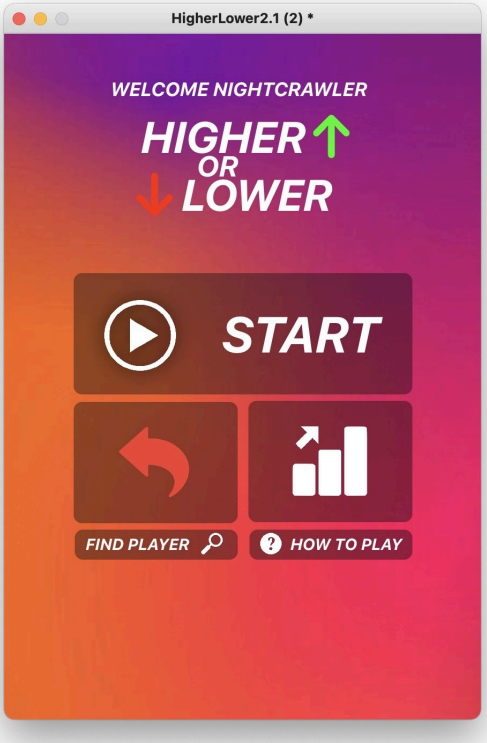
and 5.2: The database will be queried and the username and high score will be inserted into an array of records called userScores[] *Test Method:* A breakpoint will be used to check the values inside the array of records are formatted properly and that they all exist from the database	 Screenshot 2:  Screenshot 3: 	Shows the SQL statement to be executed to retrieve usernames and highscores of all players Screenshot 2: Shows the first 2 and last 2 records in the array, retrieved from the database. Screenshot 3: Data in database, matches what is in the array of records
Functional Requirement 4.4: This array of records will then be bubble sorted by username alphabetically *Test Method:* A breakpoint will be set at the end of the algorithm and the data will be observed to see if	Screenshot 1:	Screenshot 1: The order of the records in the array has been changed and is now in alphabetical order by username

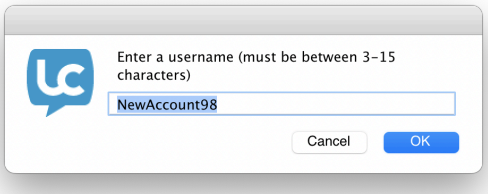
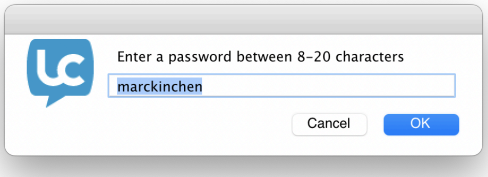
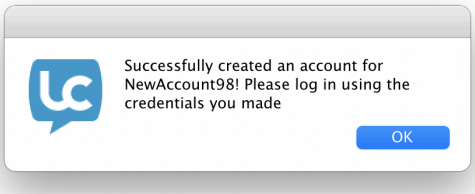
it is in alphabetical order of username.		
Functional Requirement 4.5: User will be prompted to enter a username to search *Test Method:* A box should appear to enter a username, and it should be assigned to the variable targetUser	Screenshot 1:  Screenshot 2: 	Screenshot 1: Input box for the desired username to search for. Screenshot 2: Variable 'targetUser' has been set to the input we gave it.
Functional Requirement 4.6: If a username is entered, a binary search will be carried out to find the user's score. *Test Method:* If the target username is empty, cancel the operation. If the target username is not empty, continue. A breakpoint will be used to traverse the binary search making sure it works. A username that exists will be used as well as a username that does not exist.	Screenshot 1:  Screenshot 2:  Screenshot 3:  Screenshot 4:  Screenshot 5:	Screenshot 1: targetUser is empty Screenshot 2: The program correctly does not enter the IF statement and does not proceed with the binary search as the target username is empty. Screenshot 3: Binary search variables initialised properly (ignore "it" - livecode variable) Screenshot 4: Position found on second iteration

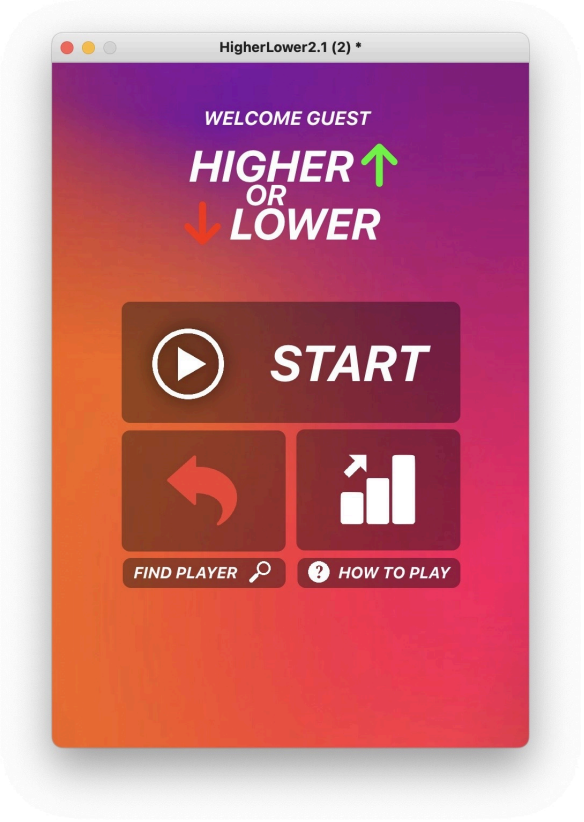
	found: false highest: 0 it: lowest: 1 middleNum: 1 targetUser: bbqBRIAN	Screenshot 5: Shows the final values in the binary search when we enter a username that does not exist (ignore "it")
Functional Requirement 4.7: If found, the username and high score will then be shown on screen, if not found the program will tell the user that it could not find that user in the database *Test Method:* The username that does exist should have the high score and name shown on screen. The username that does not exist should show an error saying it was not found in the database.	Screenshot 1:  Screenshot 2: 	Screenshot 1: Shows the expected output for our existing user, with the name and high score Screenshot 2: Expected output for a username that does not exist in the database
Functional Requirement 5.4: This array of records will then be bubble sorted by high score highest to lowest *Test Method:* A breakpoint will be set at the end of the algorithm and the data will be observed to see if it is in highest to lowest of highscore.	Screenshot 1: userScores: 1: score: 20 username: keybawd 10: score: 0 username: love2lay 2: score: 15 username: reece 3: 4: 5: 6: 7: 8: 9: score: 3 username: NIGHTCRAWLER	Screenshot 1: Shows the bubble sorted array of records by score, with the first 2 and last 2 visible.
Functional Requirement 5.5 and 5.6: This array of records will then be displayed in a table showing the first 10 records (the 10 highest scores and	Screenshot 1:	Screenshot 1: The program has switched to the leaderboard screen which contains position numbers,

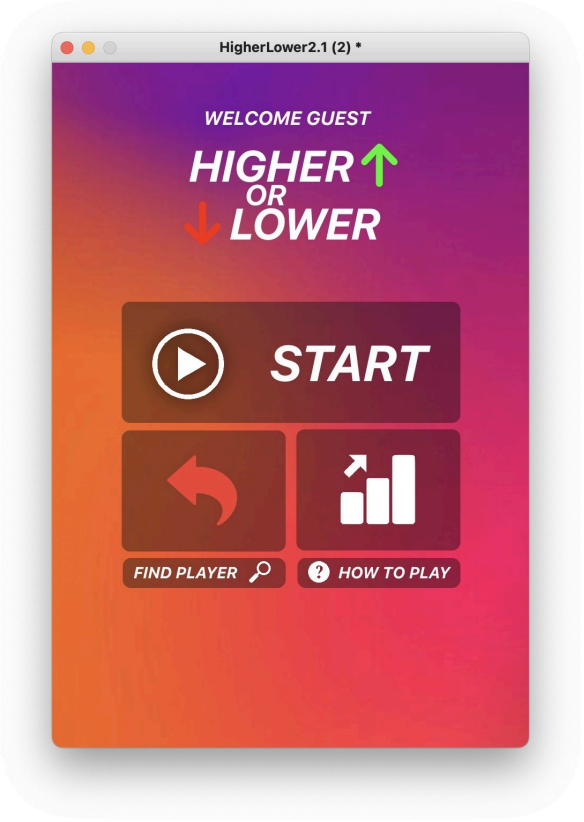
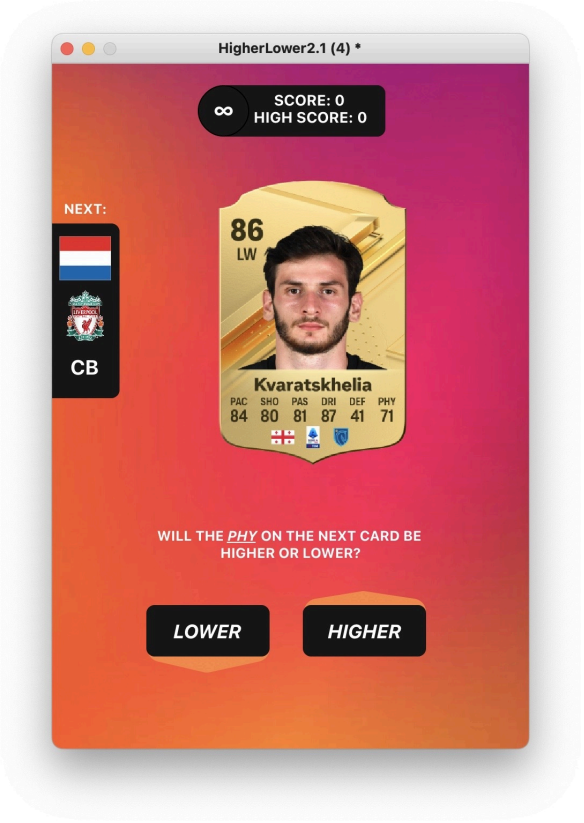
their respective usernames). *Test Method:* The table should display the first 10 records after sorting. The data in the table will be compared to the data in the array to make sure it is displaying the correct information.	 Screenshot 2: 	usernames, and high scores sorted from highest to lowest. The first 10 records of the sorted array of records are shown. Screenshot 2: Database capture of all records
Functional Requirement 5.7 and 6.2: A button in the bottom left corner returns the user to the home screen	Screenshot 1:	Screenshot 1: The user is returned to the home screen when they press the back button.

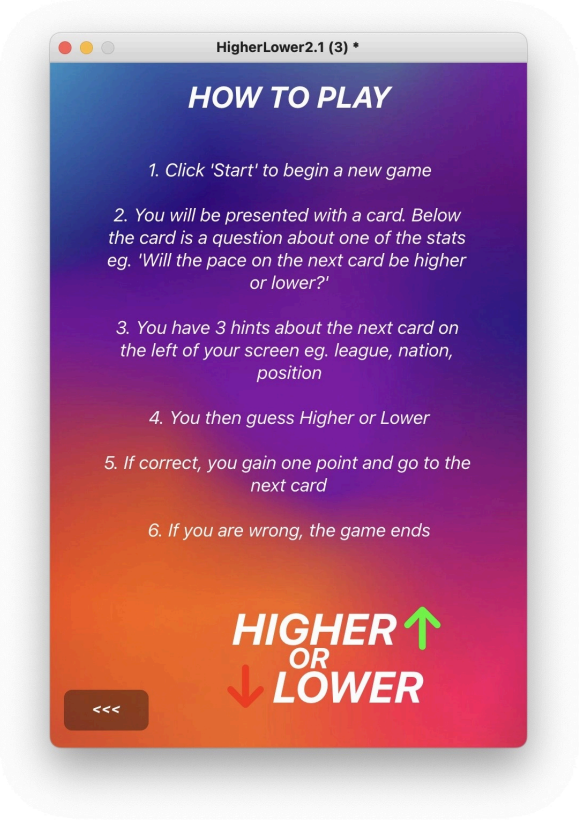
Test Method: This button will be clicked and it should return the user to the home screen		
Functional Requirement 6.1: Users are taken to the tutorial screen where they can see instructions on how to play *Test Method:* The screen should change to the tutorial page and instructions should be visible to the user	Screenshot 1: 	Screenshot 1: Instructions are legibly visible to the user, and there is a button to return them home.
END USER REQUIREMENTS		

End User Requirement 1.1: Users can log in with an existing account using their credentials (username and password) *Test Method:* An existing username and password will be entered into the program and should successfully sign in	Screenshot 1:  Screenshot 2:  Screenshot 3: 	Details of an existing account are entered and the program successfully logs in
End User Requirement 1.2: Users can create an account with a unique username and a password *Test Method:* A new account with a unique	Screenshot 1:	Account creation process shown with input boxes for username, password, and a final output box to confirm account

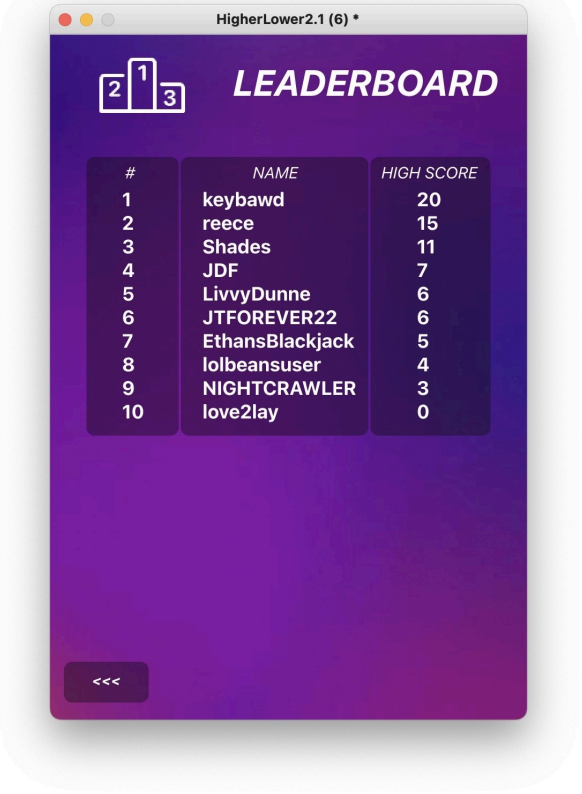
username and a password will be created and should be successful	 Screenshot 2:  Screenshot 3: 	creation.
End User Requirement 1.3: Users can play as a guest in which progress will not be saved *Test Method:* The play as guest button should take users straight to the home screen, and any high score they achieve is not saved to the database.	Screenshot 1:	Shows as 'Guest' at the top of the screen, when a new high score is achieved the updateDatabase procedure is not run as the session username variable is empty, set when the 'Play as guest' button is clicked.

	 Screenshot 2: loggedinUsername Screenshot 3: 4 if loggedinUsername <> "" then 5 updateDatabase 6 end if 7 end if 8 9 put "SCORE:" && currentScore & cr & "HIGH	
End User Requirement 2.1: Users can start a game, logout, view the leaderboard, search for a user's score, or view instructions *Test Method:* The buttons listed should be visible to the user and should be clickable - Start Game - Logout - View Leaderboard - Search User - View Instructions	Screenshot 1:	Screenshot 1: All buttons listed are visible to the user Screenshot 2: Start Game button takes you to gameplay screen with a ready to play game Screenshot 3: Tutorial button takes you to the tutorial page Screenshot 4: Leaderboard

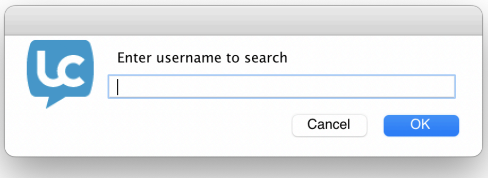
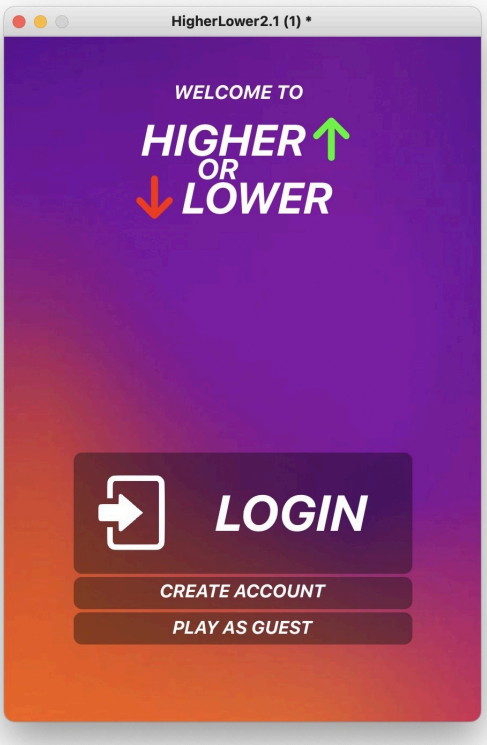
	 Screenshot 2:  Screenshot 3:	button takes you to the leaderboard page Screenshot 5: Find Player button opens up input box Screenshot 6: Logout button returns you to the login page

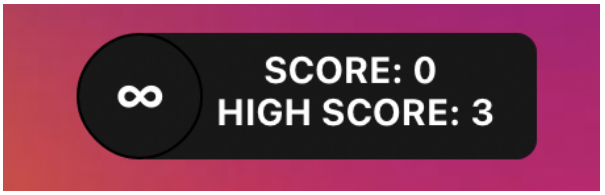
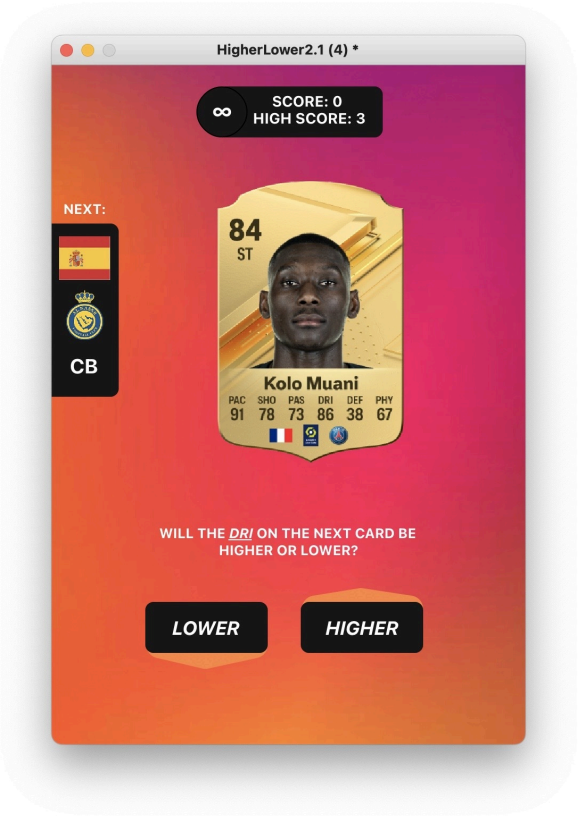
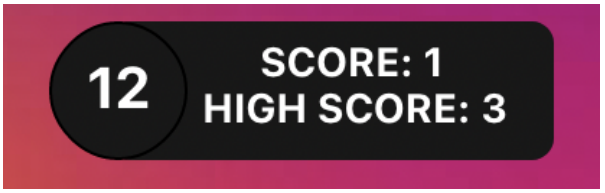
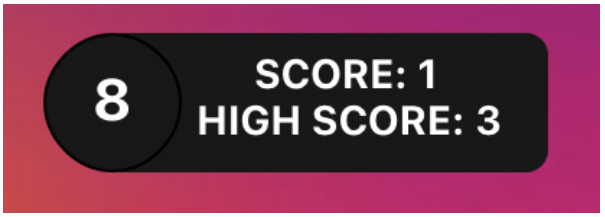


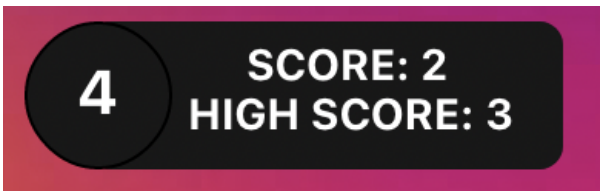
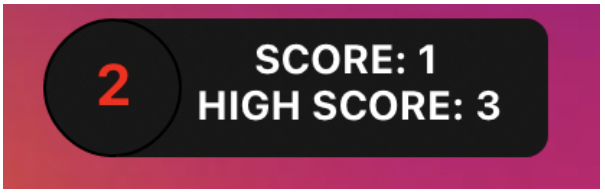
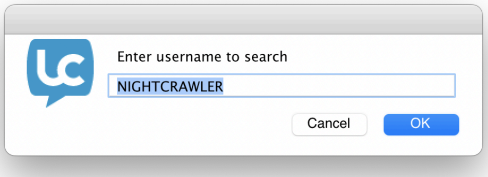
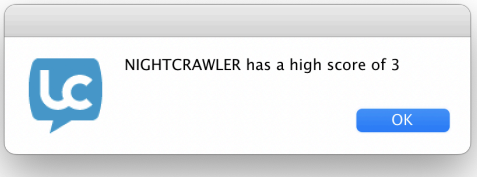
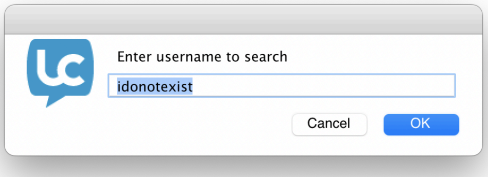
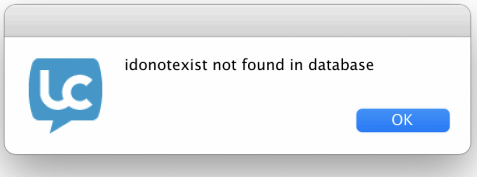
Screenshot 4:

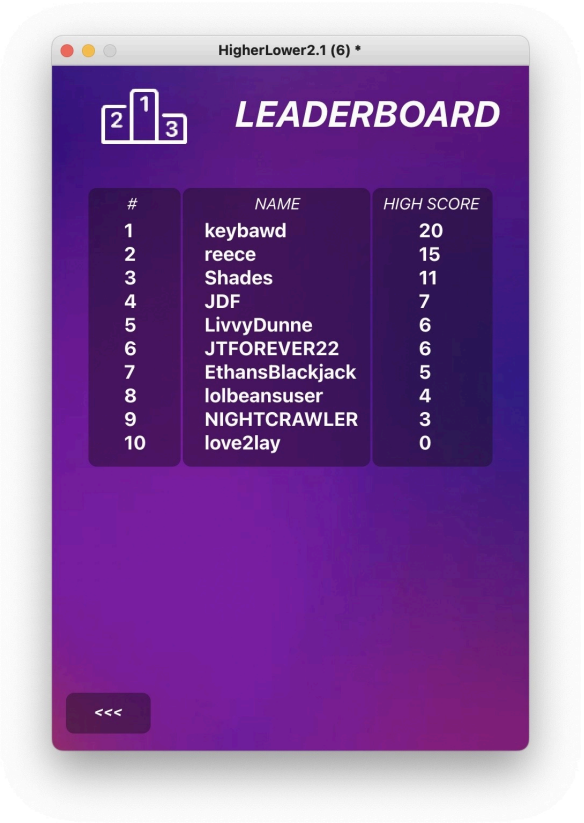


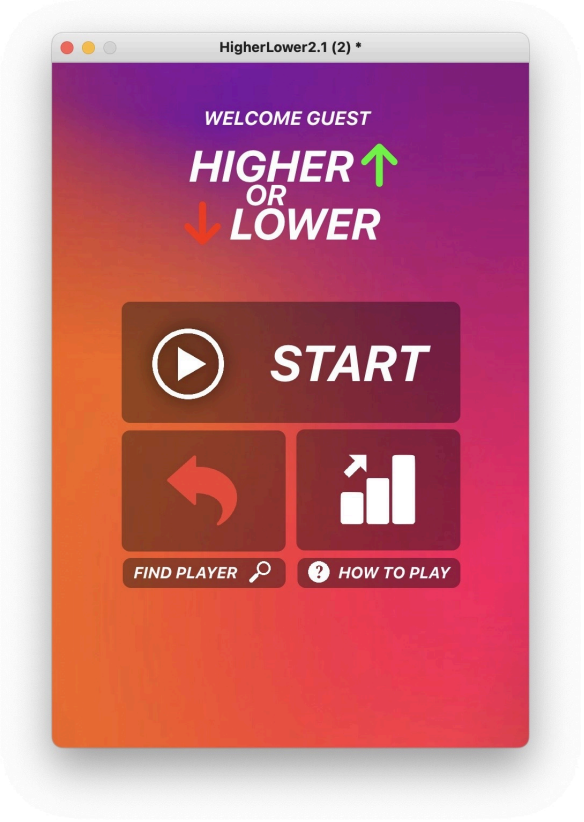
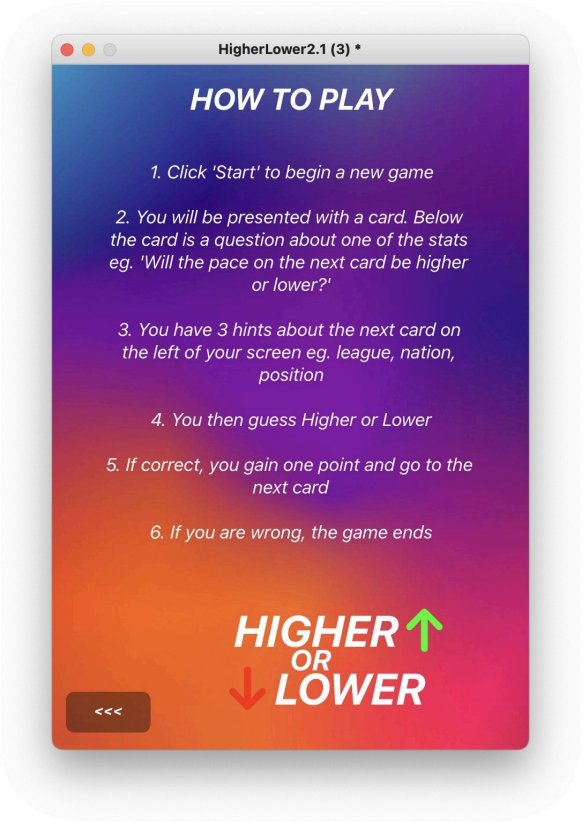
Screenshot 5:

	 Screenshot 6: 	
End User Requirement 3.1: Users can click the 'Higher' or 'Lower' button to submit their guess *Test Method:* The higher and lower button should be clickable by the user and will submit their guess to the program	Screenshot 1:  Screenshot 2: `submitGuess "LOWER"` `submitGuess "HIGHER"`	Screenshot 1: Higher/lower buttons Screenshot 2: Code in the 'Lower' button passes in the guess as a parameter Screenshot 3: Code in the 'Higher' button passes in the guess as a parameter
End User Requirement 3.2:	Screenshot 1:	Score and high

Users can view their current score and high score *Test Method:* The current score the user has as well as their high score (if logged in/session high score) should be visible at the top of the screen while playing		score are visible at the top of the screen during gameplay
End User Requirement 3.3: Users can view the current selected player card and the 3 hints for the next card *Test Method:* The current player should be shown in the middle of the screen and hints for the next card visible on the left	Screenshot 1: 	Current card image and hints for the next card are visible on screen in the correct positions
End User Requirement 3.4: Users should be able to see how long they have left to make their guess *Test Method:* A countdown timer should be visible at the top of the screen next to the score and high score, and it should be counting down every second.	 	The screenshots show the timer counting down, and the last 3 seconds are red.

	 	
End User Requirement 4.1: Users can enter a username and be shown the high score of the user whose name they entered *Test Method:* An input box should appear to enter a username. If an existing username is entered an output should show with that username and the respective high score, otherwise throw an error.	Screenshot 1:  Screenshot 2:  Screenshot 3:  Screenshot 4: 	Screenshot 1-2: A username is entered and the score is displayed Screenshot 3-4: A non-existing username is entered and the program throws an error message to the user
End User Requirement 5.1: Users can view the top 10	Screenshot 1:	Screenshot 1: The leaderboard is

highest scores along with the username of the user that achieved the score *Test Method:* The usernames and scores of the top 10 highest scores should be shown on screen.		shown the the top 10 scores and the user who achieved that score
End User Requirement 5.2 and 6.2: Users can return to the home screen *Test Method:* A 'back' button should be pressed and will take the user back to the home screen	Screenshot 1:  Screenshot 2:	Screenshot 1: The back button Screenshot 2: The user is now back on the home screen after clicking the button

		
End User Requirement 6.1: Users can view the instructions on how to play the game *Test Method:* A paragraph containing instructions of how to play the game should be shown on screen to the user	Screenshot 1: 	Screenshot 1: Instructions are shown to the user on screen and are clear and legible

Persona Testing

Results of the test cases

Case 1: Ethan Hamilton

- The user should try to create an account
    - Ethan achieved this with no problem and did not fail any of the input validation
- The user should try to log in with their account
    - Ethan entered his username correctly but when it came to his password he misspelt it, leading the program to tell him so and cancel the operation. Ethan was annoyed that he could not just re-enter his password instead of having to log in again.
- The user should try and view the tutorial
    - Ethan achieved this with no problem and managed to navigate between the main menu and tutorial page
- The user should try to start and play the game
    - Ethan got stuck into the game easily and achieved a respectable score having fun while doing so. He commented that he would like to see randomised hints added at a later stage in the future to make the game more interesting.
- The user should try to view the leaderboard
    - Ethan was hesitant as to what button was for the leaderboard and accidentally logged himself out, requiring him to log in again. On the second attempt he managed to get to the leaderboard page and saw his score and username on there.
- The user should try to log out
    - Ethan managed to log out successfully on first attempt

Case 2: Jack Douglas Ford

- The user should try to create an account
    - Jack took time with this task but managed to get there with little complications, he had to re-enter his password once as it was not long enough
- The user should try to log in with their account
    - Jack's username he entered to log in did not exactly case match the username he created, so this made him have to re-enter. Jack entered his password correctly.
- The user should try and view the tutorial
    - Jack was able to view the tutorial but found the slightly smaller button for the tutorial to be harder to click
- The user should try to start and play the game
    - Jack was able to begin the game and took a few seconds to familiarise himself with how the game works. He started off making a few incorrect guesses but managed to get some correct answers after time.
- The user should try to view the leaderboard

        - Jack was able to view the leaderboard with no problems but did not see himself on there as his score was not high enough
- The user should try to log out
    - Jack was able to log out easily with no problems

EVALUATION

Fitness for purpose

The following are my functional and end-user requirements, and if they have been achieved to the correct standard (X) or not (-).

Requirement	**Completed**
1.1.1: The file of player cards will be read into an array of records. If this fails an error message will be shown.	X
1.1.2: A connection to the database will be opened.	X
1.1.3: If it does not already exist, a table in the database will be created.	X
1.1.4: Users will be prompted to enter their username. The username will be checked if it exists in the database. If no username was found the program will display an error.	X
1.1.5: If a username is found, a prompt to enter a password will appear. If the password entered by the user matches the related password in the database, the user will be sent to the home screen. If not, an error message will appear.	X
1.1.6: The connection to the database will be closed	X
1.2.1: The file of player cards will be read into an array of records. If this fails an error message will be shown.	X
1.2.2: A variable “loggedinUsername” will be set to empty, this allows certain procedures to know they do not need to update the database	X
1.2.3: A variable “loggedinHighScore” will be set to 0	X
1.2.4: The user will be sent to the home screen	X

1.3.1: A connection to the database will be opened	X
1.3.2: If it does not already exist, a table in the database will be created.	X
1.3.3: Users will be prompted to enter a username of length between 3 and 15 characters. The entered username will be validated to ensure it meets the length check and additionally that it is unique and does not already exist in the database. Error messages will be shown if the username is invalid/taken.	X
1.3.4: If the entered username is valid and unique, the user will be asked to enter a password. The password must be between 8-20 characters long and will be validated to ensure that is the case.	X
1.3.5: A new record will be inserted into the database with the username and password chosen by the user, the high score of the new account will be set to 0.	X
1.3.6: Once the account is created the user will then be prompted to login to start playing.	X
1.3.7: The connection to the database will be closed	X
1.4.1: User will be sent back to the login screen	X
2.1: Users are presented with buttons to Play Game, View Leaderboard, Search User, View Instructions, and Logout.	X
3.1.1: The UI elements are reset to their starting position	X
3.1.2: The current score is set to 0, and countdown is not set as the first guess has no time limit. This is shown using an infinity symbol.	X
3.1.3: A starting card is randomly chosen from the array of records cardsArray[] and shown on screen	X
3.1.4: A rating is randomly chosen to be the	X

comparison condition and is shown on screen	
3.1.5: A next card is randomly chosen from the array of records cardsArray[]	X
3.1.6: 3 hints are then displayed for the next card	X
3.1.7: The user is then sent to the gameplay screen	X
3.2.1: Check if a countdown is currently running, if so, stop the countdown	X
3.2.2: Set the countdown to 16 seconds (not 15: read Design>Project design>Gameplay screen for reason why)	X
3.2.3: Start countdown timer	X
3.2.4 If the guess is correct, a tick icon is shown to the user, the next card is placed into the current card, a new condition is chosen, a new next card is chosen, new hints are shown, and the current card is shown to the user. The users score is incremented by 1, and this is updated on the screen	X
3.2.5: If the guess is incorrect or the timer reaches 0, a cross icon is displayed, followed by the image of the next card. The program checks if a new high score was achieved, if a user is logged in, the database is updated to reflect this. The game over screen is then shown with the user's score and high score. If the user's score was 0 or a new high score was achieved, a message is shown on the game over screen to reflect this.	X
4.1: A connection to the database will be opened	X
4.2: The database will be queried and the username and high score will be inserted into an array of records called userScores[]	X
4.3: The connection to the database will be closed	X
4.4: This array of records will then be bubble sorted by username alphabetically	X

4.5: User will be prompted to enter a username to search	X
4.6: If a username is entered, a binary search will be carried out to find the user's score.	X
4.7: If found, the username and high score will then be shown on screen, if not found the program will tell the user that it could not find that user in the database	X
5.1 A connection to the database will be opened	X
5.2: The database will be queried and the username and high score will be inserted into an array of records called userScores[]	X
5.3: The connection to the database will be closed	X
5.4: This array of records will then be bubble sorted by high score highest to lowest	X
5.5: This array of records will then be displayed in a table showing the first 10 records (the 10 highest scores and their respective usernames).	X
5.6: The user will be swapped to the leaderboard card	X
5.7: A button in the bottom left corner returns the user to the home screen	X
6.1: Users are taken to the tutorial screen where they can see instructions on how to play	X
6.2: A button in the bottom left corner returns the user to the home screen	X
7.1: The database will be queried upon login, creating accounts, searching users, and viewing the leaderboard	X
7.2: New records will be inserted when accounts are created	X
7.3: Records will be updated to reflect users' new high scores	X

1.1: Users can log in with an existing account using their credentials (username and password)	X
1.2: Users can create an account with a unique username and a password	X
1.3: Users can play as a guest in which progress will not be saved	X
2.1: Users can start a game, logout, view the leaderboard, search for a user's score, or view instructions	X
3.1: Users can click the 'Higher' or 'Lower' button to submit their guess	X
3.2: Users can view their current score and high score	X
3.3: Users can view the current selected player card and the 3 hints for the next card	X
3.4: Users should be able to see how long they have left to make their guess	X
4.1: Users can enter a username and be shown the high score of the user whose name they entered	X
5.1: Users can view the top 10 highest scores along with the username of the user that achieved the score	X
5.2: Users can return to the home screen	X
6.1: Users can view the instructions on how to play the game	X
6.2: Users can return to the home screen	X

My program is fit for purpose as it meets all the functional and end-user requirements, which is proved through my extensive requirements testing results. My persona testing proved very valuable and I was able to see that most test cases were able to be carried out easily by both personas, however the one takeaway is that some of my buttons could have been clearer with what they do. Inputs to my program were validated and errors were produced and shown to the user where needed. I am confident that my solution meets all my requirements identified in the analysis stage.

Robustness

My solution is robust as my inputs have been validated in my program code and database structure has validation. For example, during the account creation process the inputs are strictly validated to ensure they are within the correct length. If not, error messages are produced and shown to the user.

```
function checkValid stringToCheck, lowerLimit, upperLimit // function to check a string is the correct length
   local stringLength
   put length(stringToCheck) into stringLength

   if stringLength >= lowerLimit and stringLength <= upperLimit then
      return true
   else
      return false
   end if
end checkValid
```

Maintainability

Throughout my solution I have used meaningful variable names so my code can be understood by other developers and myself later if I need to come back to it. In terms of perfective maintainability, there is plenty of room to add new parts to my solution as it is modular so existing parts can be reused. For example, one of my personas identified that he would like the hints to be randomised so there can be different hints. In terms of adaptive maintenance, I could make my game playable on phone as the target demographic for players of this sort of game are on mobile devices, and it can reach a wider audience. In terms of corrective maintenance, I found nothing significant that needed to be corrected.

```
on createUsername // desired username for new account
   local valid, unique
   put false into valid
   put false into unique

   ask "Enter a username (must be between 3-15 characters)"
   put it into desiredUsername
   put checkValid(desiredUsername, 3, 15) into valid
   put checkUnique(desiredUsername) into unique
   repeat while valid = false or unique = false
      if desiredUsername = "" then
         exit to top
      end if
      ask "Retry. Username must be between 3-15 characters and unique"
      put it into desiredUsername
      put checkValid(desiredUsername, 3, 15) into valid
      put checkUnique(desiredUsername) into unique
   end repeat
end createUsername
```

I managed to get all of my planned features implemented and found my implementation not taking as long as I had planned, however my testing took longer than expected.