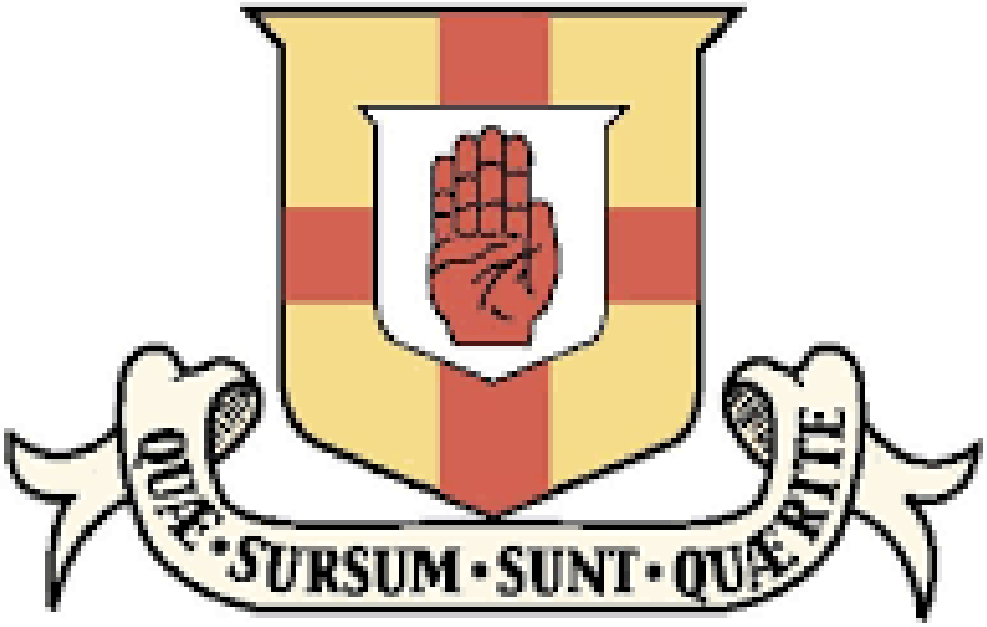
Software Systems Development

AS2 Coursework



Candidate Name: ***Callum White***
Candidate Number: ***8445***
Centre Number: ***71571***
Friends' School Lisburn

Source Code / Github

Table of Contents

Background

I designed my application to be a quiz for chess players of all abilities. My quiz will contain both chess trivia and gameplay questions. My quiz will also be appropriate for all ages and skill levels, as the questions asked will be representative of the skill / knowledge of the user.

I am very interested in chess, and I really enjoy playing and learning about the game. My passion for the game made it an obvious choice when deciding what my quiz should be about. In the past, I have made various versions of chess with C# and Windows Forms, but I had never made a full application based around it, so I decided to create one for my coursework.

Due to the fact that chess can be played by anyone, regardless of age or experience, the target audience for my quiz is naturally similar. The only prerequisite for playing the quiz is knowing how the rules work. I believe that my application will be popular among the target audience as it provides a way to challenge the user's chess ability and their knowledge of the game, which will encourage them to improve their abilities.

User Requirements

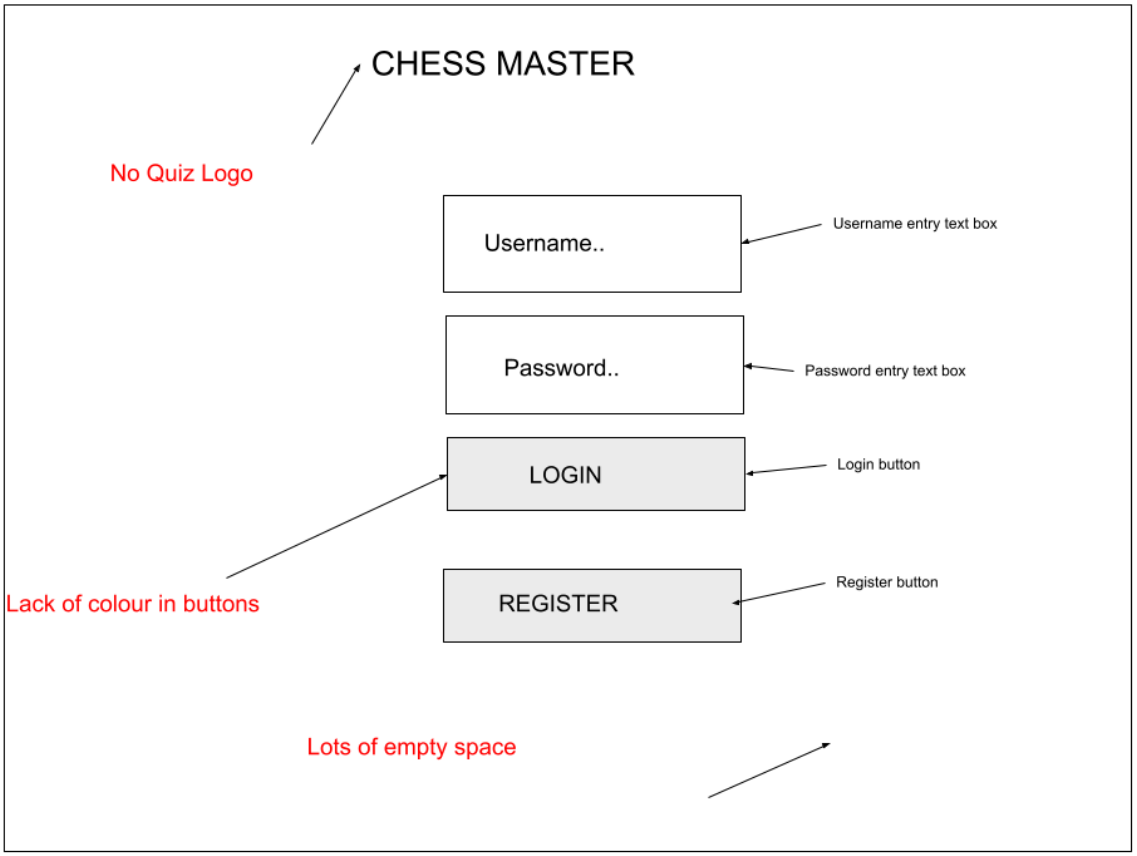
UR1	The application should have a splash screen
UR2	The application should have a login screen
UR3	The application should have a register screen
UR4	The application should have a main menu, with easy navigation to all other menus
UR5	The application should have a leaderboard to display the user in relation to the other users.
UR6	The application should have different types of questions, both text based and puzzle based questions.
UR7	The application should keep track of the user's stats, such as their accuracy of getting questions right.
UR8	The application should tailor the difficulty of the questions to the skill ability of the user.
UR9	The user should be able to logout and into the application without exiting the application.
UR10	The application should have a settings screen where they are able to change their password and profile picture.
UR11	An admin should be able to create new admins.

UR12	An admin should be able to create new puzzles and questions
UR13	An admin should be able to view all current puzzles and questions, and delete them.

Initial Storyboards

Black text highlights important parts of the design
Red text signifies user feedback to the design.

Login Page



txtBoxUsername

This text box is for the user to enter their unique username.

- Type: Textbox
- Size: 150, 40
- Location: 125, 100
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px

txtBoxPassword

This text box is for the user to enter their password.

- Type: Textbox
- Size 150, 40
- Location: 125, 150
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px

btnLogin

This button checks if the user's login details are correct and if so, logs them into the application.

- Type: Button
- Size: 150, 32
- Location: 150, 250
- ForeColour: Black
- BackColour: Gray
- Font: Arial
- Font Size: 17px

btnRegister

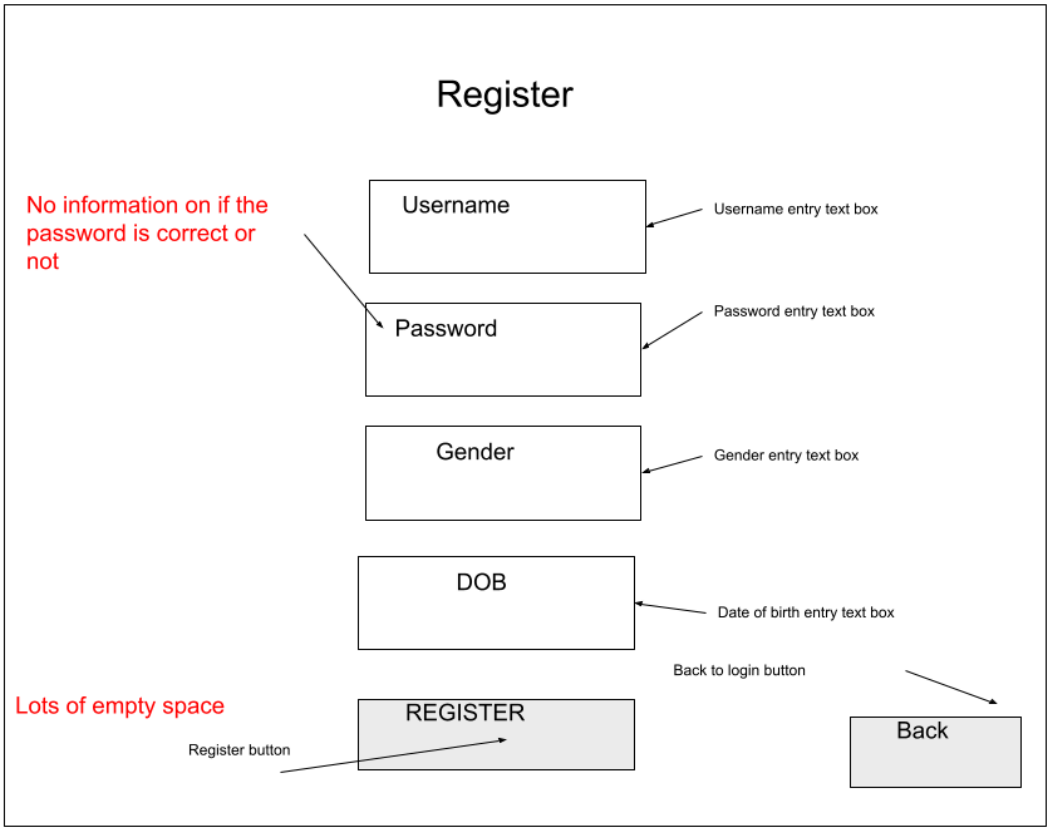
This button takes the user to the register page.

- Type: Button
- Size: 150, 32
- Location: 150, 325
- ForeColour: Black
- BackColour: Gray
- Font: Arial
- Font Size: 17px

lblChessMaster

This is a title label which shows the name of the application.

- Type: Label
- Size: 250, 40
- Location: 75, 50
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 25px



Register Page

txtBoxUsername, txtBoxPassword, txtBoxGender, txtBoxDob

These text boxes allow the user to input their own details.

- Type: Textbox
- Size: 200, 50
- Location: 150, 50 - 250
- ForeColour: Black
- BackColour: White
- Font: Arial
- FontSize: 17px

btnRegister

This button registers the new user and signs them in automatically.

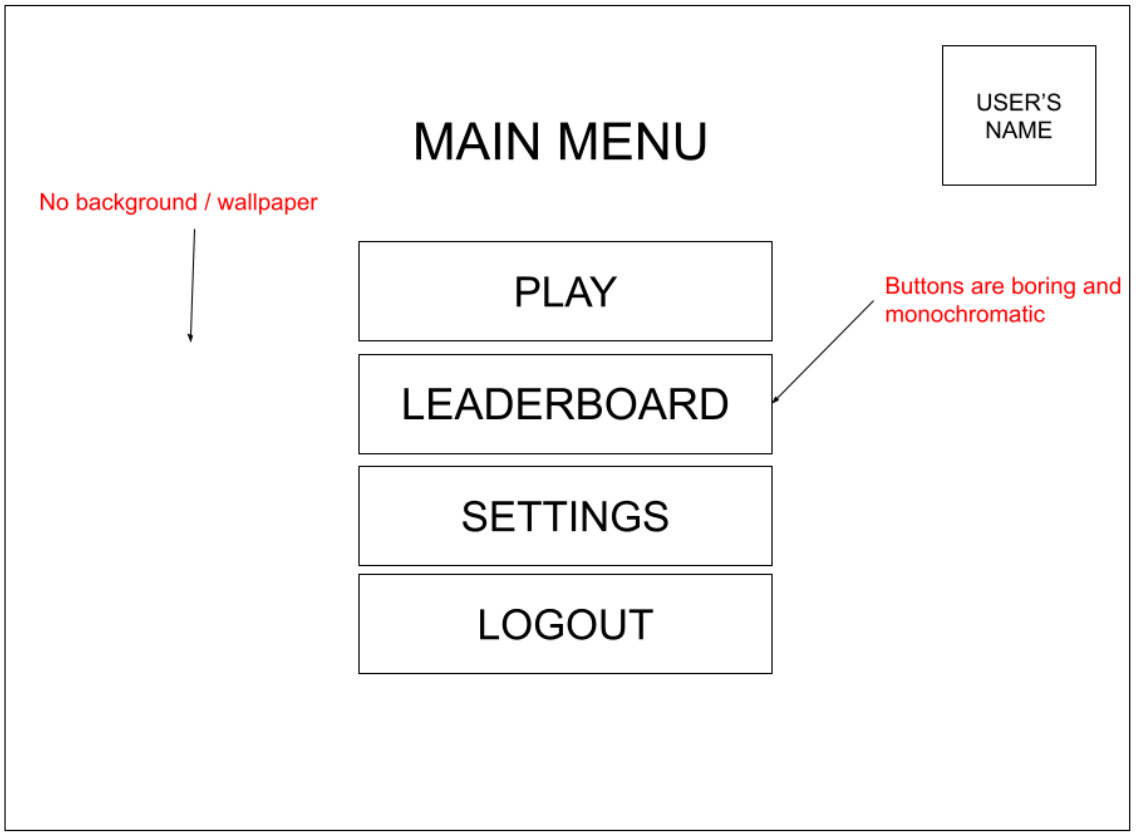
- Type: Button
- Size: 200, 50
- Location: 150, 300
- ForeColour: Black

- BackColour: Gray
- Font: Arial
- FontSize: 17px

btnBack

This button returns the user to the Login menu without registering them.

- Type: Button
- Size: 100, 40
- Location: 300, 300
- ForeColour: Black
- BackColour: White
- Font: Arial
- FontSize: 15px



Main Menu Page

lblMainMenu

This label is for indicating to the user where they are in the program.

- Type: Label
- Size: 100, 50
- Location: 150, 40
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 20px

btnPlay, btnLeaderboard, btnSettings, btnLogout

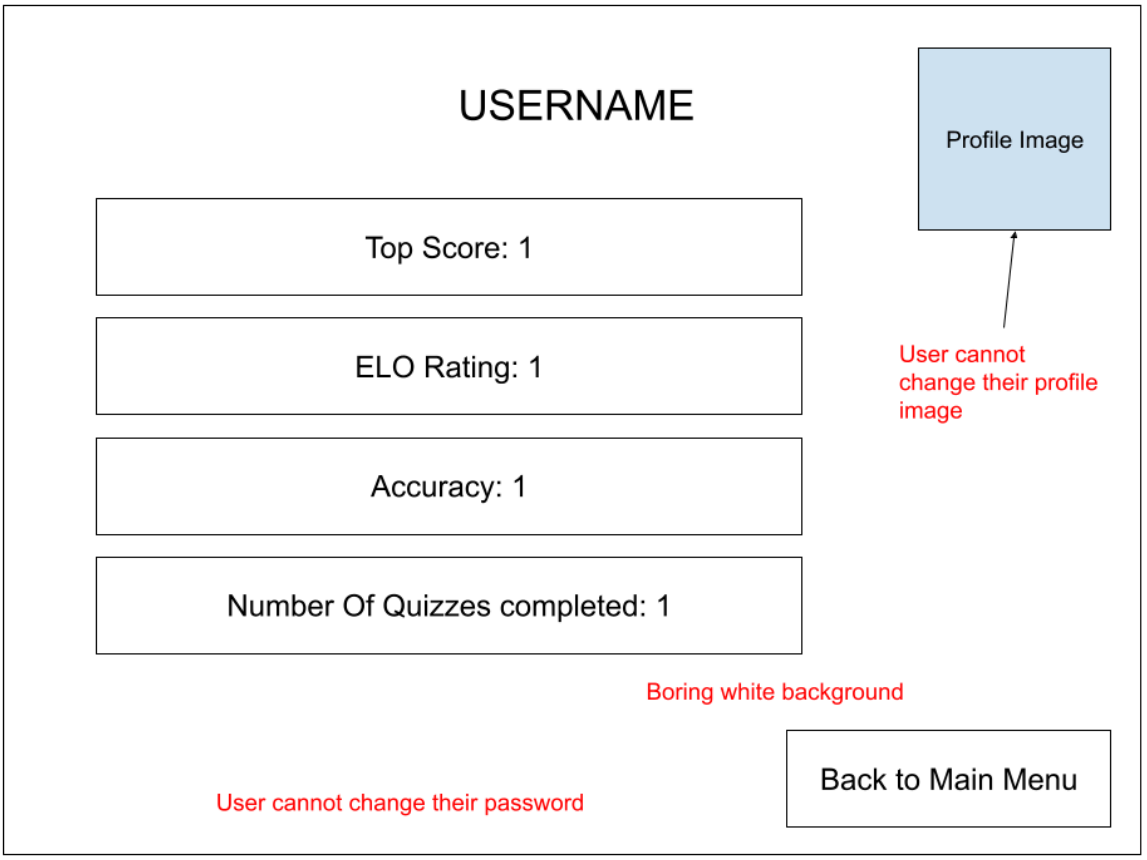
These buttons are for easy navigation to each form the user might want to visit.

- Type: Button
- Size: 150, 50
- Location: 125, 100 - 300
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 17px

btnUserProfile

This is for navigation to the user’s profile page.

- Type: Button
- Size: 50, 50
- Location: 340, 20
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 17px



User Profile

lblUsername

This label is to show the user which user they are viewing.

- Type: Label
- Size: 150, 40
- Location: 125, 20
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 20px

lblTopScore, lblEloRating, lblAccuracy, lblQuizzesCompleted

These all show data about the user, such as their accuracy and top scores.

- Type: Label
- Size: 200, 50
- Location: 50, 70 - 270
- ForeColour: Black
- BackColour White
- Font: Arial

- Font Size: 17px

btnMainMenu

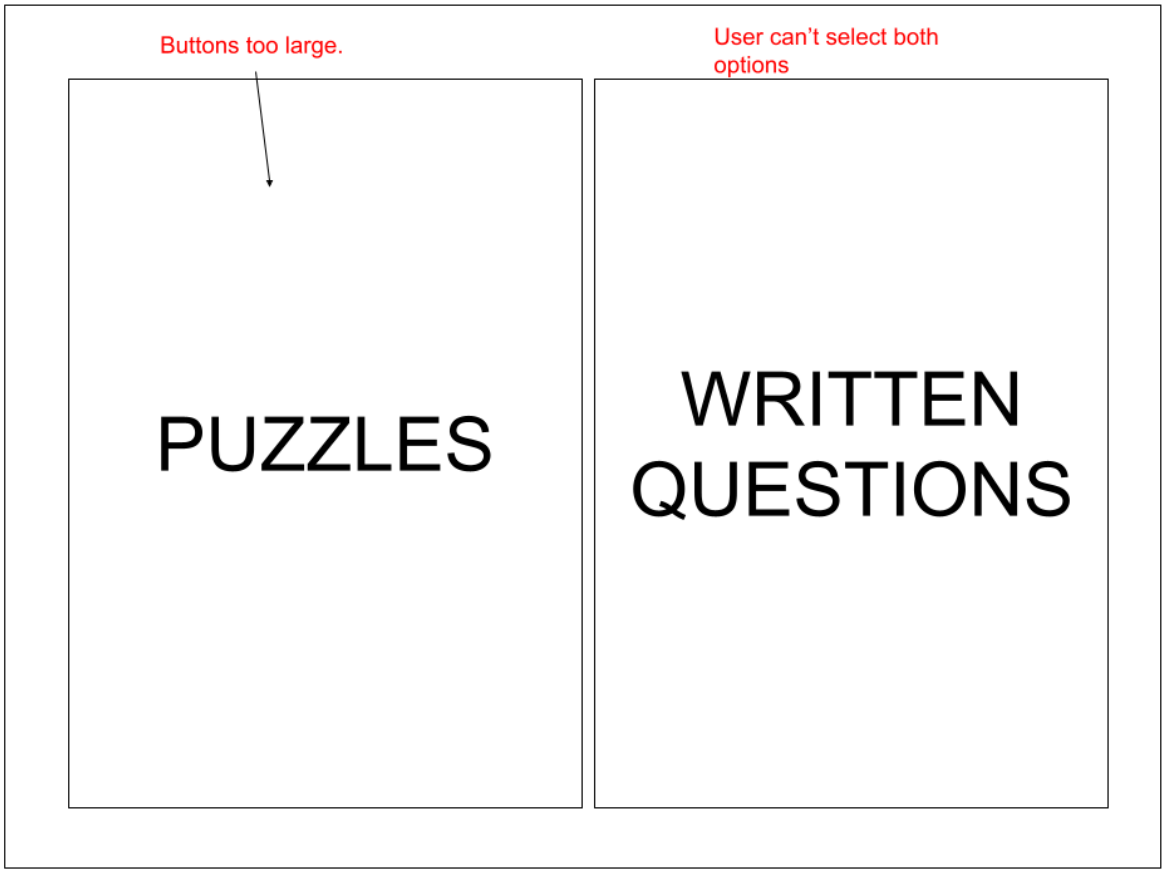
This button returns the user to the main menu.

- Type: Button
- Size: 100, 50
- Location: 300, 325
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px

pBoxProfileImage

This displays the user's profile image.

- Type: Picture Box
- Size: 75, 75
- Location: 300, 30
- ForeColour: Image
- BackColour: Image



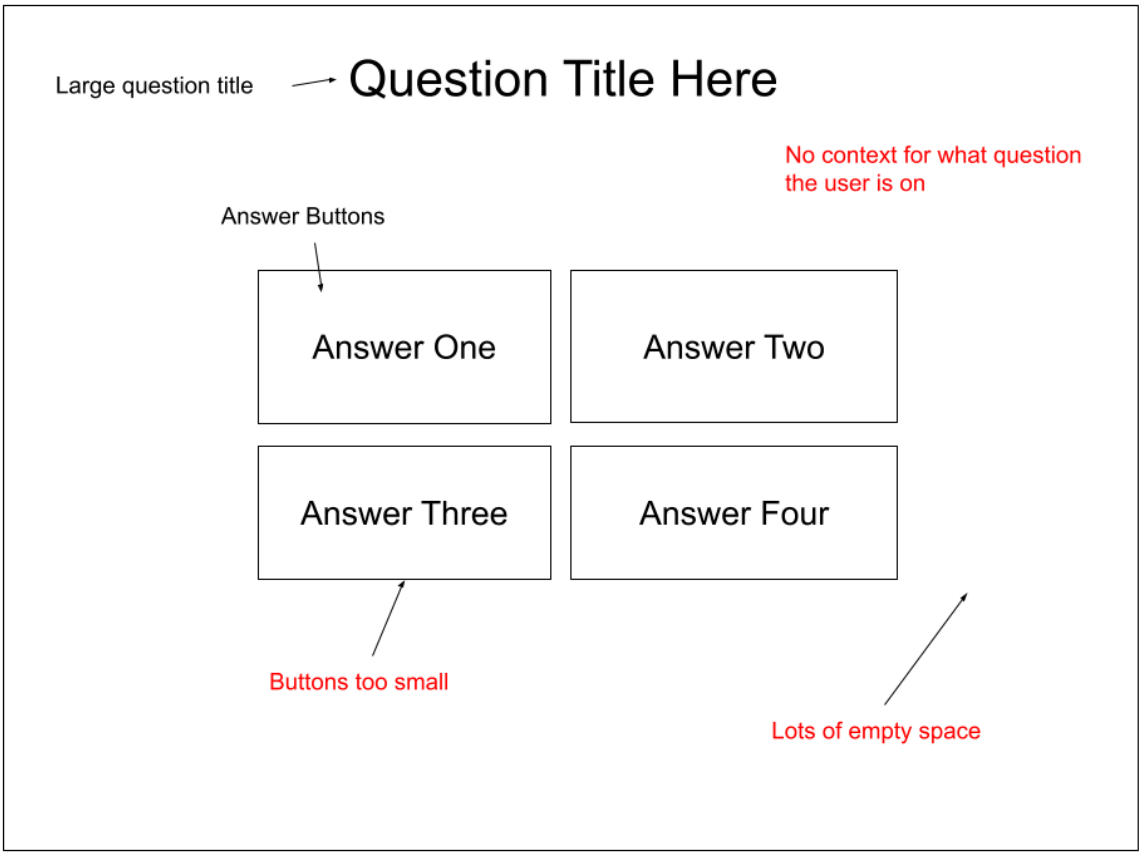
Choose Question Form

This form is for the user to choose what type of questions they want to be quizzed on. The puzzles are chess positions and the written questions are chess trivia.

btnPuzzles, btnWrittenQuestions

These buttons allow the user to select their choice.

- Type: Button
- Size: 175, 350
- Location: 30, 30 - 230
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 35px



Written Question Answer Form

This form is used to display a question and take in an input from the user.Multiple of these forms are chained together to test the user's knowledge of chess trivia.

lblQuestionTitle

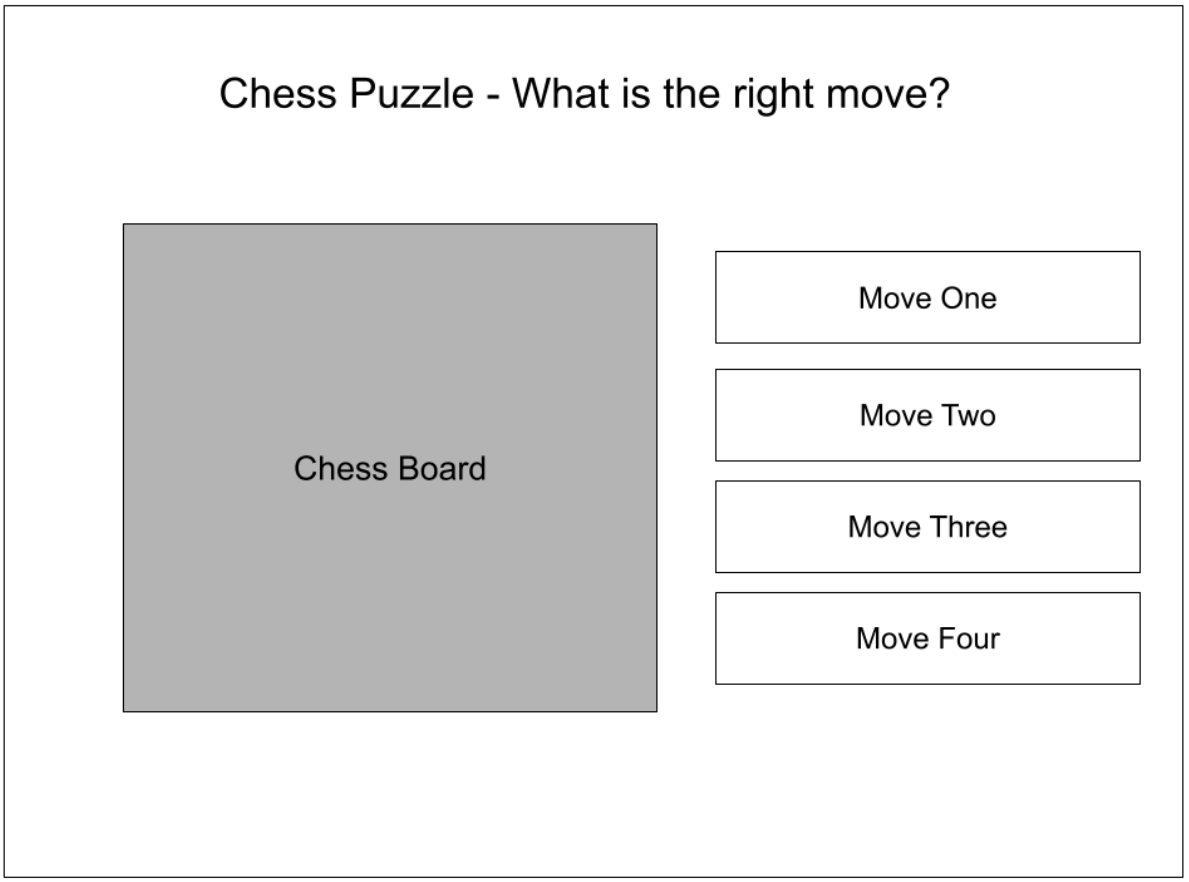
This label is used to ask the question to the user.

- Type: Label
- Size: 200, 75
- Location: 100, 40
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 25px

btnAnswerOne, btnAnswerTwo, btnAnswerThree, btnAnswerFour

These buttons allow the user to select their chosen answer.

- Type: Button
- Size: 80, 50
- Location: 140 - 220, 175 - 225
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 18px



Chess Puzzle Form

This form displays a chessboard with a puzzle that the user can answer by selecting the next move to win from the options

lblChessPuzzle

The title of the form.

- Type: Label
- Size: 250, 75
- Location: 75, 40
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 20px

pnlChessBoard

This panel can display any preset list of pieces in a custom position.

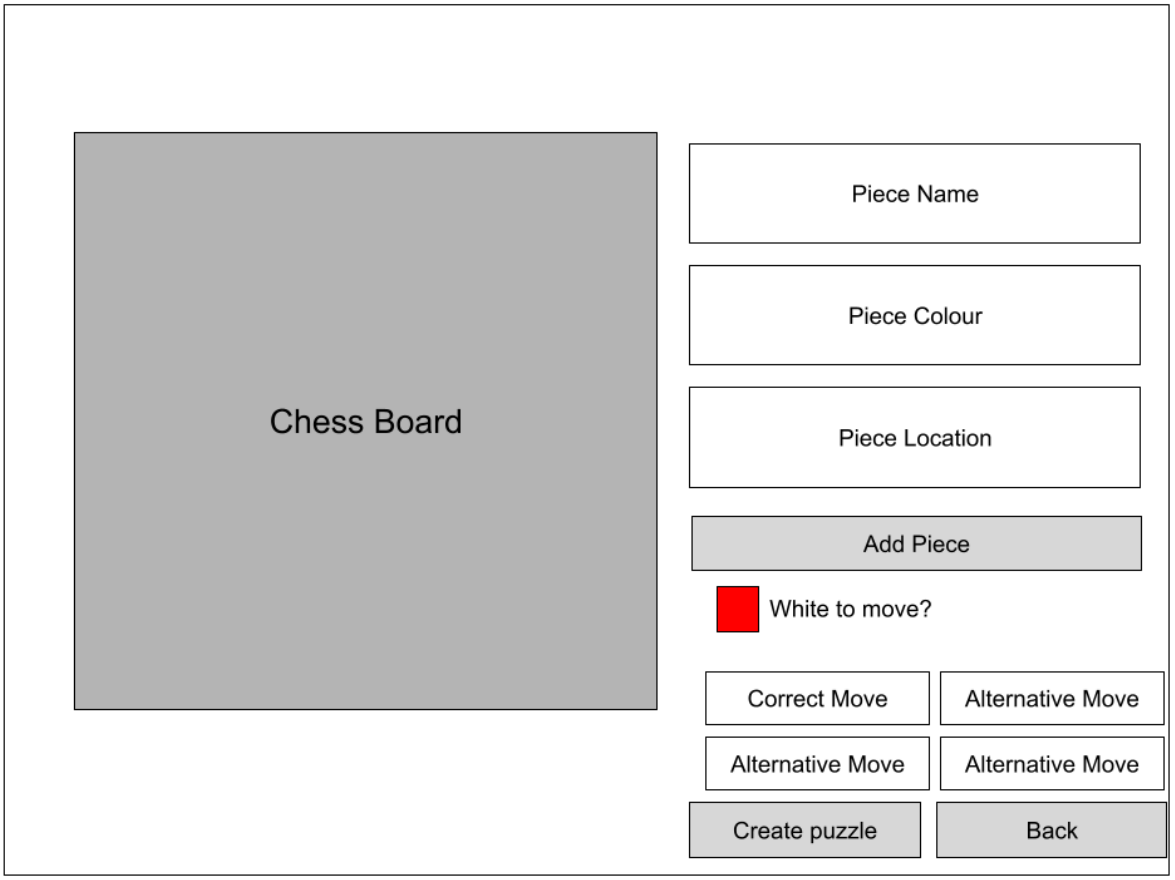
- Type: Panel
- Size: 200, 200

- Location: 40, 80
- BackColour: White, Black

btnMoveOne, btnMoveTwo, btnMoveThree, btnMoveFour

These buttons contain the possible move continuations from the chess position. Only one move is correct.

- Type: Button
- Size: 100, 50
- Location: 280, 100 - 350
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 18px



Chess Puzzle Creator

This is to be used by the admin, to create more puzzles for the quiz.

pnlChessBoard

This chessboard displays the pieces for the current puzzle being created.

- Type: Panel
- Size: 150, 150
- Location: 75, 20
- BackColour: Black, White

txtBoxPieceName, txtBoxPieceColour, txtBoxPieceLocation

These textboxes collect information about the piece to be created on the board

- Type: Text Box
- Size: 75, 35
- Location: 250, 40 - 120
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px

chckBoxWhiteToMove

This checkbox indicates to the user whether or not it is white to move.

- Type: Check Box
- Size: 60, 40
- Location: 250, 200
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px
- Check Colour: Red

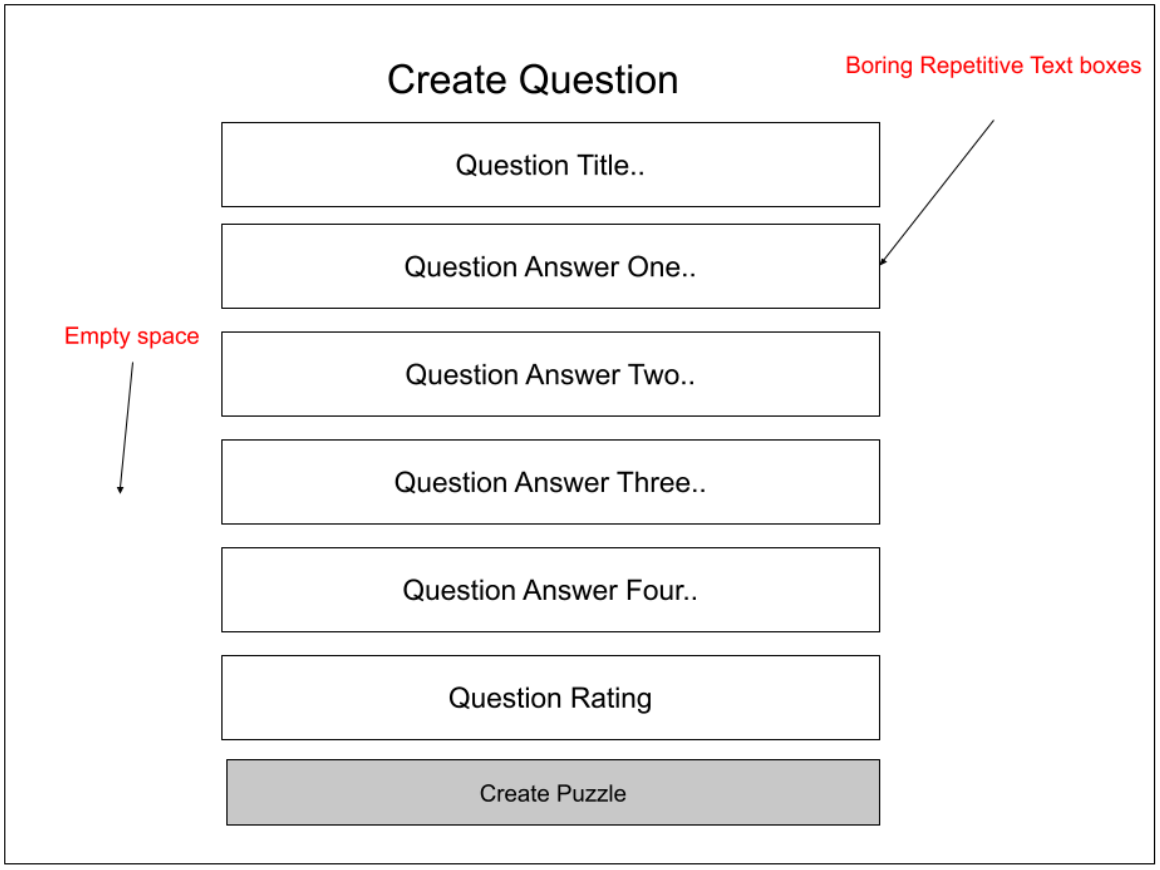
txtBoxCorrectMove, txtBoxAlternativeOne, txtBoxAlternativeTwo, txtBoxAlternativeThree

These text boxes allow the admin to enter the correct move and some alternative, incorrect moves. The user will then be able to choose one of the moves in a puzzle question.

- Type: Text Box
- Size: 75, 30
- Location: 250 - 325, 250 - 300
- ForeColour: Black
- BackColour: White

- Font: Arial
- Font Size: 15px

btnCreatePuzzle
This puzzle



Create Question Admin Form

This form allows the user to create a new question.

lblCreateQuestions
This is a title for the form to indicate to the user that they are creating a question.

- Type: Label
- Size: 125, 30
- Location: 100, 30
- ForeColour: Black
- BackColour: White
- Font: Arial

- Font Size: 20px

txtBoxQuestionTitle, txtBoxQuestionAnswerOne, txtBoxAnswerQuestionTwo, txtBoxAnswerQuestionThree, txtBoxAnswerQuestionFour, txtBoxQuestionRating

These text boxes allow the admin to enter all the information required to create a text based question.

- Type: Text Box
- Size: 150, 45
- Location: 75, 85 - 350
- ForeColour: Black
- BackColour: White
- Font: Arial
- Font Size: 15px

btnCreatePuzzle

When this button is pressed, the information from the text boxes will be used to create a new text question.

- Type: Button
- Size: 150, 30
- Location: 75, 360
- ForeColour: Black
- Gray
- Font: Arial
- Font Size: 13px

Final Storyboards

Login Form

This form will be the first landing page for the user. Here they will be able to login to the application, and also create an account if they need to.

- Type: Form
- BackColour: Gray

btnExit

Allows the user to exit the application

- Type: Button

- Size: 103, 40
- Location: 685, 398
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 14.25px

pBoxLogo

This picture box displays one of many different chess images / logos.

- Type: Picture Box
- Size: 400, 400
- Location: 353, 21
- BackColour: Chess Logo Image

txtBoxUsername, txtBoxPassword

These text boxes allow the user to enter their details to login to the application; both their username and password.

- Type: Text Box
- Size: 222, 43
- Location: 56, 125 - 174
- ForeColour: Black
- BackColour: Light Gray
- Font: JetBrains Mono
- Font Size: 20.25px

btnLogin

This checks if the user's entered information is correct. If it is, then the user will be taken to the main menu.

- Type: Button
- Size: 222, 32
- Location: 56, 223
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 11.25px

btnRegister

This button will take the user to the *formRegister* form, where they are able to create a new account.

- Type: Button
- Size:181, 50
- Location: 76, 371
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 11.25px

lblLogin

This label tells the user that the following fields are to sign in / login to the application.

- Type: Label
- Size: 114, 32
- Location: 105, 61
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 24px

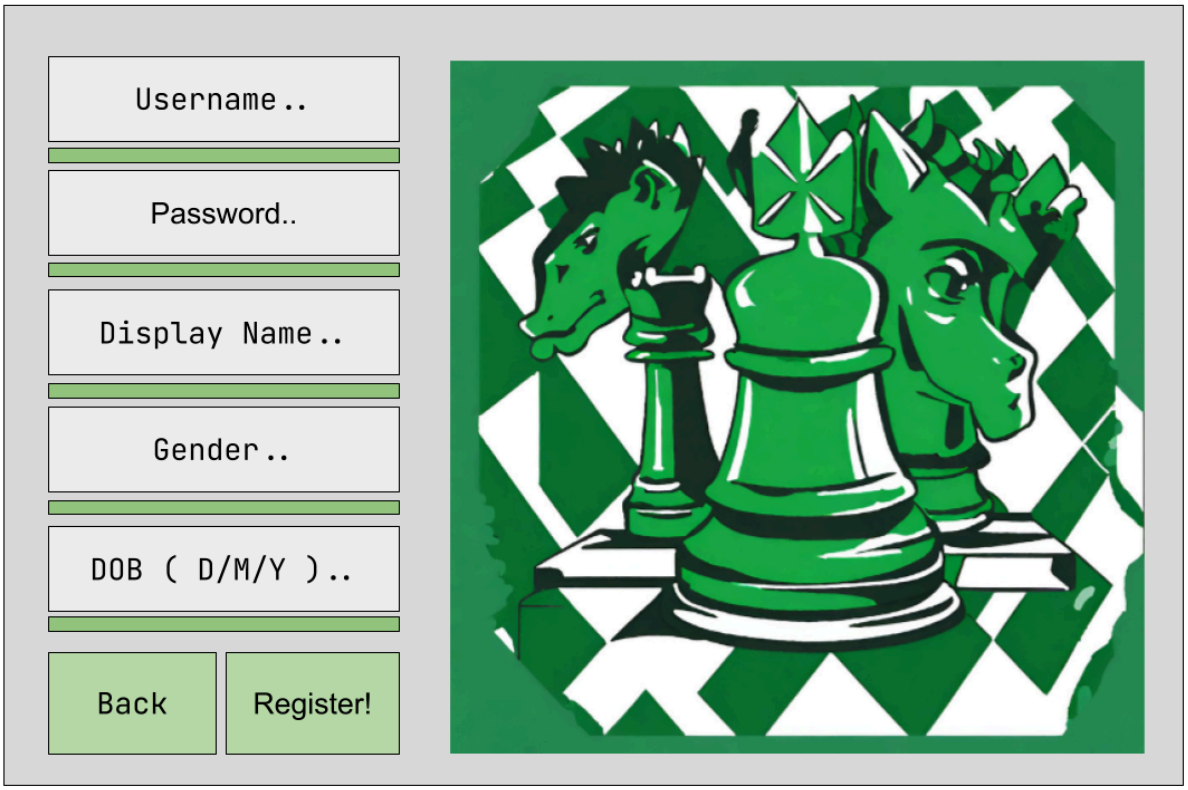
lblRegister

This label asks the user if they are new to the quiz. Below the label is the button, *btnRegister*, which the user can press to be taken to the appropriate form.

- Type: Button
- Size: 271, 36
- Location: 36, 325
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 20.25px

Register Form

This form allows the user to create a new account, and then login to the application.



btnBack

Allows the user to return back to the Login page

- Type: Button
- Size: 108, 63
- Location: 30, 357
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 28px

pBarUsername, pBarPassword, pBarDisplayName, pBarGender, pBarDob

These progress bars signal to the user when the requirements for the fields have been met. When all of the progress bars are completed, the user can then register their account.

- Type: Progress Bar
- Size: 249, 10

- Location: 47, 62 - 322
- BackColour: Light Green

pBoxLogo

This picture box displays one of many different chess images / logos.

- Type: Picture Box
- Size: 400, 400
- Location: 360, 20
- BackColour: Chess Logo Image

btnRegister

This button only enables once all of the appropriate fields are filled out. When clicked, the user will be taken to the main menu, and logged in with their new account.

- Type: Button
- Size: 174, 63
- Location: 146, 357
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 18px

txtBoxUsername, txtBoxPassword, txtBoxDisplay, txtBoxGender, txtBoxDob

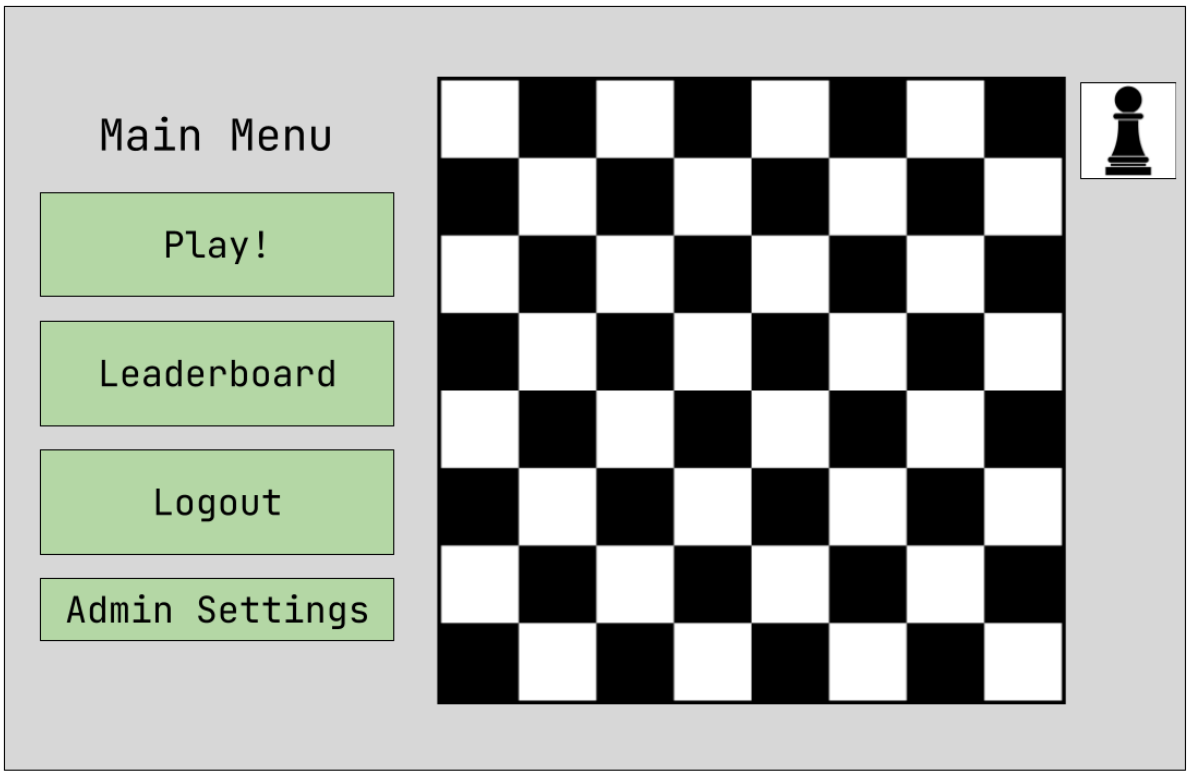
These text boxes allow the user to input the information needed for the registration.

- Type: Text Box
- Size: 249, 32
- Location: 47, 13 - 273
- ForeColour: Black
- BackColour: Light Gray
- Font: JetBrains Mono
- Font Size: 20.25px

Main Menu Form

This form is the centre point of the application. Every other menu is accessible from here.

- Type: Form
- BackColour: Gray



btnLogout

This button will log out the user from the application, sending them back to the *Login Form*.

- Type: Button
- Size: 253, 83
- Location: 24, 294
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 20.25px

btnSettings

This button will only be displayed if the user is an admin. If they are, then it will take them to the *Admin Settings* form.

- Type: Button

- Size: 253, 45
- Location: 24, 383
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 20.25px

btnPlay

This button will take the user to the *Choose Quiz Form*, where they will be able to start a quiz.

- Type: Button
- Size: 253, 83
- Location: 24, 116
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 20.25px

btnLeaderboard

This button will take the user to the *Leaderboard Form*, where they will be able to view the current leaderboard standings, and their relation to other players.

- Type: Button
- Size: 253, 83
- Location: 24, 205
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 20.25px

pBoxProfile

This picture box displays the users' current profile picture. When clicked, the user will be taken to their *Profile Page*.

- Type: Picture Box
- Size: 72, 65
- Location: 716, 12
- Image: User Profile Picture

lblMenu

This label indicates to the user that they are in the main menu.

- Type: Label
- Size: 220, 49
- Location: 40, 46
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 28px

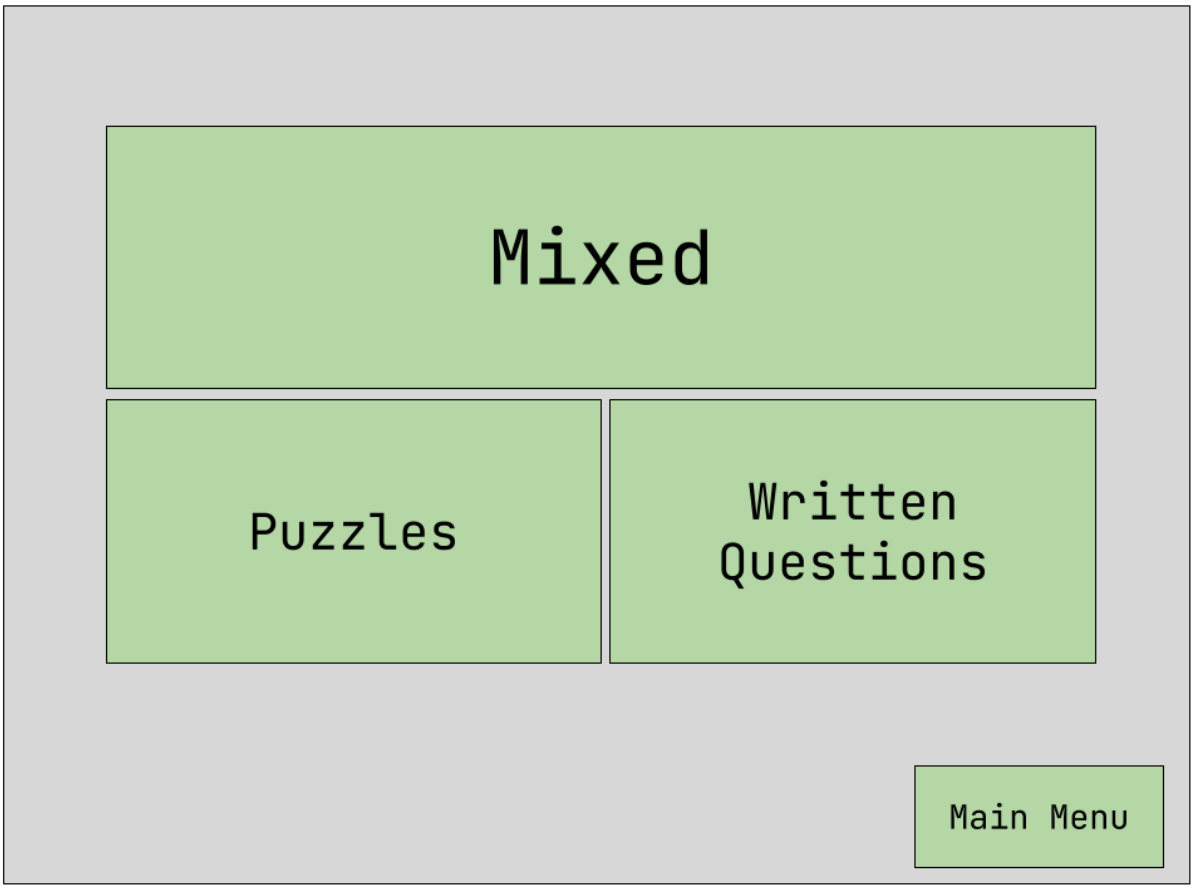
pnlBoard

This panel displays a chessboard which will be showing a random game from a library of famous chess games.

- Type: Panel / Board
- Size: 400, 400
- Location: 360, 20
- Image: Chess Board

Choose Quiz Form

This form allows the user to choose what type of question to receive. After choosing, they will then be redirected towards the appropriate question forms.



btnMixed

This button will send the user to a form which will present them with both puzzles and written questions.

- Type: Button
- Size: 528, 189
- Location: 127, 70
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 35px

btnPuzzles

This button will take the user to a form which will present them with only puzzles.

- Type: Button
- Size: 264, 189
- Location: 127, 300
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 25.25px

btnWrittenQuestions

This button will take the user to a form which will present them with only written questions.

- Type: Button
- Size: 264, 189
- Location: 397, 300
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 25.25px

btnMainMenu

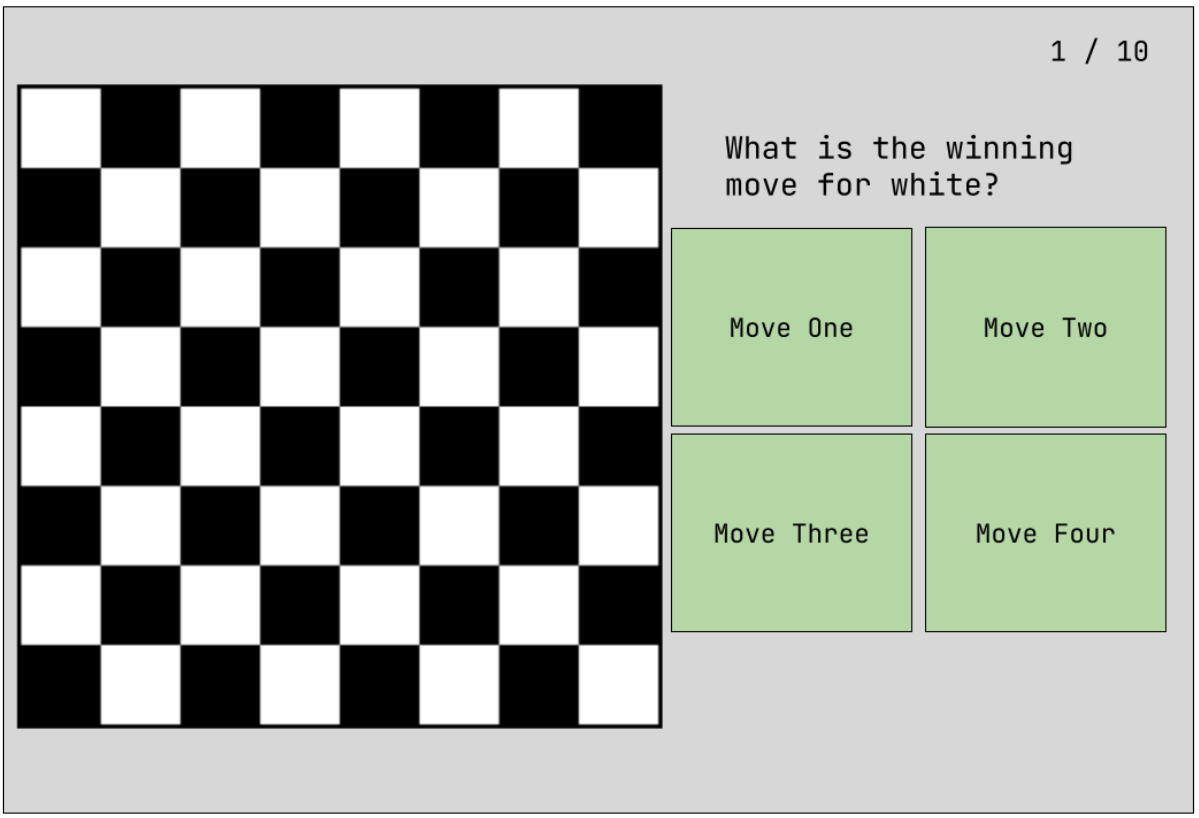
This button will take the user back to the main menu.

- Type: Button
- Size: 130, 45
- Location: 658, 393
- ForeColour: Black
- BackColour: Light Green

- Font: JetBrains Mono
- Font Size: 24.25px

Puzzle Question Form

This form displays a chess position and asks the user what the best move to make is. Four different moves are presented to the user, and they must pick one.



lblWinningMove

This label tells the user if the puzzle is “White to Move” or “Black to Move”

- Type: Label
- Size: 337, 27
- Location: 451, 77
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.5px

btnAnswer1, btnAnswer2, btnAnswer3, btnAnswer4

These buttons allow the user to choose which move they think is the best. When they click one of the buttons, the next puzzle will show up

- Type: Button
- Size: 170, 131
- Location: 442 - 618, 118 - 255
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 15.25px

pnlBoard

This panel displays a chess board with a custom position on it. The user will use this position to solve the puzzles.

- Type: Panel / Board
- Size: 400, 400
- Location: 20, 35
- Image: Chess Board

lblQuestionNumber

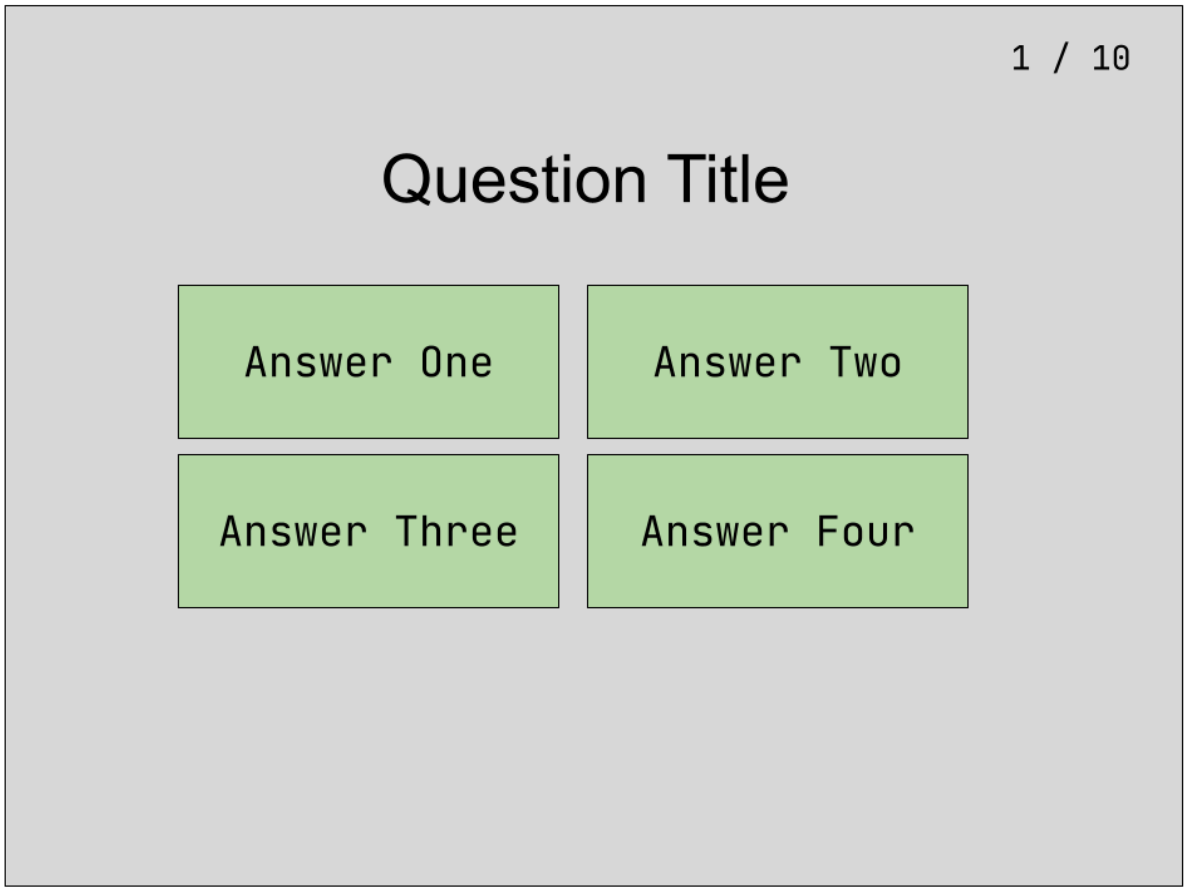
This label informs the user what question they are currently on, and how many questions they have left

- Type: Label
- Size: 90, 27
- Location: 698, 9
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.5px

Written Question Form

This form is to display a written question, which the user can then answer from any of the multiple choice questions.

- Type: Form
- BackColour: Gray



lblQuestionNumber

This label informs the user what question they are currently on, and how many questions they have left

- Type: Label
- Size: 90, 27
- Location: 698, 9
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.5px

btnAnswer1, btnAnswer2, btnAnswer3, btnAnswer4

These buttons allow the user to select the answer that they believe is correct.

- Type: Button
- Size: 277, 120
- Location: 129 - 412, 116 - 242
- ForeColour: Black

- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 15.25px

lblQuestion

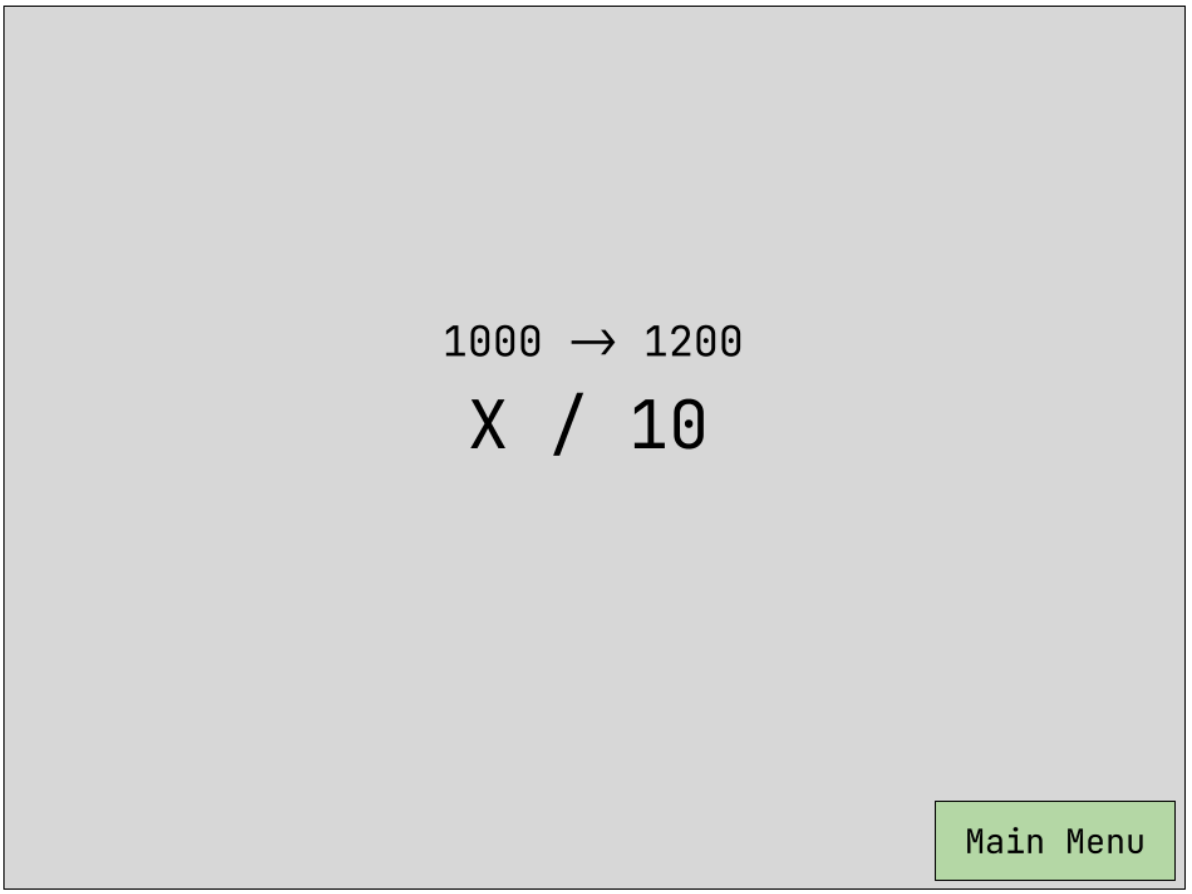
This label tells the user what the question they have to answer is.

- Type: Label
- Size: 143, 36
- Location: 341, 48
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.5px

Question Result Form

This form tells the user how many questions they got right in their quiz. It also displays the change in ELO value after answering the questions.

- Type: Form
- BackColour: Gray



btnMainMenu

This button will take the user back to the *Main Menu*

- Type: Button
- Size: 181, 53
- Location: 607, 385
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 15.5px

lblEloChange

This label displays the change in elo that the user went through due to the quiz.

- Type: Label
- Size:
- Location: 305, 19
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 27px

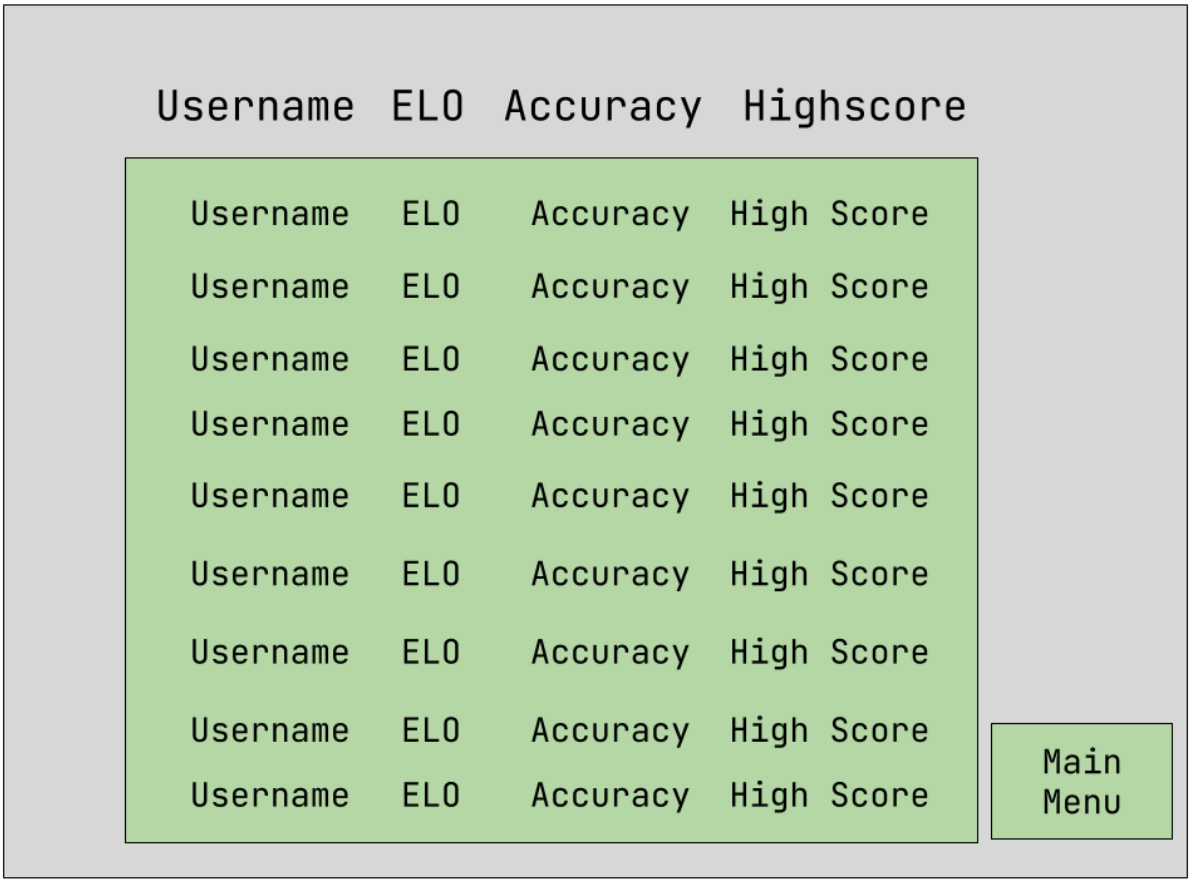
lblQuestionsCorrect

This label displays how many questions the user got right, and out of how many.

- Type: Label
- Size: 172, 63
- Location: 285, 174
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.5px

Leaderboard Form

This form will show to the user their ability and elo in relation to other users. It can be sorted in multiple different ways, from ELO to their Highscore.



btnMainMenu

This button will take the user back to the *Main Menu*

- Type: Button
- Size: 115, 51
- Location: 663, 387
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 15.5px

lblUsername, lblElo, lblAccuracy, lblHighscore

These labels show the user the type of information that will be in the representative column

- Type: Label
- Size: 100, 21
- Location: 100 - 500, 18
- ForeColour: Black
- BackColour: Gray

- Font: JetBrains Mono
- Font Size: 12px

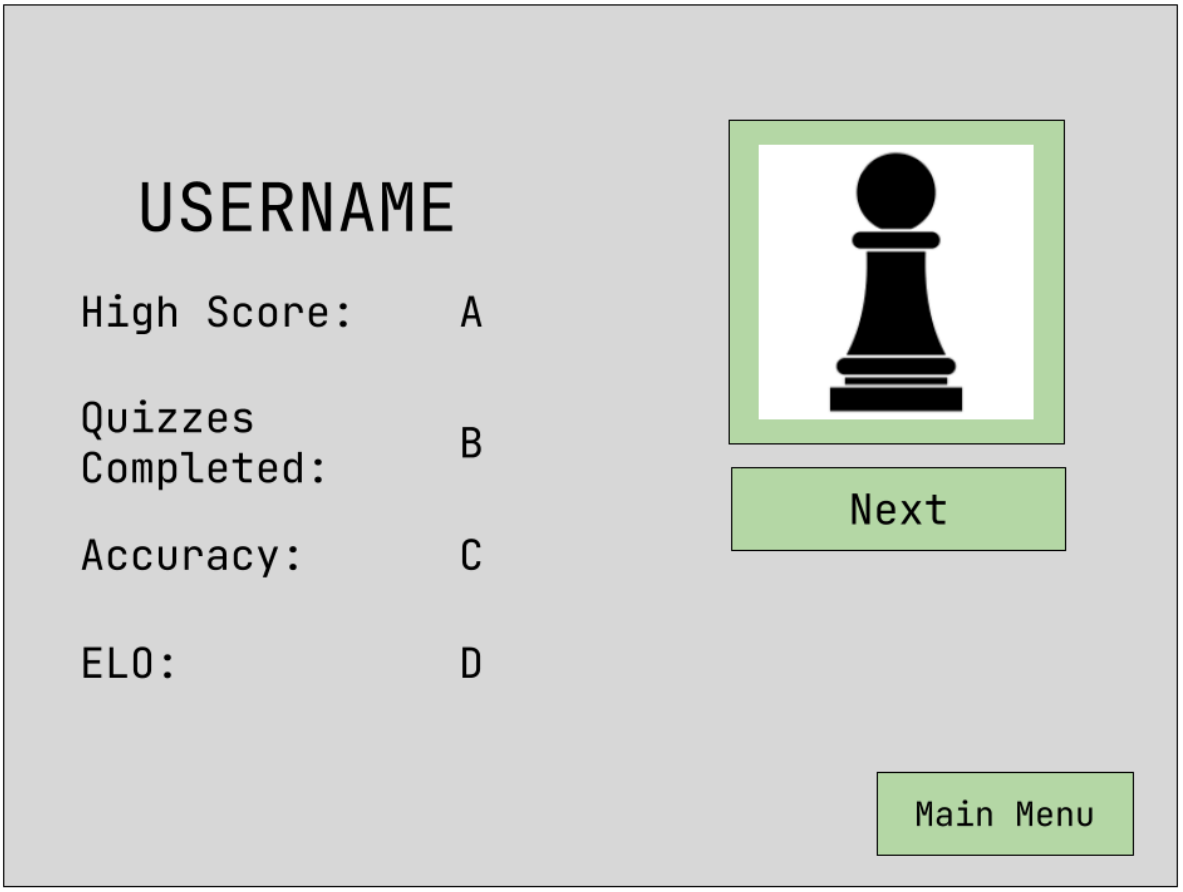
lblUserUsername, lblUserElo, lblUserAccuracy, lblUserHighscore

These labels will be cloned for each user in the leaderboard, and the user's details will be entered into the appropriate fields.

- Type: Label
- Size: 100, 25
- Location: 100 - 500, 20
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 16px

User Profile Form

This form displays all the necessary information for the user in an ordered fashion. It also allows the user to change their password and profile picture.



btnMainMenu

This button will take the user back to the *Main Menu*

- Type: Button
- Size: 115, 51
- Location: 663, 387
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 15.5px

lblUsername

This displays the current user's username, to provide clarity on whose profile is being viewed.

- Type: Label
- Size: 462, 50
- Location: 51, 9
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 30px

lblTopScore, lblQuizzesCompleted, lblAccuracy, lblElo

These labels tell the user what information is being displayed

- Type: Label
- Size: 200, 50
- Location: 50, 90 - 350
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 27px

lblTopScoreValue, lblQuizzesCompletedValue, lblAccuracyValue, lblEloValue

These labels show the actual values of the user's information. The user's details will be put in these labels.

- Type: Label
- Size: 62, 50
- Location: 150, 90 - 350

- ForeColour: Green
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 25px

pBoxProfileImage

This shows the user's current profile picture.

- Type: Picture Box
- Size: 171, 158
- Location: 559, 24
- Image: User Profile Picture

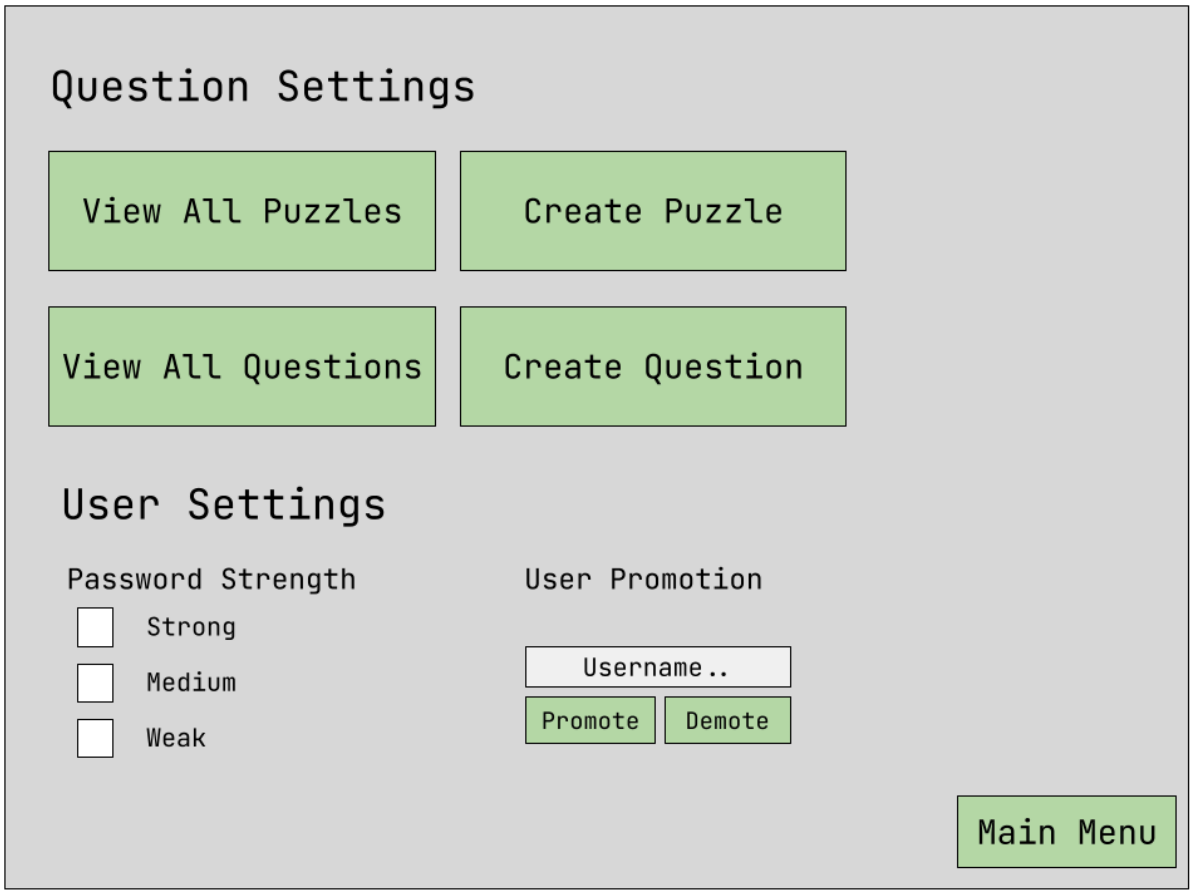
btnNextPicture

This button swaps to the next available profile image

- Type: Button
- Size: 171, 41
- Location: 559, 188
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 9px

Admin Menu Form

This form is a menu for all of the actions an Admin could take.



btnMainMenu

This button redirects the user to the *Main Menu.*

- Type: Button
- Size: 146, 44
- Location: 642, 394
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 16px

btnViewAllPuzzles, btnCreatePuzzle, btnViewAllQuestions, btnCreateQuestions

These buttons redirect the user to various different forms which allow them to both create or view certain questions / puzzles.

- Type: Button
- Size: 283, 79
- Location: 17 - 324, 70 - 155
- ForeColour: Black

- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 18px

lblQuestionSettings, lblUserSettings

These labels inform the user of different sections of the admin page

- Type: Label
- Size: 377 - 194, 47 - 27
- Location: 17 - 324, 9
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 24px

lblPasswordStrength

These labels inform the user that the following checkBoxes deal with the necessary strength of the passwords

- Type: Label
- Size: 150, 30
- Location: 374, 9
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 24px

chckBoxStrongPassword, chckBoxMediumPassword, chckBoxWeakPassword

These checkboxes allow a custom password strength to be selected.

- Type: Check Box
- Size: 83, 31
- Location: 78, 320 - 394
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.75px

lblUserPromotion

This label informs the user that the text box below deals with promoting other users to admins.

- Type: Label
- Size: 175, 35
- Location: 320, 308
- ForeColour: Black
- BackColour: Gray
- Font: JetBrains Mono
- Font Size: 15.75px

txtBoxUsernamePromotion

This textbox allows the admin to enter the name of the user they want to promote.

- Type: Label
- Size: 246, 35
- Location: 296, 338
- ForeColour: Black
- BackColour: White
- Font: JetBrains Mono
- Font Size: 15.75px

btnUserPromote, btnUserDemote

These buttons allow the user to either promote or demote the user who has the username entered in the textbox.

- Type: Label
- Size: 120, 32
- Location: 296 - 422, 279
- ForeColour: Black
- BackColour: Light Green
- Font: JetBrains Mono
- Font Size: 14.25px

Testing

Testing Plan / Unit Tests

Test Num	***Test Data***	***Test Params***	***Expected Outcome***	***Actual Outcome***	***Corrective Action***
			Splash Screen		
1.00	Launch applicatio n	None	Splash screen should be the first form displayed	Splash screen is the first form displayed	None
1.01	Splash screen location	None	Splash screen should open in the centre of the screen.	Splash screen opens in the centre of the screen.	None
1.02	Main Menu Redirect	None	Main menu should be opened when the progress bar finishes	The main menu is opened when the progress bar finishes	None
1.03	Splash Screen Chess Board	None	The chess board should be displayed in the centre of the screen	The chess board is displayed on the left hand side	The board is moved to the centre. Pg X.
			Login		
2.00	Enter username and password	Username: *root* Password: *root*	The user should be able to enter both their username and password	The user is able to enter their username and password	None
2.01	Login	Username: *root*	The user should be	The user is taken to the	None

		Password: *root*	taken to the login screen if their details are correct.	login screen when their entered details are correct.	
2.02	Login error	Username: *groot* Password: *groot*	The user should be indicated that their details are invalid	The user does not receive any alert that their details are incorrect	A label dialogue is shown to the user Pg X.
2.03	Exit	None	The application should quit	The application quits.	None
2.04	Register	None	Clicking the Register button should take the user to the *Register* form	The user is taken to the *Register* form.	None
Register					
3.00	Username	Username: *Jimmy*	Entering a valid username fills the progress bar	The progress bar is filled	None
3.01	Username failure	Username: *root*	Entered an already taken username should not fill the progress bar	The progress bar is filled	A check for other users before validating the field. Pg X.
3.02	Password	Password: *Password1@*	The progress bar fills as password is valid	The progress bar is filled fully	None
3.03	Password failure	Password: *password5*	The progress bar fills halfway	The progress bar fills halfway	None
3.04	Password confirm	Password: *Password1@* Password Confirm: *Password1@*	The progress bar fills fully	The progress bar fills fully	None

3.05	Password confirm failure	Password: *Password1@* Password Confirm: *password*	The progress bar should not fill up	The progress bar does not full up	None
3.06	Email	Email: *test@test.com*	The progress bar fills up	The progress bar fills up completely	None
3.07	Email failure	Email: *testcom*	The progress bar doesn't fill up	The progress bar doesn't fill up	None
3.08	DOB	DOB: *24/7/06*	The progress bar should fill up fully	The progress bar fills up fully	None
3.09	DOB Failure	DOB: *24/7/1900*	The progress bar should not fill up	The progress bar does not fill up	None
3.10	DOB Failure	DOB: *24/7*	The progress should not fill up	The progress bar does not fill up	None
3.11	Register	Username: *Gerald* Password: *Password1@* PasswordConfirm: *Password1@* Email: *test@test.com* DOB: 14/08/1995	All progress bars should fill up and the Register button should be enabled. When clicked the user will be taken back to the *Login* form.	All progress bars fill up but the register button does not become enabled.	Add Password_COnfirm to respective dictionary Pg X.
3.11	Register failure	Username: *Gerald* Password: *Password1* PasswordConfirm: *Password1* Email: *test@test.com* DOB: 14/08/1995	The register button should not be enabled.	The register button is not enabled.	None

3.12	Back	None	After clicking the button, the user will be returned to the *Login* form	The user is returned to the *Login* form.	None
Main Menu					
4.00	Logout	None	The logout button should return the user back to the *Login* form.	The user is returned to the *Login* form.	None
4.01	Play	None	The play button should take the user to the *Choose Quiz* form	The user is taken to the *Choose Quiz* form.	None
4.02	Leaderboard	None	The leaderboard button should return the user back to the Le*aderboard* form	The user is taken to the *Leaderboard* form	None
4.03	Admin Settings View	Normal User	This button should not appear	The button does not appear	None
4.04	Admin Settings View	Admin User	This button should appear	The button does appear	None
4.05	Chess board game	None	The chess board should play a famous game in the menu	The chess board plays a famous game.	None
4.06	Profile Picture	None	The user should be able to access their settings through the profile picture icon	The user is able to access their settings through the profile picture icon	None
Choose Quiz					
5.00	Mixed	None	The user should be able	The user is able to	None

			to select the mixed questions.	select the mixed questions	
5.01	Puzzles	None	The user should be able to select the puzzle questions	The user is able to select the puzzle questions	None
5.02	Written Questions	None	The user should be able to select the written questions	The user is able to select the written questions	None
5.03	Main Menu	None	The user should be able to return to the main menu	The user is able to return to the main menu by clicking the appropriate button	None
Puzzle Question					
6.00	View puzzle	None	The user should be able to view the chess board with the correct puzzle.	The user is able to view the current puzzle / board.	None
6.01	Right Answer	Top Left Answer	The next question should be displayed. The User's elo, and other stats should be updated.	The next puzzle is displayed, and their stats are updated.	None
6.02	Wrong Answer	Top Right Answer	The next question should be displayed, and their stats updated.	The next question is displayed and their stats are updated.	None
Written Question					
7.00	Question Display	None	The user should be able	The user is able to view	None

			to view the question title	the question title	
7.01	Question Number Display	None	The right question number should be displayed at the top right	The correct question number is displayed.	None
7.02	Right Answer	Top Left Answer	The next question should be displayed and the user's stats should be updated.	The next question is displayed and the user's stats are updated.	None
7.03	Wrong Answer	Top Right Answer	The next question should be displayed and the user's stats should be updated	The next question is displayed and the user's stats are updated.	None
Quiz Result					
8.00	Questions correct view	None	The user should be able to view how many questions they got right	The user is able to view how many questions they got right	None
8.01	ELO change view	None	The user should be able to view their change in ELO from before to after the questions	The user is able to view their change in elo.	None
8.02	Back	None	This button should take the user back to the *Main Menu*	The user is taken back to the *Main Menu*	None
8.03	Leaderboard	None	This button should take the user to the *leaderboard*	The user is taken to the *Leaderboard*	None
Leaderboard					

9.00	Main Menu	None	The user should be redirected to the *Main Menu*	The user is able to return to the *Main Menu*	None
9.01	Username	None	The usernames for all users should be displayed in their proper order	The usernames for all users on the leaderboard are displayed	None
9.02	ELO	None	The correct ELO rating for each user on the leaderboard should be displayed correctly	The correct ELO rating for each user is displayed correctly	None
9.03	Accuracy	None	The correct Accuracy rating for each user on the leaderboard should be displayed correctly	The correct Accuracy rating is displayed for each user.	None
9.04	High Score	None	The correct high score value for each user should be displayed correctly	The correct high score for each user is displayed correctly.	None
9.05	Ranking	ELO selected	The users should be sorted in descending order based on their ELO	The users are sorted in the correct order based on their ELO	None
9.06	Ranking	Accuracy Selected	The users should be sorted in descending order based on their Accuracy	The users are sorted in the correct order based on their Accuracy	None
9.07	Ranking	High score selected	The users should be	The users are sorted in	None

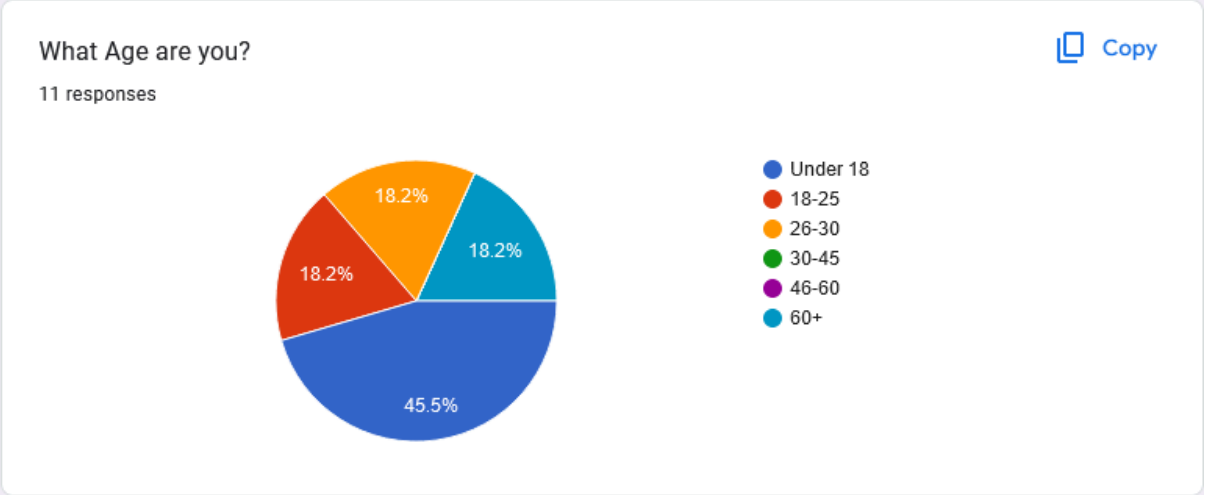
			sorted in descending order based on their High Score	the correct order based on their High Score	
Admin Settings					
10.00	View Puzzles	None	The user should be redirected to the *View All Puzzles* form	The user is redirected to the appropriate form	None
10.01	View Questions	None	The user should be redirected to the *View All Questions* form	The user is redirected to the appropriate form	None
10.02	Create Puzzle	None	The user should be redirected to the *CreatePuzzle* form	The user is redirected to the appropriate form	None
10.03	Create Question	None	The us er should be redirected to the *Create Question form*	The user is redirected to the appropriate form	None
10.04	Password Strength	Selecting either Strong, Medium or Weak	The user should be able to select one of the password strength options	The application crashes after selecting an option	Change options from checkboxes to buttons Pg X.
10.05	User promotion	Username: *jimmy*	The admin should be able to input a user's name and it should autocomplete	The admin is able to input a user's name and it autocomplet es	None
10.06	User promotion	Username: *groot*	The admin should be able to enter the name but no autocomplete	The admin is able to enter the name but there is no autocomplet	None

				e	
10.07	User Promotion	Username: *jimmy*	The textbox should be cleared and the user will be promoted to an admin.	The textbox is cleared and the user is promoted	None
10.08	User Promotion	Username: *groot*	A message should be shown to the admin to tell them that the user doesn't exist	No message is shown to the admin	Show a label to tell the admin that the user does not exist. Pg X.
10.09	User Promotion	Username: *(currentUser)*	A message should be shown to tell the admin that they cannot demote / promote themself	No message is shown to the admin	Show a label to tell the admin that they cannot demote / promote themself. Pg X.
10.10	Main Menu	None	The user should be redirected to the *Main Menu*	The user is redirected to the *Main Menu*	None
User Profile					
11.00	Main Menu	None	The user should be redirected to the *Main Menu*	The user is redirected to the *Main Menu*	None
11.01	Profile Picture	None	The user's current profile picture should be displayed	The user's profile picture is displayed	None
11.02	Profile Picture	Next button	When pressed, the user's profile picture should be changed to the next one.	The user's profile picture is changed to the next in sequence	None
11.03	User Details	None	The user's stat such as ELO and Accuracy should be displayed	The user's stats are displayed	None

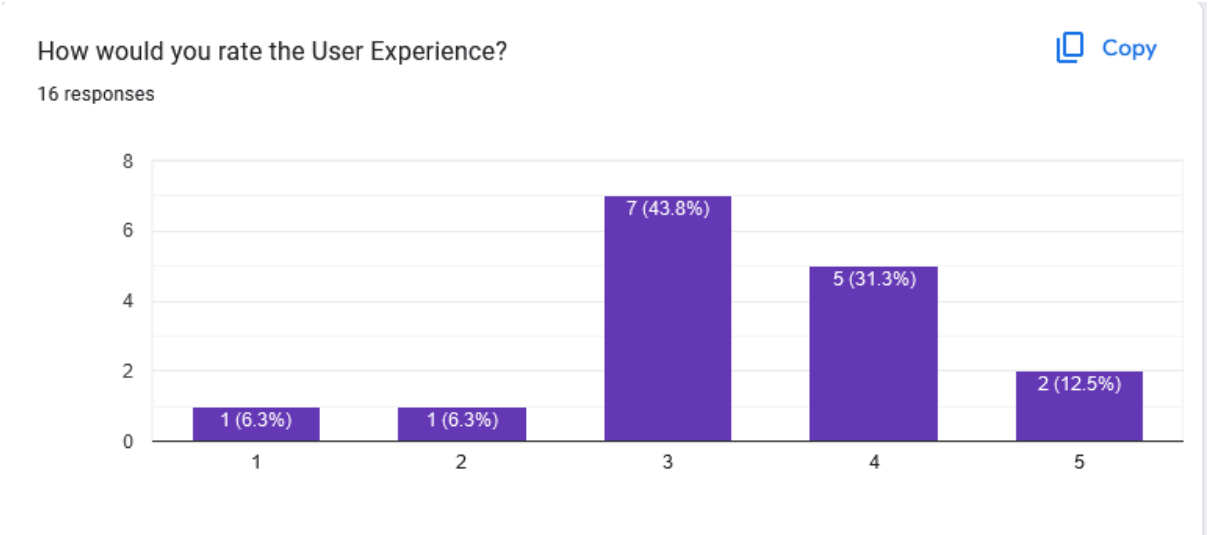
User Acceptance Testing

For the user acceptance testing in my application, I used a google form to collect user feedback. This questionnaire was sent to all user's who had used the application before. It asked them a variety of questions on their preferences and feelings about the program.

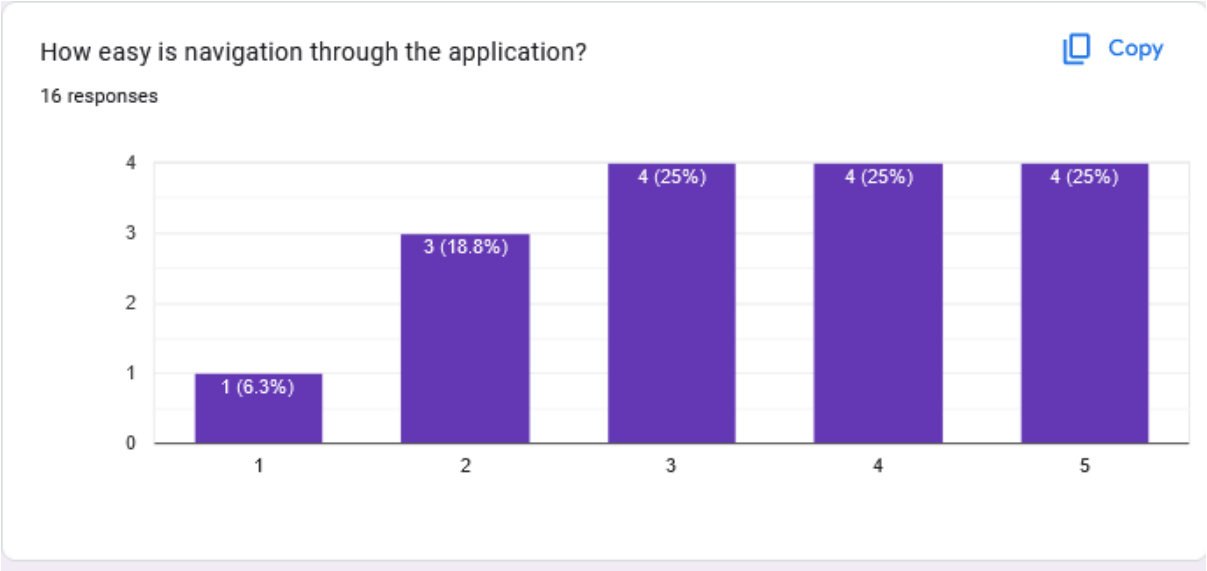
The first thing that the quiz asked was the user's age group. This would allow me to view how each age range interacted with the quiz and could potentially show any weaknesses in the application due to age. For example, the menu system may be intuitive for the younger age ranges, but not so for the older ones.



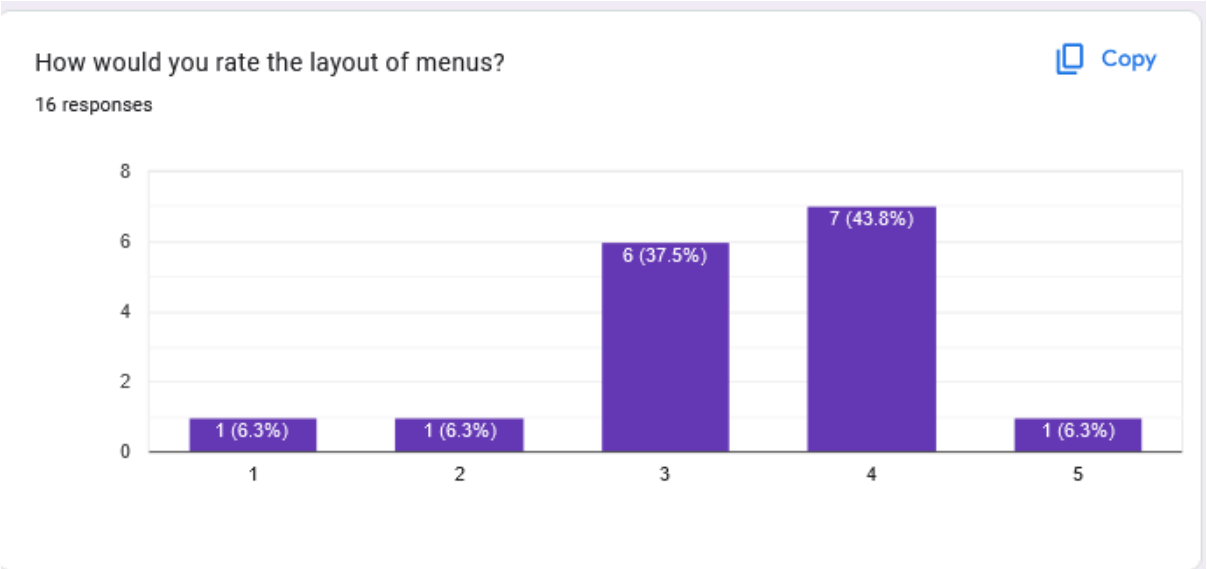
As expected, almost half of all users were under 18. This could be due to the recent surge of growth in young people playing chess, or simply because more young people had access to the application. However, I found that there were also other ranges of age groups using the application, even some over 60 years old. This data impacted my design process as I decided to enlarge most of the buttons as some of the users may find it difficult to properly view the small buttons.



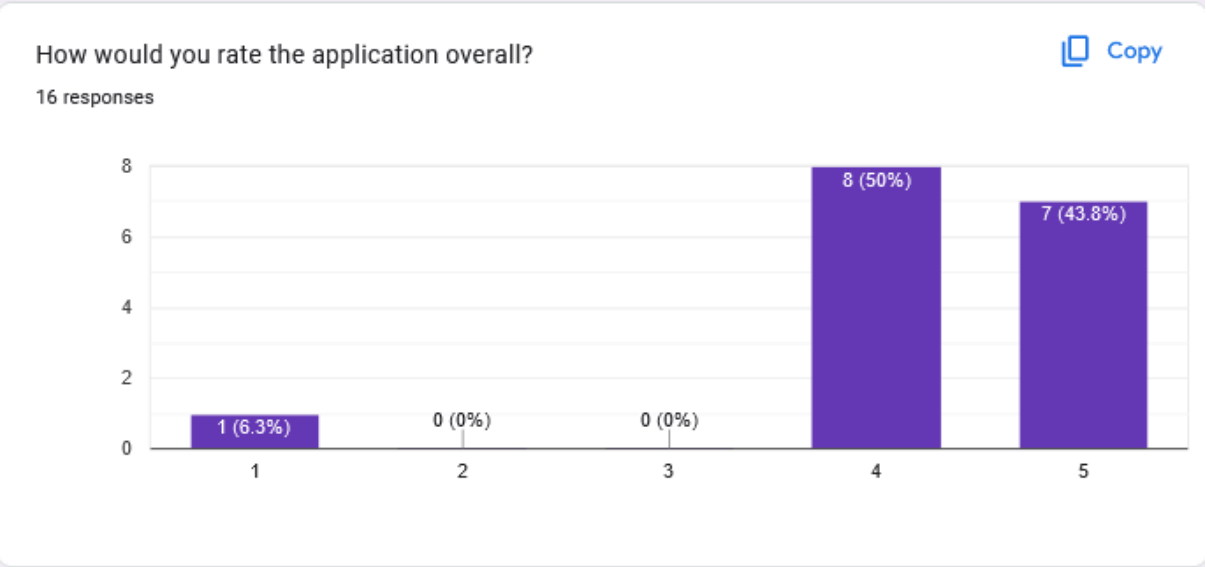
I also asked the user about how they would rate the user experience, with 1 being ‘Extremely Poor’ and 5 being ‘Excellent’. Most users answered in the middle, which further informed my design decisions.



In addition to the user experience, I also prompted the user for their opinion on navigation through the application. I believe that navigation is an extremely important aspect of the application, and I focused largely on this aspect. I am happy with the generally high answers.



The layout of menus was generally well received, as I believe that they were easy to navigate, which was also indicated by the feedback in the navigation portion of the feedback. I made sure that the menus in the application were intuitive to navigate.



The overall ratings of my application were positive, which is indicative of the quality of user experience. I believe that my application was successful in its goals of being user friendly, and having a good user experience while using the application.

Implementation / Code Dump

Program.cs

```
namespace ChessMasterQuiz;

internal static class Program
{
      /// <summary>
      ///  The main entry point for the application.
      /// </summary>
      [STAThread]
      public static void Main()
      {
      ApplicationConfiguration.Initialize();

      Application.Run(new formMain());
      }
}
```

GlobalUsings.cs

```
global using Chess;
global using static Chess.Colour;
global using static Chess.Notation;
global using static Chess.PieceType;
global using Chess.BoardRepresentation;
global using static ChessMasterQuiz.Chess.ChessPieces;

global using ChessMasterQuiz.Misc;
global using static ChessMasterQuiz.Misc.ContextTagType;
global using static
ChessMasterQuiz.Misc.PasswordRequirementLevel;

global using UserRepresentation;
global using static UserRepresentation.UserHelper;
```

```
global using static ChessMasterQuiz.Helpers.ControlHelper;
global using static ChessMasterQuiz.Helpers.Helper;
global using static ChessMasterQuiz.Helpers.FormatDirection;

global using System.Reflection;
global using DCT = ChessMasterQuiz.Misc.DataContextTag;
```

User.cs

```
using System.Net.Mail;
using System.Security.Cryptography;
using System.Text;
using ChessMasterQuiz.Chess;

namespace UserRepresentation;

// This enum is used to identify a regular user from an admin
// This is done through a property in the User class (
User.cs, line 25 )

public enum UserType
{
     USER,
     ADMIN
}

// This is a record type for a password
// A record type is vastly similar to a class, but does most
of the boring work for you.
// The values in the "()" are automatically properties of the
record type.
// In addition to a class, records are reference types.
[Serializable]
public record Password(string Hashed, byte[] Salt);

// This class handles representation of the User model.
[Serializable]
```

```
public class User
{
    // `Name` is the username of the user. Note: Not Unique
    public string Name { get; set; } = string.Empty;

    // A `Password` record is used to store not only the
hashed password, but also the salt
    public Password? Password { get; set; } = null;

    // This `Username` represents a unique username to
identify the user
    public string? Username { get; set; } = null;

    // Gender is a string for inclusivity
    public string Gender { get; set; } = string.Empty;

    // The Date of birth of the user
    public DateTime DOB { get; set; }
    // The email of the representative user
    public MailAddress? Email { get; set; }

    // The `UserType` can either be Admin or User
    public UserType Type { get; set; } = UserType.USER;

    // The current rating of the user
    public ELO Elo { get; set; } = new();

    // A percentage representing the users accuracy
    public float Accuracy { get; set; } = 100;

    // Represents the most questions answered in a single
round for a particular user
    public int HighScore { get; set; }

    // Represents the number of questions in a row without a
loss
    public int CorrectAnswersInRow { get; set; } = 0;

    // The amount of quizzes the user has completed
```

```
     public int QuizesCompleted { get; set; }

     // Reference to the image found in
"Forms/ProfilePictures.resx"
     public int ImageIndex { get; set; } = 0;

     // Bool value to validate who is logged in
     public bool IsLoggedIn { get; private set; }

     public static IReadOnlyList<Image> ProfilePictures {
get; }

     public User()
     { }

     static User()
     {
     ProfilePictures = new List<Image>()
     {
          ChessPieces.WhitePawn,
          ChessPieces.WhiteKnight,
          ChessPieces.WhiteBishop,
          ChessPieces.WhiteRook,
          ChessPieces.WhiteQueen,
          ChessPieces.WhiteKing,
          ChessPieces.BlackPawn,
          ChessPieces.BlackKnight,
          ChessPieces.BlackBishop,
          ChessPieces.BlackRook,
          ChessPieces.BlackQueen,
          ChessPieces.BlackKing
     };
     }

     // Constructor for the User class takes a plain text
password and stores it as a hash
     public User(string name, string password, bool isAdmin =
false)
     {
```

```
        Username = name;

        // Generate a random `salt` which is needed for the
hashing algorithm to make it more secure
        var salt = RandomNumberGenerator.GetBytes(64);

        // Convert the string `password` to a byte array. This
is what the hashing algorithm needs
        byte[] passwordInBytes =
Encoding.UTF8.GetBytes(password);

        // This snippet hashes the `passwordInBytes` with the
`Pbkdf2` hashing algo
        // Then converts the has to Base64, which is useful to
put in text
        // The salt is also used to hash the password
        var hashedPassword =

Rfc2898DeriveBytes.Pbkdf2(passwordInBytes,
                          salt,
                          100_000,
                          HashAlgorithmName.SHA512,
                          64);

        // Creates the password object with the hashedPassword
and the salt.
        Password = new(
                    Encoding.UTF8.GetString(hashedPassword),
                    salt
                );

        // The type is inferred from the bool isAdmin
        Type = isAdmin ? UserType.ADMIN : UserType.USER;
        }


        // This method double checks that all users are logged
out before logging in as a specific user.
        public void Login()
```

```
     {
     foreach (var user in Users)
     {
          user.Logout();
     }
     IsLoggedIn = true;
     ActiveUser = this;
     }

     // Logs the user out by simply changing their
`IsLoggedIn` property.
     public void Logout()
     {
     IsLoggedIn = false;
     ActiveUser = null;
     }
}
```

UserHelper.cs

```
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;

namespace UserRepresentation;

public static class UserHelper
{
     static readonly string _userpath = "users.json";
     public static List<User> Users => ReadUsers();
     public static List<User> ReadUsers()
```

```
     {
     string input = File.ReadAllText(_userpath);

     List<User>? users =
JsonSerializer.Deserialize<List<User>>(input);

     if (users is null)
     {
          throw new Exception($"Could not serialize the List
of users correctly :(");
     }

     return users;
     }

     public static void UpdateUser(User user)
     {
     if(user is null)
     {
          return;
     }

     List<User> users = Users;

     if(users.Any(x => x.Username == user.Username)) {
          users.RemoveAll(x => x.Username == user.Username);
     }

     users.Add(user);

     WriteUsers(users);
     }

     public static void WriteUsers()
     {
     UpdateUser(ActiveUser!);
     }

     public static void WriteUser(User user)
```

```
    {
    UpdateUser(user);
    }

    public static void WriteUsers(List<User>? u)
    {
    string json = JsonSerializer.Serialize(u);

    File.WriteAllText(_userpath, json);
    }

    public static User? ActiveUser
    {
    get;
    set;
    }

    public static Password Hash(this string pass)
    {
    // Generate a random `salt` which is needed for the
hashing algorithm to make it more secure
    var salt = RandomNumberGenerator.GetBytes(64);

    // Convert the string `password` to a byte array. This
is what the hashing algorithm needs
    byte[] passwordInBytes = Encoding.UTF8.GetBytes(pass);

    // This snippet hashes the `passwordInBytes` with the
`Pbkdf2` hashing algo
    // Then converts the has to Base64, which is useful to
put in text
    // The salt is also used to hash the password
    var hashedPassword =

Rfc2898DeriveBytes.Pbkdf2(passwordInBytes,
                  salt,
                  100_000,
                  HashAlgorithmName.SHA512,
                  64);
```

```
     // Creates the password object with the hashedPassword
and the salt.
     return new(
               Encoding.UTF8.GetString(hashedPassword),
               salt
          );

     }

     public static string SerializePassword(this Password
password)
     {
     StringBuilder sb = new();


sb.Append($"{nameof(password.Hashed)}{{{password.Hashed}}}");


sb.Append($"{nameof(password.Salt)}{{{Encoding.UTF8.GetString
(password.Salt)}}}");

     return sb.ToString();
     }


     public static User TestUser => new("root", "root",
false)
     {
     Name = "Root User",
     Gender = "N/A",
     };

     public static bool VerifyPassword(this string password,
User user)
     {
     byte[] passwordInBytes =
Encoding.UTF8.GetBytes(password);
```

```
     var hashedPassword =

Rfc2898DeriveBytes.Pbkdf2(passwordInBytes,
                    user.Password!.Salt,
                    100_000,
                    HashAlgorithmName.SHA512,
                    64);

     if (Encoding.UTF8.GetString(hashedPassword) ==
user.Password.Hashed)
     {
          return true;
     }
     return false;
     }

     public static void Add(this User user)
     {
     Users.Add(user);
     }
}
```

Rating.cs

```
/*

   This Rating class represents an integer value, which is a
```

```
relative to other players and questions.

   The rating system is based off the famous elo system
created by Arpad Elo

 */

namespace UserRepresentation;

// Primary constructor is used to create a Rating prop
public record class ELO
{
     public int Rating { get; set; }

     // Represents either a win or a loss
     public enum MatchupResult
     {
     LOSS = 0,
     WIN = 1
     }

     // History of the ratings of the user
     public List<int> PastRatings { get; set; } = new();

     // Some readonly fields to be used for values in ELO
calculations
     // NOTE: All values are subject to change
     private readonly static int s_initialRating = 1000;
     private readonly static int s_eloModifier = 32;
     private readonly static int s_expectedOutimeCModifier =
400;
     private readonly static int s_minimumPossibleElo = 100;
     private readonly static int s_bonusScore = 10;

     // Blank constructor to allow for the default initial
rating
     public ELO()
     {
     Rating = s_initialRating;
```

```
    }

    public static void Match(ELO elo, MatchupResult result,
int opponentRating)
    {
    // ELO FORMULA
    // R'a = Ra + K * (Sa - Ea) + Sa * V
    // WHERE Ea = Qa / ( Qa + Qb )

    // These are the two ratings before the matchup.
    // They are used to calculate the expected outcome.
    int Ra = elo.Rating;
    int Rb = opponentRating;

    // The standard modifier of the elo.
    // This value is the maximum elo that can be won or lost
    int K = s_eloModifier;

    // The actual outcome of the match
    // 1 for win, 0 for loss
    // Third branch will never be hit
    int Sa = result switch
    {
        MatchupResult.WIN => 1,
        MatchupResult.LOSS => 0,
        _ => default
    };

    // Qa = 10 ^ (Ra / c)
    // Qb = 10^(Rb / c)
    float Qa = (int)Math.Pow(10, Ra /
s_expectedOutimeCModifier);
    float Qb = (int)Math.Pow(10, Rb /
s_expectedOutimeCModifier);

    // Calculate the expected outcome using the appropriate
formula
    float Ea = Qa / (Qa + Qb);
```

```
     // Add a constant scaling value but only if a correct
answer
     int addedBonus = Sa * s_bonusScore;

     // Get the result rating from the modified elo formula
     float resultantRating = Ra + K * (Sa - Ea) + addedBonus;

     // Make sure the elo isnt too low
     if (resultantRating < s_minimumPossibleElo)
     {
          return;
     }

     // Add the rating to the list of previous ratings.
     // Used to show progression
     elo.PastRatings.Add(elo.Rating);

     // Finally, update the rating!
     elo.Rating = (int)resultantRating;

     UpdateUser(ActiveUser!);
     }
}
```

AdminConfiguration.cs

```
using Newtonsoft.Json;

namespace ChessMasterQuiz.Misc;

public class AdminConfiguration
{
```

```
    public PasswordRequirementLevel PasswordRequirementLevel
{ get; set; }
    private const string Path = $"AdminConfig.json";
    public static AdminConfiguration? Read()
    {
    string json = File.ReadAllText(Path);
    AdminConfiguration? config =
JsonConvert.DeserializeObject<AdminConfiguration>(json);

    if (config is null) return default;

    return config;
    }

    public static void Write(AdminConfiguration config)
    {
    File.WriteAllText(Path,
JsonConvert.SerializeObject(config));
    }

    public void Write()
    {
    Write(this);
    }
}
```

IContext.cs

```
namespace ChessMasterQuiz.Misc;

// These represent any data that could be passed through
public enum ContextTagType
{
```

```
    QUESTION,
    QUESTIONS_CORRECT,
    INDEX,
    NUMBER,
    ACTION,
    EMAIL,
    USER,
    PUZZLE
}

/// <summary>
/// Record type for Data-Context Tag Pair
/// </summary>
/// <param name="data">Object of data</param>
/// <param name="tag">Tag to identify the data</param>
public record DataContextTag(object data, ContextTagType
tag);

/*
   The IContext interface allows the Context System
   to ensure that the method is available
 */
interface IContext
{
    // DCT is an alias for DataContextTag
    public void UseContext(IEnumerable<DCT> context);
}
```

Question.cs

```
using static ChessMasterQuiz.Misc.QuestionType;

namespace ChessMasterQuiz.Misc;
```

```
public record Answer(List<string> Answers, uint Index);

public enum QuestionType
{
     TEXT,
}

public enum Difficulty
{
     EASY,
     MEDIUM,
     HARD
}

public abstract class Question : IEquatable<Question>,
IComparable<Question>
{
     public QuestionType Type { get; init; }
     public string Name { get; init; } = string.Empty;
     public Difficulty Difficulty { get; init; }
     public virtual int Rating { get; init; } = -1;

     public int CompareTo(Question? other)
     {
     return other!.Difficulty.CompareTo((int)Difficulty);
     }
     public bool Equals(Question? other)
     {
     return other!.Name == Name;
     }
}
public class TextQuestion : Question
{
     public TextQuestion()
     {
     Type = TEXT;
     if (Rating != -1)
     {
          Rating = RatingFromDifficulty(Difficulty);
```

```
    }
    }

    public TextQuestion(string q, Answer a) : this()
    {
    Q = q;
    A = a;
    }
    public string Q { get; set; } = string.Empty;
    public Answer? A { get; set; } = null;
}
```

Serializer.cs

```
// -- Serializer.cs --
//
// This file contains classes which deal with the
Serialization or Deserialization of QON
// QON stands for ( Question Object Notation ) and is used to
store question details on disk

using System.Text;
using static System.Char;
using static System.String;
using static ChessMasterQuiz.Misc.TokenType;


// The QonConvert class and its associated classes ( Lexer,
Parser, Token )
// should be kept in a different namespace to keep things
tidy
namespace ChessMasterQuiz.Misc;

/// <summary>
/// Qon ( Question Object Notation ) Converter.
```

```
/// </summary>

// This static class contains methods which deal with
Serializing and Deserializing objects and strings
respectively.
// The QonConvert class is able to Serialise both Questions
and Users
// It uses overloading so only one method call is needed
public static class QonConvert
{
     // Serialize a question into a Lon String
     public static string Serialize(Question question)
     {
     // StringBuilder is used as it is much faster than
traditional string concat.
     // This becomes necessary when working with large
strings ( 250+ concat operations )
     StringBuilder stringBuilder = new();

     // Improvement to be made:
     // Use reflection to get types at runtime, loop through
them and append to stringBuilder
     stringBuilder.Append($"Type:\"{(int)question.Type}\",");
     stringBuilder.Append($"Name:\"{question.Name}\",");
     stringBuilder.Append($"Rating:\"{question.Rating}\",");

     // If the question is a TextQuestion:
     // Loop through potential answers and append them to
stringBuilder
     // Also append the Index and Q fields to the
stringBuilder

     // Improvement to be made:
     // When multiple different types of questions are
implemented
     // it would be time consuming to manually do this.
     // Once again use reflection
     if (question is TextQuestion)
     {
```

```
            stringBuilder.Append("{");
            TextQuestion tq = (question as TextQuestion)!;
            stringBuilder.Append($"Q:\"{tq.Q}\",");
            for (int i = 0; i < tq.A!.Answers.Count; i++)
            {
                  stringBuilder.Append($"Answer{i +
1}:\"{tq.A.Answers[i]}\",");
            }
            stringBuilder.Append($"Index:\"{tq.A.Index}\"");
            stringBuilder.Append("}");
      }

      // Return the final build string
      return stringBuilder.ToString();
      }

      // This method deals with serialising a list of objects
      public static string Serialize<T>(this List<T> objects)
where T : Question
      {
      StringBuilder stringBuilder = new();

      foreach (var obj in objects)
      {
            stringBuilder.Append(Serialize(obj));
            stringBuilder.Append("\n");
      }

      return stringBuilder.ToString();
      }

      public static List<Question> Deserialize(params string[]
strs)
      {
      List<Question> questions = new();
      foreach (var str in strs)
      {
            if (IsNullOrWhiteSpace(str))
            {
```

```
                continue;
            }

            List<Token> tokens = Lexer.Lex(str);
            Question question = Parser.Parse(tokens);
            questions.Add(question);
        }
        return questions;
        }
}

// This enum represents the different `Tokens` that are found
in the Lon strings
public enum TokenType
{
        COLON,
        LEFT_BRACKET,
        RIGHT_BRACKET,
        IDENTIFIER,
        DATA
}

// Class representing the actual Token value
file class Token
{
        // The specific type of `Token`
        public TokenType Type { get; init; }
        // The data that is attached to the `Type`
        // This is only applicable to `IDENTIFIER` and `DATA`
        // Nullable as some `Types` do not have associated data
        public object? Data { get; init; }

        // This method allows for deconstruction of a Token
        // Deconstruction is the breaking down of complex types
into simpler ones
        // Example:
        // ( token, data ) = TokenName;
        public void Deconstruct(out TokenType type, out object?
data)
```

```
    {
    type = Type;
    data = Data;
    }

    // Simple constructor for the `Token` type
    public Token(TokenType type, object data)
    {
    Type = type;
    Data = data;
    }

    // This constructor can be used if no data is present.
    // However, an exception will be thrown if either a
`DATA` or an `IDENTIFER` passed
    // DON'T USE EXCEPTIONS!
    public Token(TokenType type)
    {
    if (type == DATA || type == IDENTIFIER)
    {
        throw new Exception();
    }

    Type = type;
    Data = null;
    }
}

// File scoped namespace
// This class deals with Lexical Analysis

// Lexical Analysis is taking a string of text
// and converting it into a string of tokens.
//
// Example string: Name:"test"
// Example tokens: IDENTIFIER - COLON - DATA

// This is a common technique in building programming
languages.
```

```
file static class Lexer
{
    // This is a pointer to the current char in the string
    // ( The char to be evaluated )
    static int _current;
    // The list of `Tokens` which will be returned to the
consumer
    static List<Token> _tokens = new();

    // These methods deal with navigating the string

    // An example of why they are necessary is how
programming languages deal with the `=` sign
    // When a programming language interpreter sees a `=`,
    // it doesn't know if it is an assignment operator, or a
comparasin operator
    // therefor it needs to see the next token before
evaluating itself.
    // The same concept is used here

    // Next is used to iterate to the next char (
represented by `_current` )
    static void Next(int step = 1)
    {
    _current += step;
    }

    // The `Peek` method is used to look at the next char in
the str
    // w/o having to increment the _current variable
    static char Peek(ReadOnlySpan<char> str)
    {
    return str[_current + 1];
    }

    // The `Current` method is ued to look at the current
char in the str
    // Perhaps the most important helper method here
    static char Current(ReadOnlySpan<char> str)
```

```
    {
    return str[_current];
    }

    // The `Consume` method adds the `token` to the list,
    // and moves onto the next char to be evaluated
    static void Consume(Token token)
    {
    _tokens.Add(token);
    _current++;
    }

    // This Lex method contains the actual loop of
evaluation
    // It takes in a string `input` which is the list of
chars
    public static List<Token> Lex(string input)
    {
    // Due to speed concerns with many evaluations,
    // it improves performance by first converting the
    // string to a `ReadOnlySpan`, which is must faster
    // due to it's lack of writing ability.
    ReadOnlySpan<char> str = input.AsSpan();

    // Becuase the method is static,
    // the fields should be reset to default
    _current = 0;
    _tokens = new();

    // This loop does not contain any initialization or
incrementor
    // This is because it is handled by the helper methods
mentioned previously
    for (; _current < str.Length;)
    {
        // Most of the tokens are single character,
        // so no look ahead is needed.
        switch (Current(str))
        {
```

```
                case ':':
                    Consume(new Token(COLON));
                    break;
                case ',':
                    Next();
                    break;
                case '{':
                    Consume(new Token(LEFT_BRACKET));
                    break;
                case '}':
                    Consume(new Token(RIGHT_BRACKET));
                    break;
                case '"':
                    {

                    // If there is a ("), then the token
should be of type `DATA`
                    // The `Next()` is needed to proceed to
the first character in the string
                    Next();

                    // StringBuilder is used for performance
                    StringBuilder sb = new();

                    // Loop through chars, until the second
(")
                    // While in the string, append the
current char to the StringBuilder
                    while (Current(str) != '"')
                    {
                        sb.Append(Current(str));
                        // Jump to next char in the string
                        Next();
                    }

                    // Finally consume as type `DATA`, with
the final string
                    Consume(new Token(DATA, sb.ToString()));
```

```
                    break;
                    }

               default:
                    {
                    // Very similar to the `DATA` above

                    // Due to the limitations of variable
names,
                    // No (") are necessary, just ensure all
chars are letters
                    StringBuilder sb = new();
                    while (IsLetterOrDigit(Current(str)))
                    {
                         sb.Append(Current(str));
                         Next();
                    }

                    // Finally consume as type `IDENTIFIER`
                    Consume(new Token(IDENTIFIER,
sb.ToString()));
                    break;
                    }
          }
     }

     // Return the final list of `Tokens`
     return _tokens;

     }
}

// File scoped namespace
// This static class deals with Parsing Tokens

// The `Parser` class takes a List of Tokens
// ( From the Lexical Analysis )
// And converts them into useable C# objects.
file static class Parser
```

```
{
     static readonly List<Type> ValidTypes = new()
     {
     typeof(Question),
     typeof(TextQuestion),
     typeof(Answer)
     };

     static int _current;
     static List<Token> _tokens = new();

     static void Next(int step = 1)
     {
     _current += step;
     }

     static Token Peek()
     {
     return _tokens[_current < _tokens.Count ? _current + 1 :
_current];
     }

     static Token Current()
     {
     return _tokens[_current];
     }

     static Token Previous()
     {
     return _tokens[_current > 0 ? _current - 1 : _current];
     }

     public static Question Parse(List<Token> tokens)
     {

     _current = 0;
     _tokens = tokens;

     List<string> answers = new();
```

```
int index = 0;

TextQuestion question = new();

for (; _current < _tokens.Count;)
{
     switch (Current().Type)
     {
          case LEFT_BRACKET:
               break;

          case IDENTIFIER:
               {
               var name = Current().Data as string;


               PropertyInfo? property = null;

               void GetProps(Type type)
               {
                    if (!ValidTypes.Contains(type))
                    {
                         return;
                    }

                    var props = type.GetProperties();

                    property = props.FirstOrDefault(
                         x => x.Name == name);

                    if (property is not null)
                    {
                         return;
                    }

                    foreach (var prop in props)
                    {
                         GetProps(prop.PropertyType);
                    }
```

```
                    }

                    if (name!.ToLower().Contains("answer"))
                    {
                         Next();

answers.Add((string)Current().Data!);
                         break;
                    }
                    else if
(name.ToLower().Contains("index"))
                    {
                         Next();
                         index =
int.Parse((string)Current().Data!);
                         break;
                    }
                    else
                    {
                         GetProps(question.GetType());
                    }

                    if (property is null)
                    {
                         break;
                    }



                    // If null, then identifier must be in
the Answers
                    if (property is null)
                    {
                         // Get type of the question (
TextQuestion etc... )
                         // Look up properties recursively

                         if (question.Type ==
QuestionType.TEXT)
```

```
                    {
                        property =
CheckForProperties(typeof(Answer), name);
                    }

                    if (property is null)
                    {
                        break;
                    }

                }

                Next();

                // If the property is an enum, then a
explicit conversion from stirng -> enum Type is needed
                if (property!.PropertyType.IsEnum)
                {
                    Enum.TryParse(property.PropertyType,
(string)Current().Data!, out var output);
                    property.SetValue(question, output);
                }
                else
                {
                    property.SetValue(question,
Convert.ChangeType(Current().Data, property.PropertyType));
                }


                // If its not a
                // else
                // {
                // property.SetValue(question,
Current().Data);
                // }
```

```
                    break;
                    }
          }
          Next();
     }

     // question.A = new(null,);
     question.A = new(answers, (uint)index);
     return question;
     }


     private static PropertyInfo? CheckForProperties(Type
type, string name)
     {
     var props = type.GetProperties();

     var prop = props.FirstOrDefault(
          x => x.Name == name);

     return prop;
     }



}
```

Validator.cs

```
using System.Net.Mail;
using System.Text.RegularExpressions;
using static ChessMasterQuiz.Misc.RequirementType;
using static ChessMasterQuiz.Misc.ValidationType;
```

```
namespace ChessMasterQuiz.Misc;

enum ValidationType
{
     EMAIL,
     USERNAME,
     PASSWORD,
     PASSWORD_CONFIRM,
     DISPLAY,
     DOB,
     GENDER
}

enum RequirementType
{
     LENGTH,
     SPECIALCHARACTERS,
     UPPERCASE,
     NUMBER
}

public enum PasswordRequirementLevel
{
     STRONG,
     MEDIUM,
     WEAK
}

internal static class Validator
{
     public static (bool, float) Validate(this string text,
ValidationType type, PasswordRequirementLevel? passwordLevel
= null)
     {
     passwordLevel ??=
GetAdminConfig()?.PasswordRequirementLevel;

     Dictionary<RequirementType, bool> ValidationLookup =
```

```
new()
     {
          { LENGTH, false },
          { SPECIALCHARACTERS, false },
          { UPPERCASE, false },
          { RequirementType.NUMBER, false },

     };

     switch (type)
     {
          case ValidationType.EMAIL:
               return (MailAddress.TryCreate(text, out _),
100);
          case PASSWORD_CONFIRM:
               return (false, 0);
          case DOB:
               return IsValidAge(text);
          case DISPLAY:
               return (text.ValidateDisplayName(), 100);
          case GENDER:
               return (text.ValidateGender(), 100);
          case USERNAME:
               return (text.ValidateUsername(), 100);
          case PASSWORD:
               int minimumLength = passwordLevel switch
               {
                    STRONG => 12,
                    MEDIUM => 8,
                    _ => 4,
               };

               int specialCharacterCount = passwordLevel
switch
               {
                    STRONG => 3,
                    MEDIUM => 1,
                    _ => 0
               };
```

```
                int upperCaseCount = passwordLevel switch
                {
                     STRONG => 1,
                     MEDIUM => 1,
                     _ => 0
                };

                int numberCount = passwordLevel switch
                {
                     STRONG => 2,
                     MEDIUM => 1,
                     _ => 0
                };

                if (text.Length > minimumLength)
                {
                     ValidationLookup[LENGTH] = true;
                }

                if (text.Where(x =>
!char.IsLetterOrDigit(x)).Count() >= specialCharacterCount)
                {
                     ValidationLookup[SPECIALCHARACTERS] =
true;
                }

                if (text.Where(char.IsUpper).Count() >=
upperCaseCount)
                {
                     ValidationLookup[UPPERCASE] = true;
                }

                if (text.Where(char.IsDigit).Count() >=
numberCount)
                {
                     ValidationLookup[RequirementType.NUMBER]
= true;
                }
```

```
                if (ValidationLookup.All(x => x.Value ==
true))
                {
                     return (true, 100);
                }
                return (false, ValidationLookup.Count(x =>
x.Value == true) * 25);
     }
     var regex = new Regex(ValidationTypeRegex[type]);
     return (regex.IsMatch(text), -1);
     }

     private static bool ValidateDisplayName(this string
text)
     {
     if (string.IsNullOrWhiteSpace(text))
     {
          return false;
     }

     if (text.Length < 4)
     {
          return false;
     }

     if (text.Length > 16)
     {
          return false;
     }

     if (text.Any(x => !char.IsLetterOrDigit(x)))
     {
          return false;
     }


     return true;
```

```
    }

    private static bool ValidateUsername(this string text)
    {
    if (string.IsNullOrWhiteSpace(text))
    {
        return false;
    }

    if (text.Length < 4)
    {
        return false;
    }

    if (text.Length > 16)
    {
        return false;
    }

    if (text.Any(x => !char.IsLetterOrDigit(x)))
    {
        return false;
    }
    if (Users.Any(x => x.Username?.ToLower() == text))
return false;

    return true;

    }

    private static bool ValidateGender(this string text)
    {
    if (text.Length > 15)
    {
        return false;
    }

    if (text.Any(x => !char.IsLetter(x)))
    {
```

```
          return false;
     }

     return true;
     }


     public static Dictionary<ValidationType, string>
ValidationTypeRegex = new()
     {
     { ValidationType.EMAIL,
@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%
*?&]{8,}$" },
     { PASSWORD,
@"^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%
*?&]{8,}$" },
     { DISPLAY, @"\b([A-ZÀ-ÿ][-,a-z. ']+[ ]*)+" },
     { GENDER,
@"/^[a-zA-ZàáâäãåąčćęèéêëėįìíîïłńòóôöõøùúûüųūÿýżźñçčšžÀÁÂÄÃÅĄĆ
ČĖĘÈÉÊËÌÍÎÏĮŁŃÒÓÔÖÕØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽ∂ð ,.'-]+$/u" },
     };

     public static (bool, float) IsValidAge(string b)
     {
     if (!DateTime.TryParse(b, out var birth))
          return (false, 0);

     DateTime today = DateTime.Today;
     int age = today.Year - birth.Year;

     if(birth > today.AddYears(-age))
     {
          age--;
     }

     bool isRightAge = age > 14 && age < 120;
     int percentage = 50;

     if (isRightAge) percentage = 100;
```

```
    return (isRightAge, percentage);
    }
}
```

Helper.cs

```
using System.Diagnostics;
using System.Text.Json;
using ChessMasterQuiz.Misc;

namespace ChessMasterQuiz.Helpers;

public static class Helper
{
    public static AdminConfiguration? GetAdminConfig()
    {
    return AdminConfiguration.Read();
    }

    public static Image GetLogo()
    {
    List<Image> images = new()
    {
        GeneralResources.Chess_Logo_1,
        GeneralResources.Chess_Logo_2,
        GeneralResources.Chess_Logo_3,
        GeneralResources.Chess_Logo_4,
        GeneralResources.Chess_Logo_5,
        GeneralResources.Chess_Logo_6,
```

```
            GeneralResources.Chess_Logo_7,
            GeneralResources.Untitled_design,
            GeneralResources.Untitled_design_1_,
            GeneralResources.Untitled_design_2_,
            GeneralResources.Untitled_design_3_,
            GeneralResources.Untitled_design_4_,
            GeneralResources.Untitled_design_5_,
            GeneralResources.Untitled_design_6_,
            GeneralResources.Untitled_design_7_,
            GeneralResources.Untitled_design_8_,
            GeneralResources.Untitled_design_9_,
            GeneralResources.Untitled_design_10_,
            GeneralResources.Untitled_design_11_,
            GeneralResources.Untitled_design_12_,
            GeneralResources.Untitled_design_13_,
            GeneralResources.Untitled_design_14_,
            GeneralResources.Untitled_design_15_,
            GeneralResources.Untitled_design_16_,
            GeneralResources.Untitled_design_17_,
            GeneralResources.Untitled_design_18_,
            GeneralResources.Untitled_design_19_,
            GeneralResources.Untitled_design_20_,
            GeneralResources.Untitled_design_21_,
            GeneralResources.Untitled_design_22_,
            GeneralResources.Untitled_design_23_,
        };

        return images[_random.Next(0, images.Count)];
        }

        private static readonly Random _random = new();
        public static Action EmptyAction => new(() => { });
        public static Action<T> EmptyActionGeneric<T>()
        {
        return new Action<T>((_) => { });
        }
```

```
     public static Action<Ti, Tj> EmptyActionGeneric<Ti,
Tj>()
     {
     return new Action<Ti, Tj>((_, _) => { });
     }

     public static void MainMenu()
     {
     ActivateForm<formMenu>();
     }

     public static IList<T> Shuffle<T>(this Span<T> list)
     {
     int n = list.Length;
     while (n > 1)
     {
          n--;
          int k = _random.Next(n + 1);
          T value = list[k];
          list[k] = list[n];
          list[n] = value;
     }

     return list.ToArray().ToList();
     }

     public static int RatingFromDifficulty(Difficulty
diff)
     {
     int @base = diff switch
     {
          Difficulty.HARD => 1800,
          Difficulty.MEDIUM => 1000,
          Difficulty.EASY => 200,
          _ => 2400
```

```
    };

    return new Random().Next(@base - 400, @base + 400);
    }

    private const string _puzzlePath = "puzzles.json";

    public static void WritePuzzles(this List<Puzzle>
puzzles)
    {
    string json = JsonSerializer.Serialize(puzzles);

    File.WriteAllText(_puzzlePath, json);
    }

    public static List<Puzzle> ReadPuzzles()
    {
    string input = File.ReadAllText(_puzzlePath);

    List<Puzzle>? puzzles =
JsonSerializer.Deserialize<List<Puzzle>>(input);

    if (puzzles is null)
    {
        throw new Exception($"Could not deserialize the
List of Puzzles correctly :(");
    }

    return puzzles;
    }
}
```

ControlHelper.cs

```
namespace ChessMasterQuiz.Helpers;

public enum FormatDirection
{
     X = 1,
     Y = 2
}

public static class ControlHelper
{

     public static bool Has(this IEnumerable<DataContextTag>
context, ContextTagType type)
     {
     return context.Any(x => x.tag == type);
     }

     public static IEnumerable<object> Get(this
IEnumerable<DataContextTag> context, ContextTagType type)
     {
     return context.Where(x => x.tag == type).Select(x =>
x.data);
     }

     private static readonly Dictionary<Type, ContextTagType>
TypeToContextMap = new()
     {
     { typeof(Puzzle), PUZZLE },

     };

     public static T? GetFirst<T>(this IEnumerable<DCT>
context)
     {
     var tag = TypeToContextMap[typeof(T)];
     return GetFirst<T>(context, tag);
     }

     public static T? GetFirst<T>(this
```

```
IEnumerable<DataContextTag> context, ContextTagType type)
     {
     return (T?)context.Where(x => x.tag == type).Select(x =>
x.data).FirstOrDefault();
     }

     public static object? GetFirst(this
IEnumerable<DataContextTag> context, ContextTagType type)
     {
     return context.Where(x => x.tag == type).Select(x =>
x.data).FirstOrDefault();
     }

     public static Control? FindTag(this IEnumerable<Control>
controls, string tag)
     {
     foreach (var control in controls)
     {
          if (control.Tag is null)
          {
               continue;
          }

          if ((string)control.Tag == tag)
          {
               return control;
          }
     }
     return null;
     }

     public static IEnumerable<TextBox> FindPlaceHolder(this
IEnumerable<TextBox> controls, string text)
     {
     foreach (var control in controls)
     {
          if (control.PlaceholderText is null)
          {
               continue;
```

```
            }

            if (control.PlaceholderText.ToLower() == text)
            {
                yield return control;
            }
        }
        }

        public static IEnumerable<DataContextTag> ToDCT(this
IEnumerable<object> objs)
        {
        List<DataContextTag> dcts = new();
        foreach (var o in objs)
        {
            dcts.Add((DataContextTag)o);
        }
        return dcts;
        }

        /// <summary>
        /// Reference to Main Form with correct Typing [
Alternative to "(ActiveForm as formMain)" ]
        /// </summary>
        public static formMain Main => (Form.ActiveForm as
formMain)!;


        public static void ActivateForm<T>() where T : Form,
new()
        {
        // Create the form
        formMain.ChildForm = CreateForm<T>([]);

        // All the boring stuff to display the form
        formMain.ChildForm.TopLevel = false;
        formMain.ChildForm.Dock = DockStyle.Fill;
        formMain.ChildForm.FormBorderStyle =
FormBorderStyle.None;
```

```
     formMain.ChildForm.Enabled = true;
     formMain.ChildForm.Visible = true;

     var holder = formMain.GetPanelHolder!();

     holder.Controls.Clear();
     holder.Controls.Add(formMain.ChildForm);
     holder.Show();
     }

     public static void ActivateForm<T>(params (object,
ContextTagType)[] context) where T : Form, new()
     {
     List<DCT> dct = new();
     foreach (var (obj, tag) in context)
     {
          dct.Add(new(obj, tag));
     }
     // Create the form
     formMain.ChildForm = CreateForm<T>(dct);

     // All the boring stuff to display the form
     formMain.ChildForm.TopLevel = false;
     formMain.ChildForm.Dock = DockStyle.Fill;
     formMain.ChildForm.FormBorderStyle =
FormBorderStyle.None;
     formMain.ChildForm.Enabled = true;
     formMain.ChildForm.Visible = true;

     var holder = formMain.GetPanelHolder!();

     holder.Controls.Clear();
     holder.Controls.Add(formMain.ChildForm);
     holder.Show();
     }

     public static void ActivateFormDCT<T>(params
DataContextTag[] context) where T : Form, new()
     {
```

```
     // Create the form
     formMain.ChildForm = CreateForm<T>(context);

     // All the boring stuff to display the form
     formMain.ChildForm.TopLevel = false;
     formMain.ChildForm.Dock = DockStyle.Fill;
     formMain.ChildForm.FormBorderStyle =
FormBorderStyle.None;
     formMain.ChildForm.Enabled = true;
     formMain.ChildForm.Visible = true;

     var holder = formMain.GetPanelHolder!();

     holder.Controls.Clear();
     holder.Controls.Add(formMain.ChildForm);
     holder.Show();
     }

     /// <summary>
     /// Creates a Form which passes context through if
needed
     /// </summary>
     /// <typeparam name="T">The Type of Form to
Create</typeparam>
     /// <param name="context">Enumerable of
<c>DataContextTag</c>'s to pass</param>
     /// <returns>The Form Created</returns>
     public static Form
CreateForm<T>(IEnumerable<DataContextTag> context) where T :
Form, new()
     {
     // Create the Form
     var instance = Activator.CreateInstance<T>()!;

     // Grab the method FROM THE INTERFACE!!
     var method = typeof(IContext).GetMethod("UseContext");

     // If the method exists | The form impl the interface
     if (method is not null &&
```

```
typeof(IContext).IsAssignableFrom(typeof(T)))
     {
          method!.Invoke(instance, new object[] { context });
     }

     // WinFormsScraper.WinFormsScraper.Scrape(instance);

     // Return the form
     return instance;
     }

     /// <summary>
     /// Returns an Action that will center the Control in
the active form
     /// </summary>
     /// <param name="control">The Control to be
centered</param>
     /// <returns>Action to center</returns>
     public static void Center(this Control control, params
FormatDirection[] directions)
     {
     foreach (var d in directions)
     {
          switch (d)
          {
               case X:
                    control.Location =
new((formMain.ChildForm!.Width / 2) - (int)(0.5 *
control.Width), control.Location.Y);
                    break;
               case Y:
                    control.Location =
new((formMain.ChildForm!.Width / 2) - (int)(0.5 *
control.Width), control.Location.Y);
                    break;
          };

     }
     }
```

```
     public static void Center(this Control control,
FormatDirection direction, float distance)
     {
     switch (direction)
     {
          case X:
               control.Location =
new(((formMain.ChildForm!.Width / 2) - (int)(0.5 *
control.Width)) + (int)distance, control.Location.Y);
               break;
          case Y:
               control.Location =
new(((formMain.ChildForm!.Width / 2) - (int)(0.5 *
control.Width)) + (int)distance, control.Location.Y);
               break;
     };

     }


     public static void RedIfWrong(this TextBox textbox, bool
isRight)
     {
     if (isRight)
     {
          textbox.BackColor = Color.FromArgb(255, 81, 94,
74);
     }
     else
     {
          textbox.BackColor = Color.FromArgb(255, 225, 117,
111);
     }
     }

     public static void CreateEquidistantIntervals<T>(this
List<T> controls, float distance, int index = 0) where T :
Control
```

```
    {
    if (index >= controls.Count() - 1)
    {
        return;
    }

    var bottom = controls[index].Bottom;
    controls[index + 1].Top = bottom + (int)distance;

    controls.CreateEquidistantIntervals(distance, index++);
    }


}
```

FormAddQuestion.cs

```
using System.Data;
using ChessMasterQuiz.Forms.Admin;

namespace ChessMasterQuiz.Forms
{
    public partial class formAddQuestion : Form
    {
    public formAddQuestion() =>
        InitializeComponent();

    private void btnCreate_Click(object sender, EventArgs e)
    {
        List<CheckBox> checkBoxes = new()
        {
            checkBox1,
            checkBox2,
            checkBox3,
```

```
                checkBox4,
            };

            if (checkBoxes.Where(x => x.Checked).Count() != 1)
            {
                return;
            }

            TextQuestion tq = new()
            {
                Q = txtBoxQuestion.Text,
                A = new Answer(new() { txtBoxAnswerOne.Text,
txtBoxAnswerTwo.Text, txtBoxAnswerThree.Text,
txtBoxAnswerFour.Text },
(uint)checkBoxes.IndexOf(checkBoxes.First(x => x.Checked))),
                Rating = int.Parse(txtBoxRating.Text)
            };

            var serialized = QonConvert.Serialize(tq);

            using (StreamWriter sw = new("questions.qon",
true))
            {
                sw.WriteLine(serialized);
            }

            // Clear all text boxes for next question
            foreach (var textbox in Controls.OfType<TextBox>())
            {
                textbox.Text = "";
            }
        }

        private void btnBackToMenu_Click(object sender,
EventArgs e)
        {
            ActivateForm<formAdminMenu>();
        }
        }
```

```
}
```

formAdminMenu.cs

```
namespace ChessMasterQuiz.Forms.Admin;

public partial class formAdminMenu : Form
{
     private AdminConfiguration? _config = new();
     public formAdminMenu()
     {
     InitializeComponent();

     lblUserNotExist.Hide();
     lblPromoteYourself.Hide();

     txtBoxPromote.AutoCompleteMode =
AutoCompleteMode.Suggest;
     txtBoxPromote.AutoCompleteSource =
AutoCompleteSource.CustomSource;


txtBoxPromote.AutoCompleteCustomSource.AddRange(Users.Select(
x => x.Username).ToArray());

     List<CheckBox> checkBoxes =
Controls.OfType<CheckBox>().ToList();

     _config = GetAdminConfig();
     }

     private void btnCreatePuzzles_Click(object sender,
EventArgs e)
     {
     ActivateForm<formCreatePuzzle>();
```

```
    }

    private void btnCreateQuestions_Click(object sender,
EventArgs e)
    {
    ActivateForm<formAddQuestion>();
    }

    private void btnViewPuzzles_Click(object sender,
EventArgs e)
    {
    ActivateForm<formViewPuzzles>();
    }

    private void btnViewQuestions_Click(object sender,
EventArgs e)
    {
    // View questions
    }

    private void btnPromote_Click(object sender, EventArgs
e)
    {
    if (string.IsNullOrEmpty(txtBoxPromote.Text))
        return;

    string username = txtBoxPromote.Text;

    User? user = Users.FirstOrDefault(x => x.Username ==
username);

    if (user is null)
    {
        lblUserNotExist.Show();
        lblPromoteYourself.Hide();
        return;
    }

    if (user.Username == ActiveUser?.Username)
```

```
     {
          lblUserNotExist.Hide();
          lblPromoteYourself.Show();
          return;
     }

     user.Type = UserType.ADMIN;

     UpdateUser(user);

     txtBoxPromote.Text = "";
     }

     private void btnDemote_Click(object sender, EventArgs e)
     {
     if (string.IsNullOrEmpty(txtBoxPromote.Text))
          return;

     string username = txtBoxPromote.Text;

     User? user = Users.FirstOrDefault(x => x.Username ==
username);

     if (user is null)
     {
          return;
     }

     if (user.Username == ActiveUser?.Username)
     {
          MessageBox.Show($"You cannot demote yourself");
          return;
     }

     user.Type = UserType.USER;

     UpdateUser(user);

     txtBoxPromote.Text = "";
```

```
    }

    private void btnMainMenu_Click(object sender, EventArgs
e)
    {
    ActivateForm<formMenu>();
    }

    private void btnStrong_Click(object sender, EventArgs e)
    {
    if (_config is AdminConfiguration c)
    {
        c.PasswordRequirementLevel = STRONG;
        c.Write();
    }
    }

    private void btnMedium_Click(object sender, EventArgs e)
    {
    if (_config is AdminConfiguration c)
    {
        c.PasswordRequirementLevel = MEDIUM;
        c.Write();
    }
    }

    private void btnWeak_Click(object sender, EventArgs e)
    {
    if (_config is AdminConfiguration c)
    {
        c.PasswordRequirementLevel = WEAK;
        c.Write();
    }
    }
}
```

formCreatePuzzle.cs

```
namespace ChessMasterQuiz.Forms.Admin;

public partial class formCreatePuzzle : Form
{
     private readonly List<Piece> pieces = new();
     private readonly Board? _board = new();

     private PieceType? SelectedPiece = null;
     private Colour? SelectedColour = null;
     private PictureBox? SelectedPBox = null;

     public formCreatePuzzle()
     {
     InitializeComponent();
     _board.Location = new Point(50, 20);
     _board.Pieces.Clear();

     _board.BoardClick += (Square? s) =>
     {
          if (SelectedPiece is null)
          {
               return;
          }

          if (SelectedColour is null)
          {
               return;
          }

          pieces.Add(new Piece((PieceType)SelectedPiece,
From(s!.BoardLocation), (Colour)SelectedColour!));
          _board.Pieces = pieces;
          _board!.Invalidate();
     };

     foreach (PictureBox pbox in
Controls.OfType<PictureBox>())
```

```
     {
          pbox.Click += (s, e) =>
          {
               if (SelectedPBox is not null)
               {
                    SelectedPBox.BorderStyle =
BorderStyle.None;
               }
               SelectedPBox = pbox;
               SelectedPBox.BorderStyle =
BorderStyle.FixedSingle;
          };
     }
     Controls.Add(_board);

     }

     private void btnCreatePuzzle_Click(object sender,
EventArgs e)
     {
     string correct = txtBoxWinningMove.Text;
     string altOne = txtBoxAlternative1.Text;
     string altTwo = txtBoxAlternative2.Text;
     string altThree = txtBoxAlternative3.Text;

     Puzzle.Puzzles.Add(new Puzzle()
     {
          Rating = Convert.ToInt32(txtBoxRating.Text),
          ToMove = cBoxToMove.Checked ? White : Black,
          Setup = pieces,
          Moves = [correct, altOne, altTwo, altThree],
          CorrectMove = 0
     });

     Puzzle.Puzzles.WritePuzzles();

     foreach (TextBox tBox in Controls.OfType<TextBox>())
     {
          tBox.Text = "";
```

```
        }

        pieces.Clear();

        _board!.Invalidate();
        }

        private void btnBack_Click(object sender, EventArgs e)
        {
        ActivateForm<formAdminMenu>();
        }

        private void pBoxWhitePawn_Click(object sender,
EventArgs e)
        {
        SelectedPiece = PAWN;
        SelectedColour = White;
        }

        private void pBoxWhiteBishop_Click(object sender,
EventArgs e)
        {
        SelectedPiece = BISHOP;
        SelectedColour = White;
        }

        private void pBoxWhiteQueen_Click(object sender,
EventArgs e)
        {
        SelectedPiece = QUEEN;
        SelectedColour = White;
        }

        private void pBoxWhiteKnight_Click(object sender,
EventArgs e)
        {
        SelectedPiece = KNIGHT;
        SelectedColour = White;
        }
```

```
     private void pBoxWhiteRook_Click(object sender,
EventArgs e)
     {
     SelectedPiece = ROOK;
     SelectedColour = White;
     }

     private void pBoxWhiteKing_Click(object sender,
EventArgs e)
     {
     SelectedPiece = KING;
     SelectedColour = White;
     }

     private void pBoxBlackPawn_Click(object sender,
EventArgs e)
     {
     SelectedPiece = PAWN;
     SelectedColour = Black;
     }

     private void pBoxBlackKnight_Click(object sender,
EventArgs e)
     {
     SelectedPiece = KNIGHT;
     SelectedColour = Black;
     }

     private void pBoxBlackBishop_Click(object sender,
EventArgs e)
     {
     SelectedPiece = BISHOP;
     SelectedColour = Black;
     }

     private void pBoxBlackRook_Click(object sender,
EventArgs e)
     {
```

```
        SelectedPiece = ROOK;
        SelectedColour = Black;
        }

        private void pBoxBlackQueen_Click(object sender,
EventArgs e)
        {
        SelectedPiece = QUEEN;
        SelectedColour = Black;
        }

        private void pBoxBlackKing_Click(object sender,
EventArgs e)
        {
        SelectedPiece = KING;
        SelectedColour = Black;
        }
}
```

formViewPuzzles.cs

```
namespace ChessMasterQuiz.Forms.Admin;

public partial class formViewPuzzles : Form
{
        private Board _board = new();

        private List<Puzzle> _puzzles = new();
        private int _puzzleIndex = 0;
        public formViewPuzzles()
        {
        InitializeComponent();
        _board.Location = new Point(100, 10);
        Controls.Add(_board);
```

```
     Puzzle.CreatePuzzles();
     _puzzles = Puzzle.Puzzles;

     _board.Pieces = _puzzles[_puzzleIndex].Setup!.ToList();
     }


     private void btnMainMenu_Click(object sender, EventArgs
e)
     {
     ActivateForm<formAdminMenu>();
     }

     private void btnNextPuzzle_Click(object sender,
EventArgs e)
     {
     _puzzleIndex++;
     if(_puzzleIndex >= _puzzles.Count)
     {
          _puzzleIndex = 0;
     }
     _board.Pieces = _puzzles[_puzzleIndex].Setup!.ToList();
     }

     private void btnPreviousPuzzle_Click(object sender,
EventArgs e)
     {
     _puzzleIndex--;
     if(_puzzleIndex <= 0)
     {
          _puzzleIndex = _puzzles.Count - 1;
     }
     _board.Pieces = _puzzles[_puzzleIndex].Setup!.ToList();
     }
}
```

formViewQuestions.cs

formChooseQuiz.cs

```
using ChessMasterQuiz.Forms.Questions;
using ChessMasterQuiz.QuestionDir;

namespace ChessMasterQuiz.Forms.Menus;

public partial class formChooseQuiz : Form, IContext
{
     private User? _user = null;
     public void UseContext(IEnumerable<DCT> context)
     {
     User? user = context.GetFirst<User>(USER);

     if (user is null)
     {
          user = ActiveUser!;
     }

     _user = user;
     }

     public formChooseQuiz()
     {
     InitializeComponent();
     }

     private void btnWrittenQuestions_Click(object sender,
EventArgs e)
     {
     var file = File.ReadAllLines("questions.qon");

     var questions = QonConvert.Deserialize(file);
     questions =
questions.ToArray().AsSpan().Shuffle().ToList();

     questions = questions.Take(10).ToList();
```

```
     int questionIndex = 0;
     int questionsCorrect = 0;

     var onAnswer = EmptyActionGeneric<bool, int>();

     onAnswer += (bool correct, int elo) =>
     {
          if (correct)
          {
               questionsCorrect++;
               if (_user is User u)
               {
                    u.CorrectAnswersInRow++;
                    if (u.CorrectAnswersInRow > u.HighScore)
                    {
                    u.HighScore = u.CorrectAnswersInRow;
                    }
               }
          }
          else
          {
               if (_user is User u)
                    u.CorrectAnswersInRow = 0;
          }

          if (_user is User user)
          {
               if (elo != -1)
               {
                    ELO.Match(user.Elo, correct ?
ELO.MatchupResult.WIN : ELO.MatchupResult.LOSS, elo);
               }
          }

          if (questionIndex >= questions.Count)
          {
               // Update accuracy
               if (_user?.QuizesCompleted == 0)
```

```
                {
                    _user.Accuracy = questionsCorrect * 10;
                }
                else
                {
                    _user!.Accuracy = (int)((questionsCorrect
* 10) + _user.Accuracy) / 2;
                }

                // Go to exit screen
                ActivateForm<formResult>(
                    (questionsCorrect, QUESTIONS_CORRECT),
                    (questionIndex, INDEX),
                    (_user?.Elo.Rating!, NUMBER)
                    );
                return;
            }

            ActivateForm<formTextQuestion>(
                (questions[questionIndex++], QUESTION),
                (questionIndex.ToString(), NUMBER),
                (onAnswer, ACTION)
                );
        };

        onAnswer(false, -1);
        }

        private void btnPuzzles_Click(object sender, EventArgs
e)
        {
        List<Puzzle> puzzles = Puzzle.Puzzles;

        int questionIndex = 0;
        int questionsCorrect = 0;

        var onAnswer = EmptyActionGeneric<bool, int>();

        onAnswer += (bool correct, int elo) =>
```

```
    {
        if (correct)
        {
            questionsCorrect++;
            if (_user is User u)
            {
                u.CorrectAnswersInRow++;
                if (u.CorrectAnswersInRow > u.HighScore)
                {
                u.HighScore = u.CorrectAnswersInRow;
                }
            }
        }
        else
        {
            if (_user is User u)
                u.CorrectAnswersInRow = 0;
        }

        if (_user is User user)
        {
            if (elo != -1)
            {
                ELO.Match(user.Elo, correct ?
ELO.MatchupResult.WIN : ELO.MatchupResult.LOSS, elo);
            }
        }

        if (questionIndex >= puzzles.Count)
        {
            ActivateForm<formResult>(
                (questionsCorrect, QUESTIONS_CORRECT),
                (questionIndex, INDEX),
                (_user?.Elo.Rating!, NUMBER)
                );
            return;
        }

        ActivateForm<formPuzzleQuestion>(
```

```
                (puzzles[questionIndex++], PUZZLE),
                (questionIndex.ToString(), NUMBER),
                (onAnswer, ACTION)
                );
     };

     onAnswer(false, -1);
     }

     private void btnMainMenu_Click(object sender, EventArgs
e)
     {
     ActivateForm<formMenu>();
     }

     private void btnMixed_Click(object sender, EventArgs e)
     {
     List<Question> puzzles = new();


puzzles.AddRange(Puzzle.Puzzles.ToArray().AsSpan().Shuffle())
;
     var file = File.ReadAllLines("questions.qon");

     List<TextQuestion> readQuestions =
QonConvert.Deserialize(file).Select(x =>
(TextQuestion)x).ToList();
     IList<TextQuestion> questions =
readQuestions.ToArray().AsSpan().Shuffle();

     questions = questions.Take(5).ToList();

     puzzles.AddRange(questions);

     puzzles = puzzles.ToArray().AsSpan().Shuffle().ToList();

     int questionIndex = 0;
     int questionsCorrect = 0;
```

```
     var onAnswer = EmptyActionGeneric<bool, int>();

     onAnswer += (bool correct, int elo) =>
     {
          if (correct)
          {
               questionsCorrect++;
               if (_user is User u)
               {
                    u.CorrectAnswersInRow++;
                    if (u.CorrectAnswersInRow > u.HighScore)
                    {
                    u.HighScore = u.CorrectAnswersInRow;
                    }
               }
          }
          else
          {
               if (_user is User u)
                    u.CorrectAnswersInRow = 0;
          }

          if (_user is User user)
          {
               if (elo != -1)
               {
                    ELO.Match(user.Elo, correct ?
ELO.MatchupResult.WIN : ELO.MatchupResult.LOSS, elo);
               }
          }

          if (questionIndex >= puzzles.Count)
          {
               ActivateForm<formResult>(
                    (questionsCorrect, QUESTIONS_CORRECT),
                    (questionIndex, INDEX),
                    (_user?.Elo.Rating!, NUMBER)
                    );
               return;
```

```
          }

          if (puzzles[questionIndex] is Puzzle)
          {
               ActivateForm<formPuzzleQuestion>(
                    ((puzzles[questionIndex++] as Puzzle)!,
PUZZLE),
                    (questionIndex.ToString(), NUMBER),
                    (onAnswer, ACTION)
                    );
          }
          else if (puzzles[questionIndex] is TextQuestion)
          {
               ActivateForm<formTextQuestion>(
                    ((puzzles[questionIndex++] as
TextQuestion)!, QUESTION),
                    (questionIndex.ToString(), NUMBER),
                    (onAnswer, ACTION)
                    );
          }
     };

     onAnswer(false, -1);

     }
}
```

formLeaderboard.cs

```
namespace ChessMasterQuiz.Forms.Menus;

public partial class formLeaderboard : Form
{
     public formLeaderboard()
     {
```

```
     InitializeComponent();

     DisplayInformation();
     UpdateSortColumn(lblElo);
     }

     private void DisplayInformation(SortType sortType =
SortType.ELO)
     {
     List<Panel> panels = new();

panels.AddRange(Controls.OfType<Panel>().First().Controls.OfT
ype<Panel>());
     panels.OrderBy(x => int.Parse((string)x.Tag!));

     Span<User> users = SortbyType(Users,
sortType).Take(panels.Count).ToArray().AsSpan();

     for (int i = 0; i < users.Length; i++)
     {
          Label? name = panels[i].Controls.OfType<Label>()
               .FirstOrDefault(x => (string?)x.Tag ==
"username");
          Label? elo = panels[i].Controls.OfType<Label>()
               .FirstOrDefault(x => (string?)x.Tag == "elo");
          Label? accuracy =
panels[i].Controls.OfType<Label>()
               .FirstOrDefault(x => (string?)x.Tag ==
"accuracy");
          Label? high = panels[i].Controls.OfType<Label>()
               .FirstOrDefault(x => (string?)x.Tag ==
"high");

          name!.Text = $"{users[i].Username}";
          elo!.Text = $"{users[i].Elo.Rating}";
          accuracy!.Text = $"{users[i].Accuracy}";
          high!.Text = $"{users[i].HighScore}";
     }
     }
```

```
     private List<User> SortbyType(List<User> users, SortType
type)
     {
     List<User> sorted = users.OrderBy(x => type switch
     {
          SortType.ELO => x.Elo.Rating,
          SortType.HIGHSCORE => x.HighScore,
          SortType.ACCURACY => x.Accuracy,
          _ => x.Elo.Rating
     }).ToList();

     return sorted;
     }


     private void lblElo_Click(object sender, EventArgs e)
     {
     DisplayInformation(SortType.ELO);
     UpdateSortColumn(sender as Label);
     }

     private void lblAccuracy_Click(object sender, EventArgs
e)
     {
     DisplayInformation(SortType.ACCURACY);
     UpdateSortColumn(sender as Label);
     }

     private void lblHighScore_Click(object sender, EventArgs
e)
     {
     DisplayInformation(SortType.HIGHSCORE);
     UpdateSortColumn(sender as Label);
     }

     private void btnMainMenu_Click(object sender, EventArgs
e)
     {
```

```
        ActivateForm<formMenu>();
        }

        private void UpdateSortColumn(Label? label)
        {
        if (label is null) return;
        foreach (Label l in Controls.OfType<Label>())
        {
             l.Text = l.Text.Replace("*", "");
        }

        label.Text += "*";
        }

        private enum SortType
        {
        ELO,
        HIGHSCORE,
        GAMES,
        ACCURACY
        }
}
```

formLogin.cs

```
namespace ChessMasterQuiz;

public partial class formLogin : Form
{
        public formLogin()
        {
        InitializeComponent();
        lblIncorrectDetails.Hide();

        // Fetch random logo image
```

```
        pBoxLogo.Image = GetLogo();
        pBoxLogo.SizeMode = PictureBoxSizeMode.StretchImage;

        btnRegister.Click += (s, e) =>
            ActivateForm<formRegister>();
        }

        private void btnLogin_Click(object sender, EventArgs e)
        {
        string usernameText = txtBoxEmail.Text;
        string passwordText = txtBoxPassword.Text;

        var foundUser = Users.FirstOrDefault(x => x.Username ==
usernameText);

        if (foundUser is null)
        {
            lblIncorrectDetails.Show();
            return;
        }

        if (!passwordText.VerifyPassword(foundUser!))
        {
            lblIncorrectDetails.Show();
            return;
        }

        foundUser.Login();
        // Pass the user as context through to formMenu
        ActivateForm<formMenu>((foundUser, USER));
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
        Environment.Exit(0);
        }
}
```

formMenu.cs

```
using ChessMasterQuiz.Forms;
using ChessMasterQuiz.Forms.Admin;
using ChessMasterQuiz.Forms.Menus;

namespace ChessMasterQuiz;

public partial class formMenu : Form, IContext
{
     private User? _user = null;

     public void UseContext(IEnumerable<DataContextTag>
context)
     {
     var userContext = (User?)context.GetFirst(USER);

     if (userContext is null)
     {
          userContext = ActiveUser;
     }

     _user = userContext!;

     if (_user.Type != UserType.ADMIN)
     {
          btnSettings.Hide();
     }

     pBoxProfile.Image =
User.ProfilePictures[_user!.ImageIndex];
     }

     public formMenu()
     {
     InitializeComponent();
```

```
     pBoxProfile.BackgroundImageLayout = ImageLayout.Stretch;
     pBoxProfile.SizeMode = PictureBoxSizeMode.CenterImage;

     Board board = new();

     PgnReader reader = new();

     reader.FromBytes(PGNLibrary.Steinitz_Best_Games);
     PGN game = new();

     if (reader.Games.Count != 0)
     {
          game = reader.Games.First();
     }

     board.Location = new Point(300, 25);
     Controls.Add(board);

     board.DisplayGame(game);
     }

     private void pBoxProfile_Click(object sender, EventArgs
e)
     {
     MoveHelper.CurrentBoard?.StopGame();
     ActivateForm<formUserProfile>((_user!, USER));
     }

     private void btnPlay_Click(object sender, EventArgs e)
     {
     MoveHelper.CurrentBoard?.StopGame();
     ActivateForm<formChooseQuiz>((_user!, USER));
     }

     private void btnLeaderboard_Click(object sender,
EventArgs e)
     {
     MoveHelper.CurrentBoard?.StopGame();
     ActivateForm<formLeaderboard>();
```

```
    }

    private void btnSettings_Click(object sender, EventArgs
e)
    {
    MoveHelper.CurrentBoard?.StopGame();
    ActivateForm<formAdminMenu>();
    }

    private void btnLogout_Click(object sender, EventArgs e)
    {
    ActiveUser!.Logout();
    UpdateUser(ActiveUser!);
    ActivateForm<formLogin>();
    }
}
```

formRegister.cs

```
using System.Net.Mail;
using static ChessMasterQuiz.Misc.ValidationType;

namespace ChessMasterQuiz;
public partial class formRegister : Form, IContext
{
    private readonly Dictionary<ValidationType, ProgressBar>
ValidationToBarMap = new();
    private readonly Dictionary<ValidationType, bool>
ValidationToBoolMap = new()
    {
    { USERNAME, false },
    { PASSWORD, false },
    { PASSWORD_CONFIRM, false },
    { ValidationType.EMAIL, false },
    { DOB, false }
```

```
        };

        public formRegister()
        {
        InitializeComponent();

        btnRegister.Enabled = false;
        ValidationToBarMap = new()
        {
              { DISPLAY, pBarDisplayName },
              { USERNAME, pBarUsername },
              { DOB, pBarDob },
              { ValidationType.EMAIL, pBarGender },
              { PASSWORD_CONFIRM, pBarDisplayName }
        };

        pBoxLogo.Image = GetLogo();
        pBoxLogo.SizeMode = PictureBoxSizeMode.StretchImage;
        progressPassword.Maximum = 100;
        TextBox? passwordBox =
Controls.OfType<TextBox>().FirstOrDefault(x => (x.Tag as
string) == "password");
        foreach (var control in Controls.OfType<TextBox>())
        {
              control.TextChanged += (s, _) =>
              {
                    var sender = (s as TextBox)!;
                    ValidationType vt =
(ValidationType)Enum.Parse(typeof(ValidationType),
sender.Tag!.ToString()!.ToUpper());
                    (var val, var percentage) =
sender.Text.Validate(vt);
                    if (vt is PASSWORD)
                    {
                          progressPassword.Value =
Math.Min(Math.Max(0, (int)percentage), 100);
                    }
                    else
                    {
```

```
                    if (vt is PASSWORD_CONFIRM &&
passwordBox?.Text == sender.Text)
                    {
                    pBarDisplayName.Value = 100;
                    val = true;
                    }
                    else ValidationToBarMap[vt].Value = val ?
(int)percentage : 0;
               }

               ValidationToBoolMap[vt] = val;

               if (ValidationToBoolMap.All(x => x.Value ==
true))
               {
                    btnRegister.Enabled = true;
               }
               else
               {
                    btnRegister.Enabled = false;
               }
          };
     }
     }

     public void UseContext(IEnumerable<DataContextTag>
context)
     {
     foreach (var dct in context)
     {
          if (dct.tag == ContextTagType.EMAIL)
          {
               txtBoxEmail.Text = (string)dct.data;
          }
     }
     }

     private void btnRegister_Click(object sender, EventArgs
e)
```

```
     {
     Dictionary<ValidationType, string> ValidationToTextBox =
new()
     {
          // Fields
          { USERNAME,  txtBoxEmail.Text },
          { PASSWORD, txtBoxPassword.Text },
          { ValidationType.EMAIL, txtBoxDisplayName.Text } ,
          { DOB, txtBoxDob.Text }
     };

     var enumValues =
Enum.GetValues(typeof(ValidationType)).Cast<ValidationType>()
;

     foreach (var val in
enumValues.Where(ValidationToTextBox.ContainsKey))
     {
          if (!ValidationToBoolMap[val])
          {
               return;
          }
     }

     var _ =
MailAddress.TryCreate(ValidationToTextBox[ValidationType.EMAI
L], out var addy);

     var user = new User(
          name: ValidationToTextBox[USERNAME],
          password: ValidationToTextBox[PASSWORD]
          )
     {
          Email = addy,
          DOB = DateTime.Parse(ValidationToTextBox[DOB]),
          Username = ValidationToTextBox[USERNAME]
     };

     Users.Add(user);
```

```
     WriteUser(user);

     ActivateForm<formLogin>();
     }

     private void btnBack_Click(object sender, EventArgs e)
     {
     ActivateForm<formLogin>();
     }

}
```

formUserProfile.cs

```
namespace ChessMasterQuiz.Forms;

public partial class formUserProfile : Form, IContext
{
     public formUserProfile()
     {
     InitializeComponent();
     }

     private User? _user;
     public void UseContext(IEnumerable<DataContextTag>
context)
     {
     User? user = (User)context.Get(USER).First();

     if (user is null)
     {
          user = ActiveUser!;
```

```
        }

        lblNewPasswordInvalid.Hide();
        lblOldPasswordIncorrect.Hide();

        _user = user;

        lblUsername.Text = user.Username;
        lblAccuracyValue.Text = user.Accuracy.ToString();
        lblQuizCompleteValue.Text =
user.QuizesCompleted.ToString();
        lblTopScoreValue.Text = user.HighScore.ToString();
        lblRatingValue.Text = user.Elo.Rating.ToString();

        pBoxProfileImage.Image =
User.ProfilePictures[_user.ImageIndex];
        pBoxProfileImage.BackgroundImageLayout =
ImageLayout.Tile;
        pBoxProfileImage.SizeMode =
PictureBoxSizeMode.CenterImage;
        }

        private void lblMainMenu_Click(object sender, EventArgs
e)
        {
        WriteUsers();
        ActivateForm<formMenu>((_user!, USER));
        }

        private void btnNext_Click(object sender, EventArgs e)
        {
        if (_user is null)
        {
            return;
        }

        if (_user.ImageIndex == User.ProfilePictures.Count - 1)
        {
            _user.ImageIndex = 0;
```

```
        }
        else
        {
            _user.ImageIndex++;
        }

        pBoxProfileImage.Image =
User.ProfilePictures[_user.ImageIndex];

        WriteUsers();
        }

        private void btnChangePassword_Click(object sender,
EventArgs e)
        {
        string? old = txtBoxOldPassword.Text;
        string? @new = txtBoxNewPassword.Text;


        if (!old.VerifyPassword(_user!))
        {
            lblOldPasswordIncorrect.Show();
            return;
        }

        (bool valid, var _) =
@new.Validate(ValidationType.PASSWORD);

        if (_user!.Type == UserType.USER && !valid)
        {
            lblNewPasswordInvalid.Show();
            return;
        }

        Password hashedPassword = @new.Hash();

        _user!.Password = hashedPassword;

        UpdateUser(_user!);
```

```
    }
}
```

formPuzzleQuestion.cs

```
namespace ChessMasterQuiz.Forms.Questions;

public partial class formPuzzleQuestion : Form, IContext
{
    public formPuzzleQuestion()
    {
    InitializeComponent();
    }

    public Action<bool, int> OnAnswered { get; set; } =
EmptyActionGeneric<bool, int>();

    public void UseContext(IEnumerable<DCT> context)
    {
    Puzzle? puzzle = context.GetFirst<Puzzle>();
    if (puzzle is null)
    {
        return; // Error here?
    }
    List<string>? answers = puzzle.Moves;

    lblWinningMove.Text += $"\nfor {(puzzle.ToMove == White
? "white" : "black")}?";

    List<Button> buttons = new()
    {
        btnAnswer1,
        btnAnswer2,
```

```
            btnAnswer3,
            btnAnswer4,
      };

      OnAnswered = context.GetFirst<Action<bool,
int>>(ACTION)!;

      foreach (var b in buttons)
      {
            b.Click += (s, e) =>
            {
                  if (OnAnswered is Action<bool, int> answer)
                  {
                        var ans = buttons.IndexOf(b) ==
puzzle.CorrectMove;

                        answer(ans, puzzle!.Rating);
                  }
            };
      }

      if (answers?.Count != buttons.Count)
      {
            return; // Error?
      }

      for (int i = 0; i < answers.Count; i++)
      {
            buttons[i].Text = answers[i];
      }

      label1.Text = $"{context.GetFirst<string>(NUMBER)} /
10";

      Board board = new();
      board.Location = new(30, 30);
      board.Pieces = puzzle.Setup!.ToList();

      Controls.Add(board);
```

```
    }
}
```

formResult.cs

```
using ChessMasterQuiz.Forms.Menus;

namespace ChessMasterQuiz.Forms.Questions;

public partial class formResult : Form, IContext
{
    public formResult()
    {
    InitializeComponent();
    }

    public void UseContext(IEnumerable<DCT> context)
    {
    var correct = (int)context.GetFirst(QUESTIONS_CORRECT)!;
    var questionIndex = (int)context.GetFirst(INDEX)!;

    var elo = (int)context.GetFirst(NUMBER)!;

    lblQuestion.Text = $"{correct}/{questionIndex}";
    label1.Text =
$"{ActiveUser?.Elo.PastRatings[^questionIndex]} -> {elo}";
    label1.Center(X);
    lblQuestion.Center(X);

    ActiveUser!.QuizesCompleted++;
    WriteUsers();
    }

    private void btnHome_Click(object sender, EventArgs e)
    {
```

```
        MainMenu();
        }

        private void btnLeaderboard_Click(object sender,
EventArgs e)
        {
        ActivateForm<formLeaderboard>();
        }
}
```

formTextQuestion.cs

```
namespace ChessMasterQuiz.QuestionDir;

public partial class formTextQuestion : Form, IContext
{
        private readonly List<Button> _buttons;
        private TextQuestion? _tq = null;
        public formTextQuestion()
        {
        InitializeComponent();

        // Buttons
        // C# 12 Collection Initializer
        _buttons = [
                btnAnswer1,
                btnAnswer2,
                btnAnswer3,
                btnAnswer4
                ];


        // Ensure that each button has a click delegate attached

        // This delegate should check if the answer is correct,
and then open the next question
```

```
     foreach (var b in _buttons)
     {
          b.Click += (s, e) =>
          {
               if (OnAnswered is Action<bool, int> answer)
               {
                    var ans = _buttons.IndexOf(b) ==
_tq?.A?.Index;
                    answer(ans, _tq!.Rating);
               }
          };
     }
     }

     public Action<bool, int> OnAnswered { get; set; } =
EmptyActionGeneric<bool, int>();

     public void UseContext(IEnumerable<DataContextTag>
context)
     {
     var questionData = context.GetFirst(QUESTION) as
TextQuestion;
     if (questionData is TextQuestion q)
     {
          _tq = q;
     }

     if (questionData is null)
     {
          ActivateForm<formMenu>();
     }

     for (int i = 0; i < _buttons.Count; i++)
     {
          if (_buttons[i] is null)
          {
               continue;
          }
```

```
            if (questionData?.A?.Answers.Count <= i)
            {
                continue;
            }

            _buttons[i].Text = questionData?.A?.Answers[i];
        }

        lblQuestion.Text = questionData?.Q;

        var number = context.GetFirst(NUMBER) as string;

        if (number is null)
        {
            ActivateForm<formMenu>();
        }

        lblNumber.Text = $"{number} / 10";

        var onAnswered = context.GetFirst(ACTION);

        if (onAnswered is null)
        {
            ActivateForm<formMenu>();
        }

        OnAnswered += (Action<bool, int>)onAnswered!;

        lblQuestion.Location = new Point((Width / 2) -
(lblQuestion.Width / 2), lblQuestion.Location.Y);
        }
}
```

formMain.cs

```
using ChessMasterQuiz.Forms;
```

```
namespace ChessMasterQuiz;

public partial class formMain : Form
{
     public static Form? ChildForm { get; set; } = null;
     public static Func<Panel>? GetPanelHolder { get; set; }
= null;

     public formMain()
     {
     InitializeComponent();
     GetPanelHolder = () => panelHolder;
     Text = "Chess Master Quiz";
     FormBorderStyle = FormBorderStyle.FixedSingle;
     CenterToScreen();

     Users.ForEach(x => x.Logout());

     ActivateForm<formSplashScreen>();
     }
}
```

formSplashScreen.cs

```
namespace ChessMasterQuiz.Forms;

public partial class formSplashScreen : Form
{
     private float progress = 0;
     public formSplashScreen()
     {
     InitializeComponent();

     pnlProgressBar.Paint += (s, e) =>
```

```
     {
          e.Graphics.FillRectangle(Brushes.LightGreen, new
Rectangle(0, 0, (int)(pnlProgressBar.Width * progress),
pnlProgressBar.Height));
     };

     System.Timers.Timer timer = new(0.05);

     timer.Elapsed += (s, e) =>
     {
          progress += 0.005f;
          if (progress >= 1.0f)
          {
               timer.Stop();
          }

          pnlProgressBar.Invalidate();
     };

     timer.Start();

     WaitForTimer();

     Board board = new();
     board.Location = new
Point((int)((0.5*Width)-(0.5*board.Width)), Location.Y);
     Controls.Add(board);
     board.DisplayGame(ChessHelper.GetScholarsMate(), 500);
     }

     private async void WaitForTimer()
     {
     await Task.Delay(3250);
     ActivateForm<formLogin>();
     }
}
```

Board.cs

```
using static Chess.ChessHelper;

namespace Chess.BoardRepresentation;

public class Board : Panel
{
     // These indexers allow for easy grabbing of a piece.
     public Piece? this[string location] =>
     Pieces.FirstOrDefault(x =>
x.Location!.Equals(location));
     public Piece? this[Notation location] =>
     Pieces.FirstOrDefault(x => x.Location! == location);

     // This is a callback action which gets called when the
board is clicked
     // The square which was clicked gets passed through
     public Action<Square?> BoardClick =
EmptyActionGeneric<Square?>();

     // The normal board size
     private static readonly Size _defaultBoardSize =
new(400, 400);

     private int _squareWidth => Size.Width / 8;

     // Which square was last selected
     public Square? SelectedSquare = null;

     // All of the squares on the boarad
     public List<Square> Squares = new();

     // The pieces that should be displayed on the board
     public List<Piece> Pieces = new();

     public Board(Size? size = null) : base()
     {
```

```
     // Get rid of flickering in the board
     SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
     MoveHelper.CurrentBoard = this;
     base.BackColor = Color.Gray;
     size ??= _defaultBoardSize;
     Size = (Size)size;

     // Loop through all squares coords on the board and
create the representive Square object
     for (int i = 0; i < 8; i++)
     {
          for (int j = 8; j >= 0; j--)
          {
               Squares.Add(
                    new Square(
                    (i + j) % 2 == 0
                         ? Black
                         : White,
                    new Point(i * 50, 350 - (j * 50)),
                    ((char)(i + 97), j + 1)
                         ));
          }
     }

     // Generate the default pieces
     Pieces = PopulatePieces();

     Paint += Draw;
     Click += (object? s, EventArgs e) =>
     {
          var pos = PointToClient(MousePosition);

          // Gets the point of the square clostest to where
the user clicked
          Point roundedPoint = new(
               (int)Math.Floor(pos.X / 50m) * 50,
               (int)Math.Floor(pos.Y / 50m) * 50
               );
```

```
            foreach (var square in Squares)
            {
                // Run the callback
                if (square.Location == roundedPoint)
                {
                    SelectedSquare = square;
                    BoardClick(SelectedSquare);
                }
            }

            // Display click on board
            Invalidate();
        };

        Invalidate();
        }


        private void Draw(object? sender, PaintEventArgs e)
        {
        e.Graphics.CompositingMode =
System.Drawing.Drawing2D.CompositingMode.SourceOver;
        e.Graphics.CompositingQuality =
System.Drawing.Drawing2D.CompositingQuality.HighSpeed;
        e.Graphics.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.NearestNeighbor;
        e.Graphics.PixelOffsetMode =
System.Drawing.Drawing2D.PixelOffsetMode.Half;
        e.Graphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.HighSpeed;
        foreach (var square in Squares)
        {
            // Draw all the black and white squares
            e.Graphics.FillRectangle(
                square.Colour == White ? Brushes.LightGray :
Brushes.DarkGray,
                new Rectangle(square.Location, new(50, 50))
            );
```

```
          if (SelectedSquare is Square key)
          {
               // Draw green if square clicked
               if (key == square)
               {
                    e.Graphics.FillRectangle(
                    square.Colour == White ?
Brushes.LightGreen : Brushes.DarkGreen,
                    new Rectangle(square.Location, new(50,
50))
                    );
               }
          }


          foreach (var piece in Pieces)
          {
               if
(MoveHelper.SquareIsNotation(From(square.BoardLocation),
piece.Location))
               {
                    // Draw the piece!
                    var image = piece.GetImage();
                    e.Graphics.DrawImage(image,
image.GetCenter(square.Location, new(_squareWidth,
_squareWidth)));
               }
          }


     }
     }

     // Stop the external thread
     public void StopGame()
     {
     _continueGame = false;
     }
```

```
     private bool _continueGame = true;

     // Go through all the moves in the PGN and display at
interval, 'PlyPause'
     public void DisplayGame(PGN pgn, int PlyPause = 1000,
Action? onGameOver = null)
     {
     Thread gameThread = new(() =>
     {
          _continueGame = true;
          foreach (var (white, black) in pgn.Moves)
          {
               if (white.NullMove) break;
               this[white.InitialSquare]?.Move(white);
               Thread.Sleep(PlyPause);
               if (black is null)
               {
                    break;
               }
               if (black.NullMove) break;
               this[black.InitialSquare]?.Move(black);
               Thread.Sleep(PlyPause);
               if (!_continueGame)
               {
                    break;
               }

               if(onGameOver is Action action)
               {
                    action();
               }
          }
     });
     gameThread.Start();

     }
}
```

Square.cs

```
namespace Chess.BoardRepresentation;

public class Square : Panel
{
     public static readonly Size _defaultSquareSize = new(50,
50);

     // Where the square is on the board ( Notation )
     public (char, int) BoardLocation { get; set; }

     // The screen coords of the square
     new public Point Location { get; set; } = new();
     public Colour Colour { get; init; }
     new public Label Text { get; set; } = new();

     public Square(Colour colour, Point location, (char, int)
boardLocation) : base()
     {
     Colour = colour;

     // Their actual colour in terms of winforms
     BackColor = Colour switch
     {
          Black => Color.Black,
          White => Color.White,
          _ => Color.White
     };

     Padding = new(0, 0, 0, 0);
     Margin = new(0, 0, 0, 0);


     BoardLocation = boardLocation;
     Size = _defaultSquareSize;
```

```
        Location = location;

        Controls.Add(Text);
        }
}
```

ChessHelper.cs

```
namespace Chess;

// Represents the two colours regardless of their RGB values
public enum Colour
{
        White = 1,
        Black = 2
}

// A way to represent a place on the chess board
public record struct Notation(char File, int Rank) :
IEquatable<SAN>, IEquatable<string>
{
        public override string ToString() => $"{File}{Rank}";

        // Functional programming inspired
        public static Notation From((char, int) location)
        {
        return new Notation(location.Item1, location.Item2);
        }

        public static Notation From(string location)
        {
        if (location.Length != 2)
        {
                return default;
```

```
        }

        return new Notation(location[0],
int.Parse(location[1].ToString()));
        }

        public static Notation From(char file, int rank)
        {
        return new Notation(file, rank);
        }

        public bool Equals(SAN? other)
        {
        return ToString() == other?.GetNotation();
        }

        public bool Equals(string? other)
        {
        return ToString() == other;
        }
}

public static class ChessHelper
{
        // Gets the corresponding image of the piece
        public static Image GetImage(this Piece piece)
        {
        return piece.Type switch
        {
                PAWN => piece.Colour == White ? WhitePawn :
BlackPawn,
                KNIGHT => piece.Colour == White ? WhiteKnight :
BlackKnight,
                BISHOP => piece.Colour == White ? WhiteBishop :
BlackBishop,
                ROOK => piece.Colour == White ? WhiteRook :
BlackRook,
                QUEEN => piece.Colour == White ? WhiteQueen :
BlackQueen,
```

```
          KING => piece.Colour == White ? WhiteKing :
BlackKing,
          _ => WhitePawn
     };
     }

     // For the splash screen ( 4 Move Checkmate )
     public static PGN GetScholarsMate()
     {
     return new PGN(
          new List<(SAN, SAN)>()
          {
               (SAN.From("e4"), SAN.From("e5")),
               (SAN.From("Bc4"), SAN.From("Nc6")),
               (SAN.From("Qh5"), SAN.From("Nf6")),
               (SAN.From("Qxf7#"), SAN.From("null")),
          }
          );
     }

     // Readlly useful extensoin method to turn array into
IEnumerable while keeping some functional programming ideas
     public static IEnumerable<T> From<T>(params T[] nums)
     {
     foreach (var n in nums)
     {
          yield return n;
     }
     }


     // Grabs piece from char
     public static PieceType GetPiece(this char c)
     {
     switch (c.ToString().ToLower())
     {
          case "n":
               return KNIGHT;
          case "b":
```

```
                return BISHOP;
            case "r":
                return ROOK;
            case "q":
                return QUEEN;
            case "K":
                return KING;
        }

        return PAWN;
        }

        public static char GetChar(this PieceType type)
        {
        return type switch
        {
            PAWN => ' ',
            KNIGHT => 'N',
            BISHOP => 'B',
            ROOK => 'R',
            QUEEN => 'Q',
            KING => 'K',
            _ => '_'

        };
        }

        public static Dictionary<char, PieceType> CharToNotation
= new()
        {
        { 'p', PAWN },
        { 'n', KNIGHT },
        { 'b', BISHOP },
        { 'r', ROOK },
        { 'q', QUEEN },
        { 'k', KING },
        { 'P', PAWN },
        { 'N', KNIGHT },
        { 'B', BISHOP },
```

```
     { 'R', ROOK },
     { 'Q', QUEEN },
     { 'K', KING }
     };

     public static Dictionary<PieceType, char> NotationToChar
= new()
     {
     { PAWN , 'p'},
     { KNIGHT , 'n'},
     { BISHOP , 'b'},
     { ROOK , 'r'},
     { QUEEN , 'q'},
     { KING , 'k'},
     { PAWN , 'P'},
     { KNIGHT , 'N'},
     { BISHOP , 'B'},
     { ROOK , 'R'},
     { QUEEN , 'Q'},
     { KING , 'K'}
     };


     public static Point GetCenter(this Image image, Point
location, Size size)
     {
     var imageSize = image.Size;

     PointF midpoint = new(
          location.X + (float)0.5 * size.Width,
          location.Y + (float)0.5 * size.Height
          );

     PointF accountForImage = new(
          midpoint.X - (float)0.5 * imageSize.Width,
          midpoint.Y - (float)0.5 * imageSize.Height
          );
```

```
     return Point.Round(accountForImage);
     }


     // Generate all the pieces in a normal starting chess
position
     public static List<Piece> PopulatePieces()
     {
     List<Piece> Pieces = new();


     // Pawns
     Pieces.AddRange(
          Enumerable.Range(1, 2)
          .Select(x =>
               Enumerable.Range(1, 8)
                    .Select(y => new Piece(
                    PAWN, Notation.From((char)(y + 96), x ==
1 ? 2 : 7), x == 1 ? White : Black)
                    ).ToList()).Aggregate((x, y) =>
                    {
                         y.AddRange(x);
                         return y;
                    })
               );


     // Knights
     Pieces.AddRange(
          Enumerable.Range(1, 2).Select(colour =>
               From(2, 7).Select(
                    x => new Piece(KNIGHT,
Notation.From((char)(x + 96), (colour == 1 ? 1 : 8)),
(Colour)colour))
                    ).Aggregate((x, y) =>
                    {
                         return y.Concat(x);
                    }
          ));
```

```
     // Bishops
     Pieces.AddRange(
          Enumerable.Range(1, 2).Select(colour =>
               From(3, 6).Select(
                    x => new Piece(BISHOP,
Notation.From((char)(x + 96), (colour == 1 ? 1 : 8)),
(Colour)colour))
                    ).Aggregate((x, y) =>
                    {
                         return y.Concat(x);
                    }
          ));

     // Rooks
     Pieces.AddRange(
          Enumerable.Range(1, 2).Select(colour =>
               From(1, 8).Select(
                    x => new Piece(ROOK,
Notation.From((char)(x + 96), (colour == 1 ? 1 : 8)),
(Colour)colour))
                    ).Aggregate((x, y) =>
                    {
                         return y.Concat(x);
                    }
          ));

     // Queens
     Pieces.AddRange(
          Enumerable.Range(1, 2).Select(colour =>
                    new Piece(QUEEN, Notation.From('d',
(colour == 1 ? 1 : 8)), (Colour)colour))
          );

     // Kings
     Pieces.AddRange(
          Enumerable.Range(1, 2).Select(colour =>
                    new Piece(KING, Notation.From('e',
```

```
(colour == 1 ? 1 : 8)), (Colour)colour))
          );


     // Ensure only the pieces which fit on the board are in
the piece list
     foreach (var piece in Pieces)
     {
          if (piece.Location.Rank == 9)
          {
               Pieces.Remove(piece);
          }
     }

     return Pieces;
     }


}
```

MoveHelper.cs

```
namespace Chess;

public static class MoveHelper
{
     // Holds a reference to the chess board that is being
displayed at the current moment
     public static Board? CurrentBoard = null;

     public static bool CheckIfPieceOnSquare(Notation
location)
     {
```

```
     if (CurrentBoard is Board board)
     {
          foreach (var piece in board.Pieces)
          {
               if (piece.Location == location)
                    return true;
          }
     }

     return false;
     }

     public static bool SquareIsNotation(Notation square,
Notation move)
     {
     return square == move;
     }

     public static bool CheckIfCheck()
     {
     return false;
     }

     public static IEnumerable<Piece>
GetPieceThatCouldMove(Notation location)
     {
     if (CurrentBoard is null)
     {
          throw new Exception("CurrentBoard must not be
null");
     }

     List<Piece> pieces = CurrentBoard.Pieces;

     Func<Notation, Notation, Board, bool> validateMove;

     for (int i = 0; i < pieces.Count; i++)
     {
          validateMove = pieces[i] switch
```

```
          {
               { Type: PieceType.PAWN } => PawnMove,
               { Type: PieceType.KNIGHT } => KnightMove,
               { Type: PieceType.BISHOP } => BishopMove,
               _ => PawnMove
          };

          if (validateMove is Func<Notation, Notation, Board,
bool> meth)
          {
               if (meth(pieces[i].Location, location,
CurrentBoard))
               {
                    yield return pieces[i];
               }
          }
     }
     }

     public static bool PawnMove(Notation start, Notation
destination, Board? board = null)
     {
     if (board is null && CurrentBoard is null)
     {
          throw new Exception("Board must be instantiated");
     }

     if (board is Board b)
     {
          CurrentBoard = b;
     }


     return true;
     }

     public static bool KnightMove(Notation start, Notation
destination, Board? board = null)
     {
```

```
        return false;
        }

        public static bool BishopMove(Notation start, Notation
destination, Board? board = null)
        {
        return false;
        }


}
```

PgnReader.cs

```
using System.Text;

namespace Chess;

// Represents ONE pgn game
public record class PGN(
        string Event,
        string Site,
        DateTime? Date,
        int Round,
        string White,
        string Black,
        string Result,
        string? Annotator,
        int? PlyCount,
        string? TimeControl,
        DateTime? Time,
        string? Termination,
        string? Mode,
        string? FEN,
        List<(SAN, SAN)> Moves
        )
{
```

```
     public PGN(List<(SAN, SAN)> moves) : this(string.Empty,
string.Empty, null, -1, string.Empty, string.Empty,
string.Empty, null, null, null, null, null, null, null,
new())
     {
     Moves = moves;
     }

     public PGN() : this(new List<(SAN, SAN)>())
     {

     }
}

public class PgnReader
{
     // Holds a list of the currently parsed PGNs
     public List<PGN> Games { get; set; } = new();

     // This method is useful when reading straight from a
.resx file
     public void FromBytes(params byte[] bytes)
     {
     string str = Encoding.Default.GetString(bytes);

     // Splitting by double new line
     string[] parts = str.Split(new string[]
     {
          "\r\n\r\n"
     },
     StringSplitOptions.RemoveEmptyEntries);


     PGN pgn = new(string.Empty, string.Empty, default, -1,
string.Empty, string.Empty,
          string.Empty, null, null, null, null, null, null,
null, new());
```

```
     // Lex both parts of the game
     var meta = LexMetadata(parts[0]).Where(x => x.Type ==
TokenType.KEY
                                             || x.Type ==
TokenType.VALUE).ToList();

     var game = LexGame(parts[1]).ToList();

     // Do something with tokens to convert to PGN
     pgn = MetaTokensToPgn(pgn, meta);

     // Convert all the moves into SAN
     for (int j = 0; j < game.Count; j += 2)
     {
          (SAN, SAN?) move;

          if (j + 1 == game.Count)
          {
               move = (SAN.From((string)game[j].Data!),
null);
          }

          else
          {
               move = (SAN.From((string)game[j].Data!),
SAN.From((string)game[j + 1].Data!));
          }
          pgn.Moves.Add(move!);
     }

     Games.Add(pgn);


     }

     private PGN MetaTokensToPgn(PGN pgn, List<Token> meta)
     {
     var props = typeof(PGN).GetProperties();
     for (int i = 0; i < meta.Count(); i++)
```

```
     {
          if (meta[i].Type != TokenType.KEY)
          {
               continue;
          }

          PropertyInfo? property = props.FirstOrDefault(x =>
x.Name == (string)meta[i].Data!);

          // If the property is found
          if (property is PropertyInfo prop)
          {
               if (((string)meta[i + 1].Data!).Contains("?"))
               {
                    continue;
               }
               if (prop.PropertyType == typeof(string))
               {
                    prop.SetMethod?.Invoke(pgn, new object[]
{ (string)meta[i + 1].Data! });
               }
               else if (prop.PropertyType ==
typeof(DateTime?))
               {
                    DateTime date = Convert.ToDateTime(meta[i
+ 1].Data);
                    prop.SetMethod?.Invoke(pgn, new object[]
{ date });
               }
               else if (prop.PropertyType == typeof(int?))
               {
                    int num = Convert.ToInt32(meta[i +
1].Data);
                    prop.SetMethod?.Invoke(pgn, new object[]
{ num });
               }
               else
               {
                    var data = Convert.ChangeType(meta[i +
```

```
1].Data, prop.PropertyType);
                    prop.SetMethod?.Invoke(pgn, new object[]
{ data! });
                }
            }

            i++;
        }

        return pgn;
        }

        enum TokenType
        {
        KEY,
        VALUE,
        NOTATION,
        LEFT_BRACKET,
        RIGHT_BRACKET,
        LEFT_CURLY,
        RIGHT_CURLY
        }

        record struct Token(TokenType Type, object? Data)
        {
        public override string ToString()
        {
            return $"{Type}: {Data ?? ""}";
        }
        }

        private IEnumerable<Token> LexMetadata(string str)
        {
        int _current = 0;

        var next = (int n) => _current += n;
        var current = () => str[_current];

        var consume = (TokenType type, object? data = null) =>
```

```
     {
          _current++;
          return new Token(type, data);
     };

     Type pgnType = typeof(PGN);
     var props = pgnType.GetProperties();

     for (; _current < str.Length;)
     {
          switch (current())
          {

               case '[':
                    yield return
consume(TokenType.LEFT_BRACKET);
                    break;

               case ']':
                    yield return
consume(TokenType.RIGHT_BRACKET);
                    break;



               case '"':
                    {

                    StringBuilder sb = new();
                    next(1);
                    while (current() != '"')
                    {
                         sb.Append(current());
                         next(1);
                    }

                    yield return consume(
                         TokenType.VALUE,
                         sb.ToString()
```

```
                         );
                    break;
                    }


               default:
                    {
                    if (!char.IsLetterOrDigit(current()))
                    {
                         next(1);
                         break;
                    }
                    StringBuilder sb = new();
                    while (char.IsLetterOrDigit(current()))
                    {
                         sb.Append(current());
                         next(1);
                    }

                    yield return consume(
                         TokenType.KEY,
                         sb.ToString()
                         );

                    continue;
                    }
          }
     }


     }

     private static readonly Dictionary<char, char>
SkipperToCloserMap = new()
     {
     { '{', '}' },
     { '(', ')' },
     { '[', ']' },
     };
```

```
     private IEnumerable<Token> LexGame(string str)
     {
     int _current = 0;

     var next = (int n = 1) => _current += n;
     var peek = () => str[_current++];
     var current = () =>
     {

          if (_current >= str.Length)
          {
               return str[^1];
          }

          return str[_current];
     };

     var skipto = (char skipper) =>
     {
          int referenceCount = 1;
          while (referenceCount != 0)
          {
               next();
               if (current() == skipper)
               {
                    referenceCount++;
               }
               else if (current() ==
SkipperToCloserMap[skipper])
               {
                    referenceCount--;
               }

          }
     };

     var consume = (TokenType type, object? data) =>
     {
```

```
            _current++;
            return new Token(type, data);
      };

      if (!string.IsNullOrEmpty(str))
      {
            for (; _current < str.Length;)
            {
                  switch (current())
                  {
                        case '{':
                        skipto('{');
                        break;

                        case '(':
                        skipto('(');
                        break;

                        case '[':
                        skipto('[');
                        break;

                        case '$':
                        next(2);
                        break;

                        default:
                        {
                              if (char.IsNumber(current()) ||
current() == '.' || char.IsWhiteSpace(current()) ||
!char.IsLetter(current()))
                              {
                                    next();
                                    break;
                              }


                              StringBuilder sb = new();
                              while (current() != ' ')
```

```
                        {
                            sb.Append(current());
                            next();
                        }

                        yield return consume(
                            TokenType.NOTATION,
sb.ToString());

                        break;
                    }

                }
            }
        }
    }
}
```

Piece.cs

```
namespace Chess;

// Represents every possible piece type
public enum PieceType
{
    PAWN,
    KNIGHT,
    BISHOP,
    ROOK,
    QUEEN,
    KING
}

public class Piece
{
```

```
     public Piece(PieceType Type, Notation Location, Colour
Colour)
     {
     this.Type = Type;
     this.Location = Location;
     this.Colour = Colour;
     }

     public PieceType Type { get; set; }
     public Notation Location { get; set; }
     public Colour Colour { get; set; }

     // Moves the current piece to a square, `location`
     public void Move(SAN location)
     {
     // Check if piece on square | null if there isnt
     var pieceOnSquare =
MoveHelper.CurrentBoard!.Pieces.FirstOrDefault(x =>
x.Location == location.Square);

     // If the piece should capture or not
     if (pieceOnSquare is Piece p)
     {
          p.Remove();
     }

     // Move the piece
     Location = location.Square;
     // Redraw the current chessboard
     MoveHelper.CurrentBoard?.Invalidate();
     }

     // Get rid of the piece
     public void Remove()
     {
     MoveHelper.CurrentBoard?.Pieces.Remove(this);
     }

     // Pretty print the information ouht
```

```
    public override string ToString()
    {
    return $"{Colour} {Type} at {Location}";
    }
}
```

Puzzle.cs

```
using ChessMasterQuiz.Misc;

namespace Chess;

public class Puzzle : Question
{
    static Puzzle()
    {
    CreatePuzzles();
    }
    public Puzzle()
    {
    Setup = default;
    Moves = default;
    }
    public static List<Puzzle> Puzzles { get; set; } =
new();
    public override int Rating { get; init; }
    public Colour ToMove { get; set; }
    public IEnumerable<Piece>? Setup { get; set; }
    public List<string>? Moves { get; set; }
    public int CorrectMove { get; set; }
    public static void CreatePuzzles()
    {
    Puzzles = ReadPuzzles();
    }
}
```

SAN.cs

```
namespace Chess;

public class SAN : IEquatable<SAN>, IEquatable<Notation>
{
    // 0-0 0-0-0
    public int Castling { get; set; }

    // dxe5 Bxe5 Qxf7
    public bool Capturing { get; set; } = false;

    // Rf5+ Qxg6+
    public bool Check { get; set; } = false;

    // Qxf7#
    public bool CheckMate { get; set; } = false;

    // ( 'e', 4 ), ( 'f' 5 )
    public (char, int) PawnCapturing { get; set; }

    // PAWN, QUEEN, BISHOP
    public PieceType Piece { get; set; }

    // Rbf4, f5xg4
    public char? SpecifyPiece { get; set; }

    // f8=Q h1=Q
    public PieceType? IsQueening { get; set; } = null;

    // e4, d5
    public Notation Square { get; set; }

    // d3, g6
    public Notation InitialSquare { get; set; }
    public bool NullMove { get; set; } = false;
```

```
     // Print proper notation
     public override string ToString()
     {
     return GetNotation().Trim();
     }

     // Converts the instance into a standard notation string
     public string GetNotation()
     {
     if (Castling == 1)
     {
          return "O-O";
     }
     else if (Castling == -1)
     {
          return "O-O-O";
     }

     string ret = ${Piece.GetChar()}" +
     ${(SpecifyPiece is null ? "" : SpecifyPiece)}" +
     ${(Capturing && Piece == PieceType.PAWN ? (
          PawnCapturing.Item2 == -1 ?
${PawnCapturing.Item1}" : ${PawnCapturing.Item2}"
          ) : "")}" +
     ${(Capturing ? "x" : "")}" +
     ${Square.File}" +
     ${Square.Rank}" +
     ${(IsQueening == null ? "" :
$"={((PieceType)IsQueening).GetChar()}")}";

     return ret.Trim();

     }

     public SAN(string str)
     {
     From(str);
     }
```

```
    // Empty constructor
    public SAN()
    {
    }

    // Inspired by functional programming
    // Converts a SAN string into a SAN object
    public static SAN From(string str)
    {
    SAN san = new();

    if(str == "resigns" || str == "null")
    {
         san.NullMove = true;
         return san;
    }

    if (str.Count(x => x == 'O') == 2)
    {
         san.Castling = 1;
         return san;
    }
    else if (str.Count(x => x == 'O') == 3)
    {
         san.Castling = -1;
         return san;
    }

    if (str.Contains('x'))
    {
         san.Capturing = true;
    }

    if (str.Contains('+'))
    {
         san.Check = true;
    }
```

```
    if (str.Contains('#'))
    {
        san.Check = true;
        san.CheckMate = true;
    }

    // CharToNotation.TryGetValue(str[0], out PieceType p);
    if (char.IsUpper(str[0]))
    {
        san.Piece = str[0].GetPiece();
    }
    else if (san.Capturing)
    {
        san.PawnCapturing = (str[0], -1);
        san.Piece = PieceType.PAWN;
    }
    else
    {
        san.Piece = PieceType.PAWN;
    }

    int numLocation = str.IndexOfAny("12345678".ToArray());

    char file = str[numLocation - 1];
    int rank = Convert.ToInt16(str[numLocation].ToString());

    san.Square = Notation.From(file, rank);

    san.InitialSquare = san.GetInitialLocation();

    return san;
    }

    private static int FileNumberFromChar(char file)
    {
    return file switch
    {
        'a' => 1,
        'b' => 2,
```

```
            'c' => 3,
            'd' => 4,
            'e' => 5,
            'f' => 6,
            'g' => 7,
            'h' => 8,
            _ => -1
        };
        }

        // This finds out where a piece could have come from
based on a singlular SAN and a piece list
        public Notation GetInitialLocation()
        {
        int PAWN_DIRECTION;

        switch (Piece)
        {
            case PAWN:
                {
                    foreach (var pawn in
MoveHelper.CurrentBoard!.Pieces.Where(x => x.Type ==
PieceType.PAWN))
                    {
                    // Grab current pawn's file + the final
file
                    int pawnFile =
FileNumberFromChar(pawn.Location.File);
                    int destFile =
FileNumberFromChar(Square.File);

                    // If the pawn is white or black - moves
different direction
                    PAWN_DIRECTION = pawn.Colour ==
Colour.White ? -1 : 1;
                    if (pawn.Location.File != Square.File)
                    {
                        continue;
                    }
```

```
                    if (pawn.Location.Rank == Square.Rank +
PAWN_DIRECTION)
                    {
                         return pawn.Location;
                    }
                    // Pawn can only move two squares if it
is on it's inital square
                    else if (pawn.Location.Rank ==
Square.Rank + PAWN_DIRECTION * 2)
                    {
                         if (pawn.Colour == Colour.White)
                         {
                              if (pawn.Location.Rank != 2)
                              {
                                   continue;
                              }
                         }
                         else if (pawn.Colour ==
Colour.Black)
                         {
                              if (pawn.Location.Rank != 7)
                              {
                                   continue;
                              }
                         }
                    }

                    // Check all pawn one square moves
                    else if (pawnFile == destFile + 1 ||
pawnFile == destFile - 1)
                    {
                         if (pawn.Location.Rank +
PAWN_DIRECTION == Square.Rank)
                         {
                              return pawn.Location;
                         }
                    }
                    else
```

```
                    {
                        // No move found, try again on the
next pawn
                        continue;
                    }

                    return pawn.Location;
                    }

                    break;
                }

            case KNIGHT:
                foreach (var knight in
MoveHelper.CurrentBoard!.Pieces.Where(x => x.Type ==
PieceType.KNIGHT))
                {
                    int knightFile =
FileNumberFromChar(knight.Location.File);
                    int destFile =
FileNumberFromChar(Square.File);

                    // Knight Jumps
                    if (knightFile == destFile + 1 ||
knightFile == destFile - 1)
                    {
                    if (knight.Location.Rank == Square.Rank +
2 ||
                        knight.Location.Rank == Square.Rank
- 2)
                    {
                        return knight.Location;
                    }
                    }
                    else if (knightFile == destFile + 2 ||
knightFile == destFile - 2)
                    {
                    if (knight.Location.Rank == Square.Rank +
1 ||
```

```
                        knight.Location.Rank == Square.Rank
- 1)
                    {
                        return knight.Location;
                    }
                    }
                }
                break;

            case BISHOP:
                foreach (var bishop in
MoveHelper.CurrentBoard!.Pieces.Where(x => x.Type == BISHOP))
                {
                    int bishopFile =
FileNumberFromChar(bishop.Location.File);
                    int destFile =
FileNumberFromChar(Square.File);

                    for (int i = 1; i < 8; i++)
                    {
                    // Check in all four directions
                    if (bishopFile + i == destFile &&
bishop.Location.Rank + i == Square.Rank)
                    {
                        return bishop.Location;
                    }
                    else if (bishopFile - i == destFile &&
bishop.Location.Rank + i == Square.Rank)
                    {
                        return bishop.Location;
                    }
                    else if (bishopFile - i == destFile &&
bishop.Location.Rank - i == Square.Rank)
                    {
                        return bishop.Location;
                    }
                    else if (bishopFile + i == destFile &&
bishop.Location.Rank - i == Square.Rank)
                    {
```

```
                        return bishop.Location;
                    }
                    }
                }

                break;
            case QUEEN:
                foreach (var queen in
MoveHelper.CurrentBoard!.Pieces.Where(x => x.Type == QUEEN))
                {
                    int queenFile =
FileNumberFromChar(queen.Location.File);
                    int destFile =
FileNumberFromChar(Square.File);

                    for (int i = 1; i < 8; i++)
                    {
                    // Check in all four directions
                    if (queenFile + i == destFile &&
queen.Location.Rank + i == Square.Rank)
                    {
                        return queen.Location;
                    }
                    else if (queenFile - i == destFile &&
queen.Location.Rank + i == Square.Rank)
                    {
                        return queen.Location;
                    }
                    else if (queenFile - i == destFile &&
queen.Location.Rank - i == Square.Rank)
                    {
                        return queen.Location;
                    }
                    else if (queenFile + i == destFile &&
queen.Location.Rank - i == Square.Rank)
                    {
                        return queen.Location;
                    }
                    }
```

```
                }
                break;
        }

        return new();
        }


        public bool Equals(SAN? other)
        {
        return ToString() == other?.ToString();
        }

        public bool Equals(Notation other)
        {
        return GetNotation() == other.ToString();
        }
}
```

panelLeaderboardEntry.cs

```
namespace ChessMasterQuiz.CustomControls;

internal class panelLeaderboardEntry : Panel
{
        public User? user = null;
        public panelLeaderboardEntry(User u, int x, int y, int
width, int height, int index) : base()
        {
        Size = new Size(width, height / 5);
        user = u;
        int panelWidth = width / 5;
        int panelHeight = height / 5;

        List<Panel> panels = new();
        for (int i = 0; i < 5; i++)
```

```
     {
          Panel panel = new()
          {
               Size = new(panelWidth, 50),
               Location = new(x + (panelWidth * i), y +
(panelHeight * index))
          };

          panels.Add(panel);
     }
     panels.ForEach(panel =>
     {
          Controls.Add(panel);
          panel.Controls.Add(new Label());
          panel.Controls[0].Font = new
Font(FontFamily.GenericMonospace, 15);
          panel.Controls[0].Dock = DockStyle.Fill;
          (panel.Controls[0] as Label)!.TextAlign =
ContentAlignment.MiddleCenter;
          (panel.Controls[0] as Label)!.AutoSize = false;
     });

     panels[0].Controls[0].Text = $"{user.Username}";
     panels[1].Controls[0].Text = $"{user.Elo.Rating}";
     panels[2].Controls[0].Text = $"{user.Accuracy}";
     panels[3].Controls[0].Text = $"{user.HighScore}";
     panels[4].Controls[0].Text = $"{user.QuizesCompleted}";
     }

}
```

Key Algorithms

Rating System

The rating system in my application is modelled around the famous system for calculating relative skill levels of players, called the Elo Rating System. The Elo system was originally developed by Arpad Elo to improve the chess rating system.

For use in my application, I have modified the Elo rating system slightly for the user experience, by adding a constant value to the formula. This change results in the following formula of:

$$R = Ra + K * (Sa - Ea) + Sa * V$$

Where:
R -> resultant rating,
Ra -> initial player rating
K -> constant modifier
Sa -> outcome of the match
Ea -> expected outcome
V -> constant modifier.

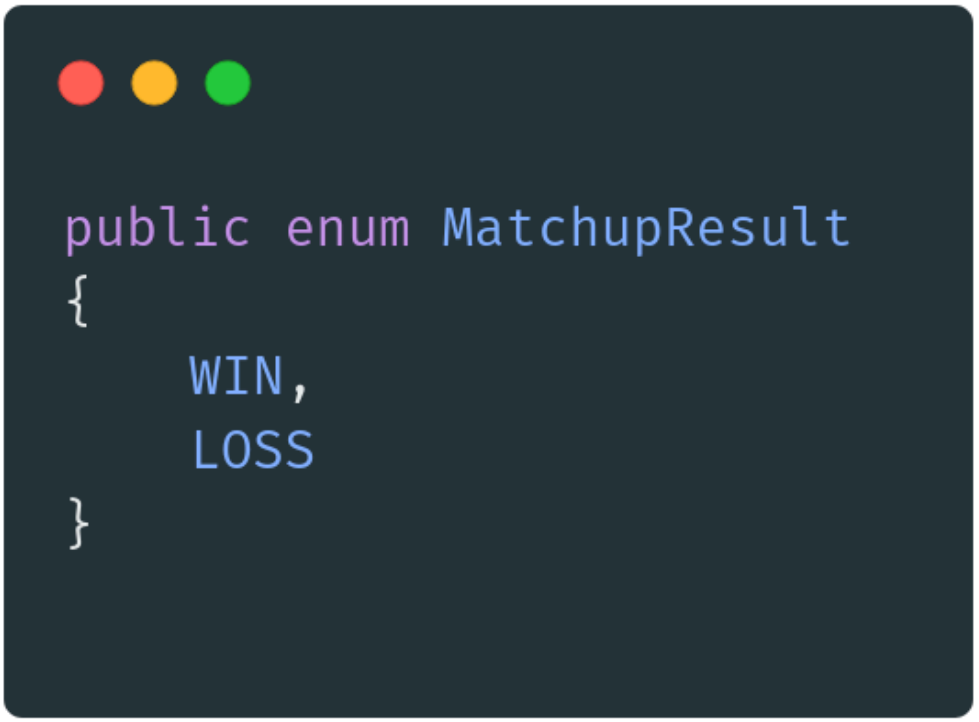
Implementation:

```
public static void Match(ref ELO elo, MatchupResult result, int opponentRating);
```

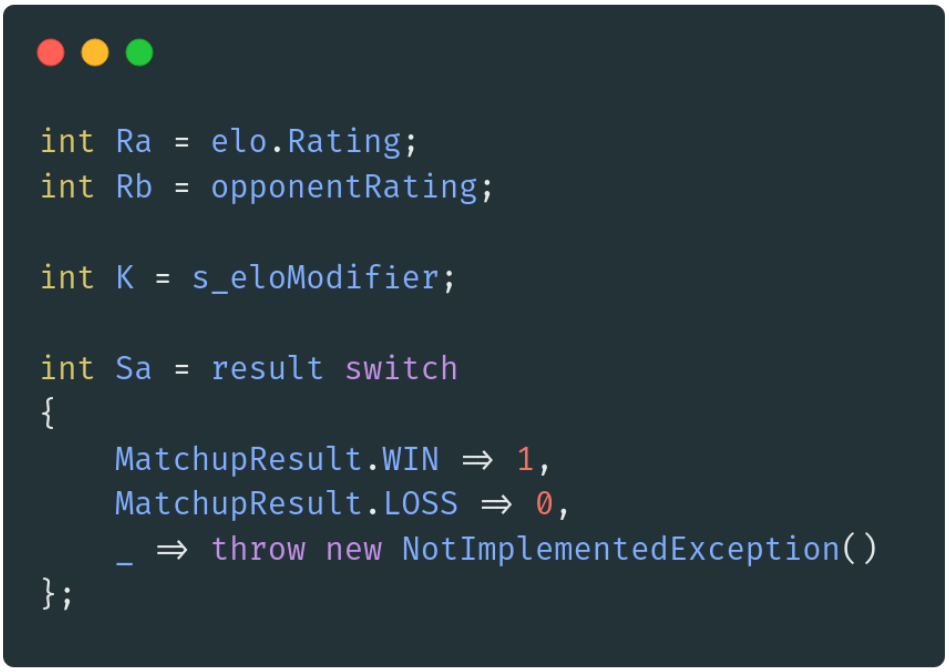
The static method `Match` is used to create a matchup between two ratings. In the case of the quiz, the opponent rating is derived from the difficulty rating of the question.

The elo passed through to the method must be a `ref`. This is due to the fact that all structs exist on the stack and therefore are value types, and cannot be modified in place. Using the `ref` keyword allows the method to modify the original elo passed through, and not a copy of it.

`MatchupResult` is an Enum, which can either be a `WIN` or `LOSS`



```
public enum MatchupResult
{
    WIN,
    LOSS
}
```



```
int Ra = elo.Rating;
int Rb = opponentRating;

int K = s_eloModifier;

int Sa = result switch
{
    MatchupResult.WIN ⇒ 1,
    MatchupResult.LOSS ⇒ 0,
    _ ⇒ throw new NotImplementedException()
};
```

The code above is the beginning of the implementation of the Elo rating system. The Initial rating of the player is accessible from a property in the `ELO` type.

The `result` variable is passed to the method as a `MatchupResult`, however the formula needs this as an Integer (Where 1 represents a win and 0 represents a loss). To fix this problem, I used a switch expression.

The switch expression is a much cleaner and concise version of the classical switch statement. The expression removes much of the boilerplate that plagued the switch statement (`case` etc). The expression also needs a default branch, which is denoted by an underscore.

The variable `K` is set to a field `s_eloModifier`. The practice of marking static private fields with an `s_` is very similar to Hungarian Notation, which involves preceding type names with the first letter of their respective types. For example, an Integer variable would be called iCounter.

This practice increases the readability of the codebase and helps the programmer to easily tell the difference between different fields with just a glance. For my implementation of the Elo rating system, this is set at `32`, which is a sensible starting point.

```
private readonly static int s_eloModifier = 32;
```

The two variables, `Qa` and `Qb`, are used to derive the expected outcome matchup. The formula looks like this:

$$Ea = \frac{Qa}{Qa + Qb}$$

$$Qa = 10^{\frac{Ra}{c}}$$

$$Qb = 10^{\frac{Rb}{c}}$$

I used the `Math` class provided by the `System` namespace. This class

```
float Qa = (int)Math.Pow(10, Ra / s_expectedOutimeCModifier);
float Qb = (int)Math.Pow(10, Rb / s_expectedOutimeCModifier);

float Ea = Qa / (Qa + Qb);

int addedBonus = Sa * s_bonusScore;

float resultantRating = Ra + K * (Sa - Ea) + addedBonus;
```

contains various methods for mathematical functions. The `Math.Pow(float x, float y)` method takes in two floats, so the resultant type of both `Qa` and `Qb` must also be a float.

After the expected outcome is calculated, the added bonus is only added if the matchup resulted in a win, this means that even if the player gets a high elo,

they will not result in lower bonus' which would therefore result in a stale progression through the leaderboard. Instead, the small but important bonus score keeps the progression exciting.

Finally, the `resultantRating` is calculated through the formula which can then be applied to the original `elo` object.

```
if(resultantRating < s_minimumPossibleElo)
{
    return;
}

elo.PastRatings.Add(elo.Rating);

elo.Rating = (int)resultantRating;
```

The final part of the method involves a check to ensure that the elo cannot drop below the minimum possible. This elo is around ~150 points. Ultimately, the initial rating is added to a list of previous ratings and then the rating is updated based on the matchup.

This modification of the Elo algorithm is an essential part of my application due to its impact on gameplay. The questions presented to the user are directly related to the rating of the player. In addition, the rating of the player is the main factor in determining the ranking of the leaderboards, and players with a higher rating will progress through the leaderboards with a higher rate than those with a low rating.

Context System

A problem that I encountered early on in the development of the application was the issue of passing information around from form to form. I had three concerns that needed to be addressed in my solution.

1. Speed

2. Simplicity
3. Modular

I designed a solution which would address all of the above problems. The solution was called the `Context` system. The system was created as an extension of the pre-existing Form management system.

Before the Context system, I was using a “master” panel, which I could then display forms using a `DisplayForm(Form form)` method.
While this did work, a major problem was that it required the consumer of the method to create the form themselves, resulting in a long method call. Another problem was calling the method required a reference to the current form being displayed, which resulted in a lot of unnecessary code.

```
public void DisplayForm(Form form) {
    panel.Controls.Add(form);
    // Other configuration here
}
```

```
// Form has a static reference to the current form
Form currentForm = Form.ActiveForm;
if(currentForm is null) {
  // Handle error
}

// Cast form to correct type
formMain main = (formMain) currentForm;

// Call the method
main.DisplayForm( new formMenu() );
```

Above is the code needed to display a new form to the user. This is a lot of code for such a simple task. You could refactor this code into a one-liner, shown below, but this makes some assumptions about the nullability of the ActiveForm reference and is generally not considered good practice.

```
(ActiveForm as formMain)!.DisplayForm(new formMenu());
```

In addition to the subjective ugliness of the code, there is a major issue prominent. Let's say that you need to pass some user data from one form to another. I didn't want to use hacked together solutions involving `public static` variables when not needed as this would go against basic encapsulation principles, and would not be a good solution in the long run due to its lack of modularity.

Instead, I decided to design a system which could pass any data through to any form needed. While this necessarily wouldn't be difficult to implement naively, I wanted to ensure that my goals of speed, simplicity and modularity were maintained throughout the application. To begin the process, I decided to rewrite the code to display a form to the screen, where I ended up with this:

```
public static void ActivateForm<T>() where T : Form, new()
{
    // Create the form
    formMain.ChildForm = CreateForm<T>();

    // Configuration here

    // Grab the reference to the form
    var holder = formMain.GetPanelHolder!();

    holder.Controls.Clear();
    holder.Controls.Add(formMain.ChildForm);
    holder.Show();
}
```

The code above uses the Generic Type system in C#. In short, this allows any `Type` to be passed to the method. However, this type is constrained to some parameters. The type MUST inherit from the base `Form` class in some way, and it must have an empty constructor. These two constraints allow the method to only accept suitable forms.

Another point to note is that no object references are passed through to the form. This greatly reduces the amount of boilerplate required to create a new form. One issue with this approach is that you cannot create an object from a Type at runtime, without reflection that is.

Reflection can be extremely useful in some niche situations. In addition to other use cases, it allows you to gain information about your types at runtime. For example, viewing the properties of an unknown object at runtime.

To actually call this method, the consumer must use angle brackets, such as the following:

For our purposes, we can use reflection to create an instance of our unknown form subtype at runtime.

Onto the arguably the most important part of the context system, which is the `CreateForm` method.

```
public static Form CreateForm<T>(IEnumerable<DataContextTag> context) where T : Form, new()
{
    // Create the Form
    var instance = Activator.CreateInstance<T>()!;

    // Grab the method from the interface
    var method = typeof(IContext).GetMethod("UseContext");

    // If the method exists and if the form impl the interface
    if (method is not null && typeof(IContext).IsAssignableFrom(typeof(T)))
    {
        method!.Invoke(instance, new object[] { context });
    }

    // Return the form
    return instance;
}
```

This method is the glue that holds the context system together. It is responsible for creating the instance of the form, and passing the context through to the instance. The `Activator` class is accessible from the `System` namespace and is the class that does most of the heavy lifting for us, by returning an instance of the type.

For the most modularity, I decided that each class which will consume the context should have a method called `UseContext`, and the easiest way to ensure that they do implement the method is through an interface. The Interface, called `IContext`, contains the method of UseContext.

Using reflection, we can actually grab an object representing the method, which can then be called later. This method object contains all of the data around the method and needs an object reference of the appropriate type to be called.

The next step is to check if the method exists, and if the generic type `T` actually implements the interface. If not, then we can totally just ignore the whole context system and return the newly created form. However, if it does implement the interface, the method is invoked on the instance, with the context passed through.

Finally, the instance is returned back to the consumer.

One major piece of the puzzle that I have ignored up to now is the `DataContextTag` type. This type is effectively a wrapper for a Key Value Pair.

```
public static void ActivateForm<T>(params DataContextTag[] context) where T : Form, new()
{
    // Create the form
    formMain.ChildForm = CreateForm<T>(context);

    // Configuration here

    // Grab the reference to the form
    var holder = formMain.GetPanelHolder!();

    holder.Controls.Clear();
    holder.Controls.Add(formMain.ChildForm);
    holder.Show();
}
```

The record is a reference type that contains an inline constructor. The `data` can be anything that inherits from object, (so everything) and the `tag` can be any option from the Enum `ContextTagType`.
The DataContextTag is a record type, which means that it auto implements the `Deconstruct` method, allowing a consumer to break down the type into a tuple. These added benefits allow the DCT to be effective in moving data around the application.

The `ActivateForm<T>()` method should be rewritten to accommodate the DCT:

This is the final version of the ActivateForm method. The `params` keyword allows for any amount of DCT to be passed through to the method, including zero, which means no parameters may be passed through.

To call the method now, you can either use the same syntax as before, ignoring the DCT, or you can pass through as many as you want:

```
public class formMenu : Form, IContext {

  public void UseContext(IEnumerable<DataContextTag> context) {
        string username_data = context.First(dct => dct.tag == USERNAME);
  }

}
```

The last step is to handle the context in the consumer:

Below is an example of using LINQ to get some data about a username.

I believe that I have accomplished my three initial requirements:

Speed
Creating new forms which use the context system only requires the consumer to implement one interface, and the system will work automatically. Secondly, creating new instances of forms does not require the consumer to create them

```
ActivateForm<formMenu>(new ("autocomplete-username", USERNAME) );
```

themselves. Finally, Creating DCT tags are quick, and can easily be deconstructed into their separate parts.

Simplicity
Removing most of the unnecessary boilerplate from the original code greatly improves the readability of the code, and using the context system, at a base level, does not require great knowledge of the inner workings.

Modularity

The context system is greatly modular as any data can be passed to any compliant form, without any restrictions. In addition, the context system can be applied to any type of form.

Serialisation

When designing the chess system inside my application, I had a problem. I needed a way to read in chess games and positions from files into useful c# objects. While I could have used a third-party NuGet package, such as pgn.net, I decided it would be a better idea to create my own serializer for the PGN file format. If you are unsure what a PGN is, an example can be found here.

Having experience building programming languages from books, such as Crafting Interpreters by Robert Nystrom, and Building An Interpreter In Go by Thorsten Ball, I believed that I could successfully create not just a functional, but also fast serializer. A key part to creating a programming language is called the Tokeniser. The tokenizer is responsible for turning plain text files into “tokens”. These tokens can then be used to craft the Abstract Syntax Tree and so on. For the purposes of the serializer, we only need a system to create these tokens, and then turn the tokens into C# objects.

This custom serializer would be made of two parts, the Lexer and the Parser. The lexer is responsible for the Lexical Analysis of the source file, which means that it will look at the PGN file and turn everything into tokens. These tokens are represented by a `record struct` called `Token`.

```
record struct Token(TokenType Type, object? Data)
{
    public override string ToString() ⇒
        $"{Type}: {Data ?? ""}";
}
```

Each token has a `Type`, which is an enum which contains all possible types of tokens that the Lexer could encounter. Each token can also have some data, depending on the Type. Due to the simplistic nature of the PGN file format, I decided to go with a Key Value system.

Now that we have the internal data structures for our Lexer, we can begin

```
enum TokenType
{
    KEY,
    VALUE,
    NOTATION,
    LEFT_BRACKET,
    RIGHT_BRACKET,
    LEFT_CURLY,
    RIGHT_CURLY
}
```

lexical analysis. (Lexing!). The Lex function takes in a string, which represents the file. The main idea behind the Lexer is that we can traverse over this string and gradually convert parts of it to tokens.

While the code for performing lexical analysis is extremely similar for both the metadata portion of the PGN and the actual moves portion of the PGN, it is much easier to show just the metadata portion.

The first thing we need to do is to declare some inline helper functions. These are extremely important for traversing over the string while keeping the readability of the code.

```
int _current = 0;

var next = (int n) => _current += n;
var peek = () => str[_current++];
var current = () => str[_current];

var consume = (TokenType type, object? data = null) =>
{
    _current++;
    return new Token(type, data)
};
```

The `_current` int is a pointer to the current character of the string that we are inspecting. Depending on the next characters, the token type might change. All three of the `next`, `peek` and `current` functions are self explanatory, they either change the current pointer, or they return a character from the string.

The `consume` function may be the most important, as it deals with the instantiation of a new `Token`. As an added bonus, it also increments the `_current` reference in preparation for the next character to be analysed.

```
for(; _current < str.Length;)
{
    switch(current())
    {

        case '[':
            yield return consume(TokenType.LEFT_BRACKET);
            break;
```

This snippet above is the bread and butter of the Lexer. Initially, we create an unusual for loop, which is missing its declarator and its incrementor. We don't need these parts as we will be manually controlling the `_current` reference. We then perform a check on the current character under the `_current` reference. If it's any char such as a `'['`, we can consume the token as it is a single character token. But what if the token is multiple characters long?

```
case '"':
    {

        StringBuilder sb = new();
        next(1);
        while(current() ≠ '"')
        {
            sb.Append(current());
            next(1);
        }

        yield return consume(
            TokenType.VALUE,
            sb.ToString()
            );

        break;
    }
```

This case handles all values of the metadata portion. All values must be in quotation marks which makes it extremely easy to parse into a separate string. A string builder is used as many string concatenations will occur and creating many strings is unnecessary. A simple while loop keeps adding the current character, and then moving onto the next. This loop will only stop when the next quotation mark is found.

Finally, the token should be consumed, but this time the new string which has just been created should be passed through as the data. This is important in the parsing step later.

The same process is used to tokenize the keys, but no quotation marks are present so added precautions must be taken.

Once the loop is finished, all of the string will have been converted into a list of tokens. These tokens can then be fed into the second step of the deserializer, the `Parser`. The parser is important as it takes the tokens and turns them into real objects.

```
PropertyInfo? property = props.FirstOrDefault(x ⇒ x.Name == (string)meta[i].Data!);
```

Here, reflection is used to find the correct property. Once we have the correct property, we first check for null and then we can convert the `object?` Type to the correct type the property expects. If the type implements the interface, `IConvertable`, it can be used in the `Convert.ChangeType()` method.

```
var data = Convert.ChangeType(meta[i + 1].Data, prop.PropertyType);
prop.SetMethod?.Invoke(pgn, new object[] { data! });
```

In the code above, we first ensure that we have the correct type for the parameter, and then we call the set method associated with the property. The first parameter is the object that the property method should be called onto, and the second parameter is an array of parameters that should be passed into the method. This use of reflection allows the parsing of the PGN metadata into the C# object, `PGN`.

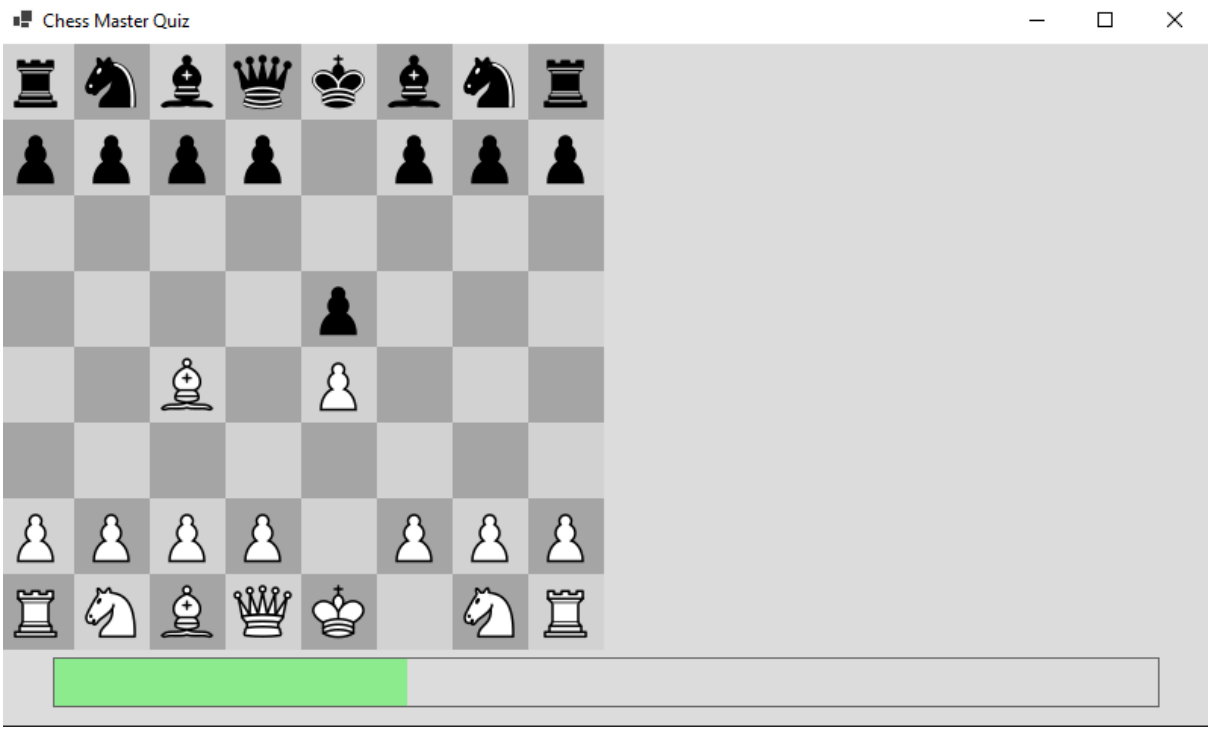
To parse the actual moves, a simple for loop is used to iterate through all of the tokens and convert them into a `SAN` object, or (Standard Algebraic Notation). This SAN object can then be added to the final PGN object and the *.pgn files have been successfully serialised.

Overall, I believe that my serialisation system was robust enough to handle almost all PGN files. This allowed me to simply import the file rather than having to program the moves in by hand.

These 3 algorithms were key to the success of my application. While creating the application was fully possible without them, they made the development process much more enjoyable and allowed me to automate a lot of the boring boilerplate code that is so commonly associated with these systems.

Evidence of Testing

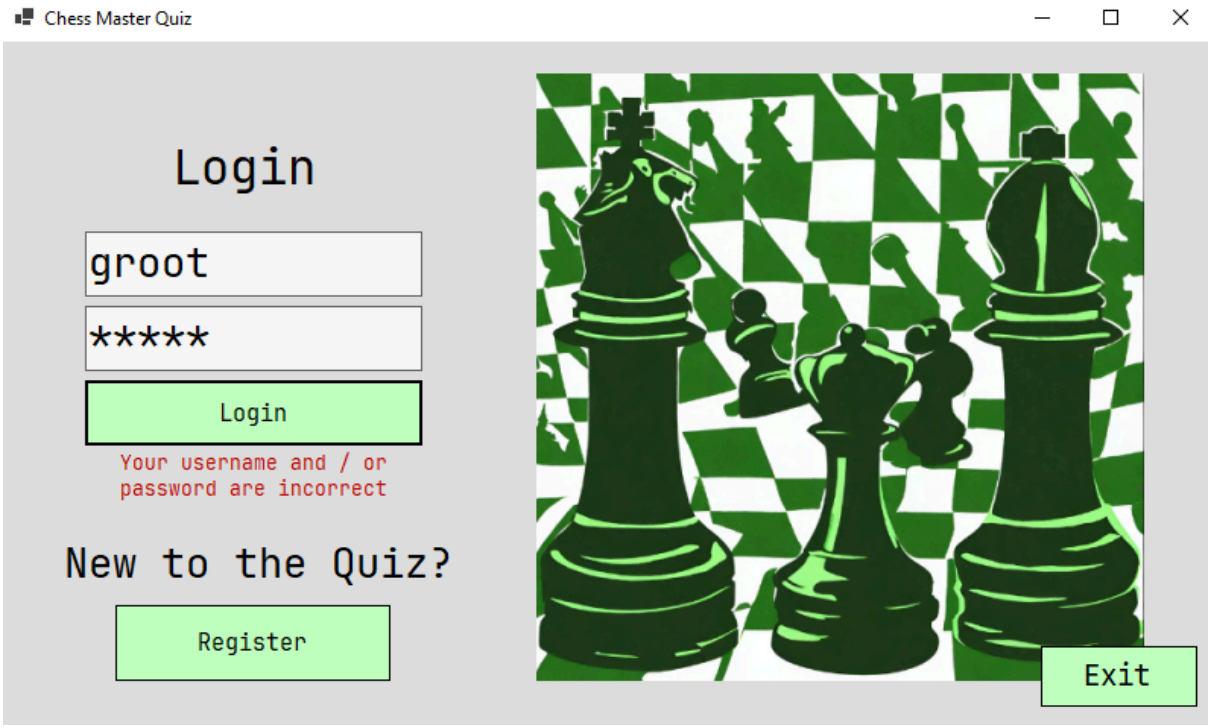
Test 1.03 Failure



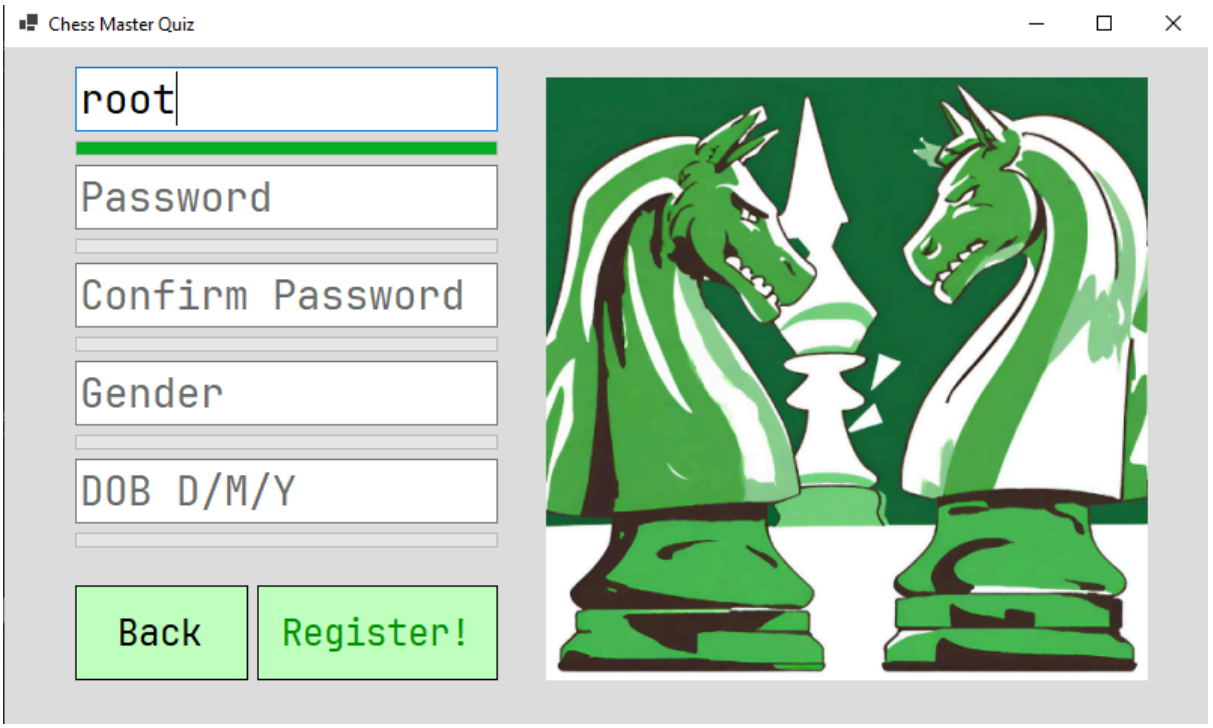
Test 1.03 Success

Test 2.02 Failure

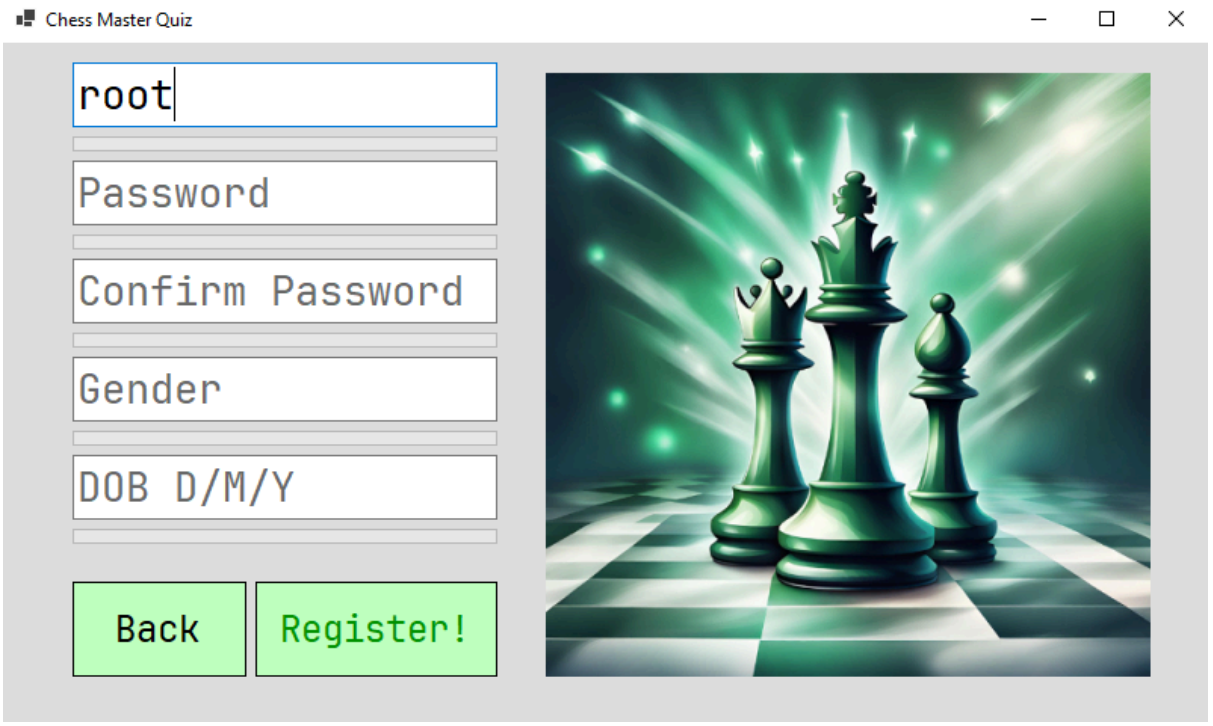
Test 2.02 Success



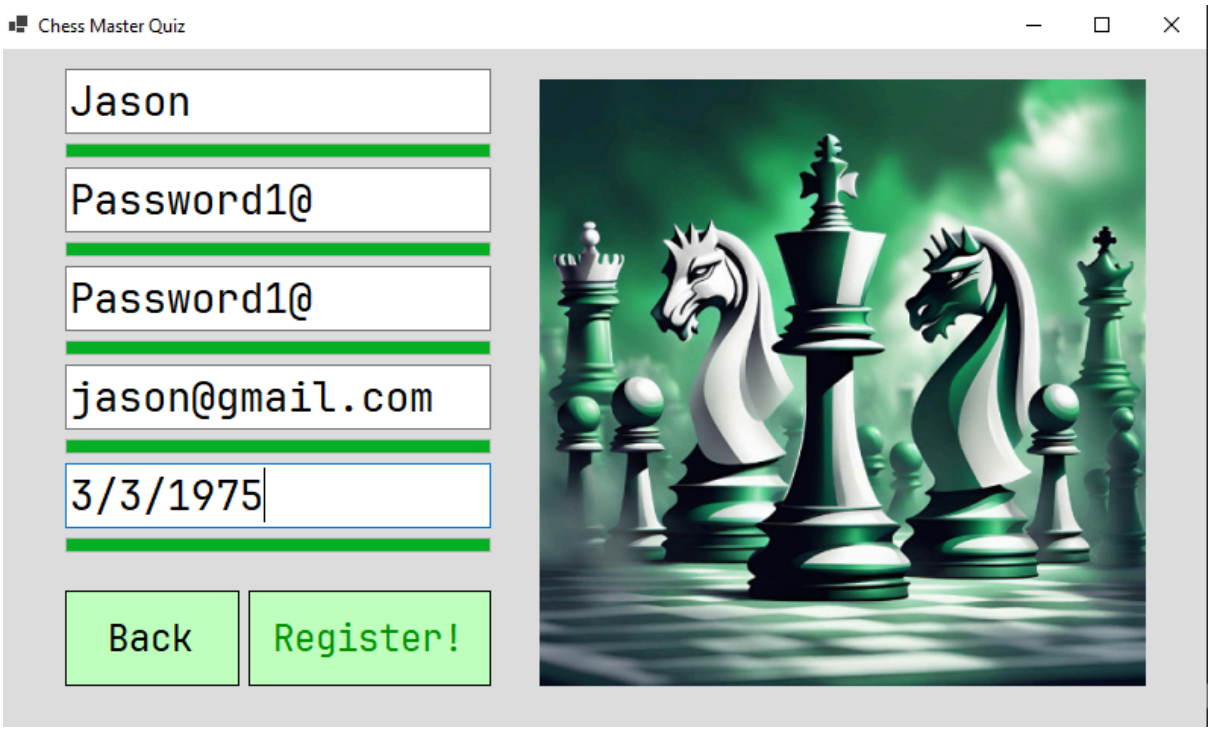
Test 3.01 Failure



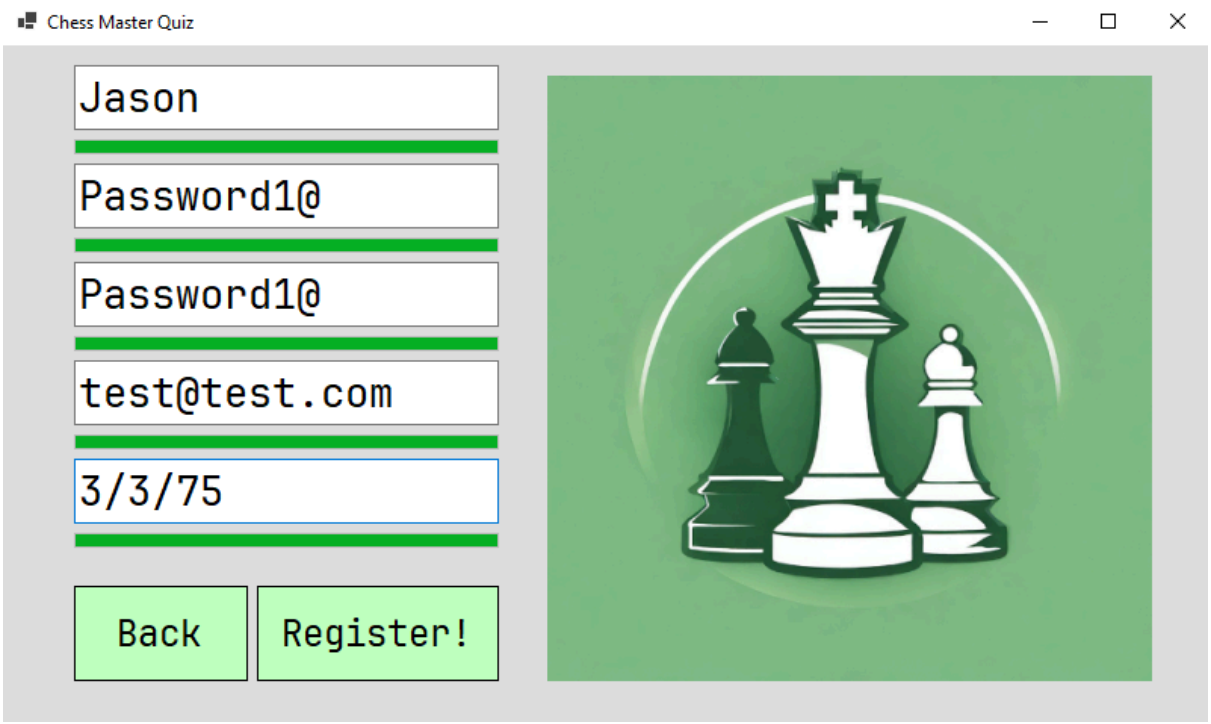
Test 3.01 Success



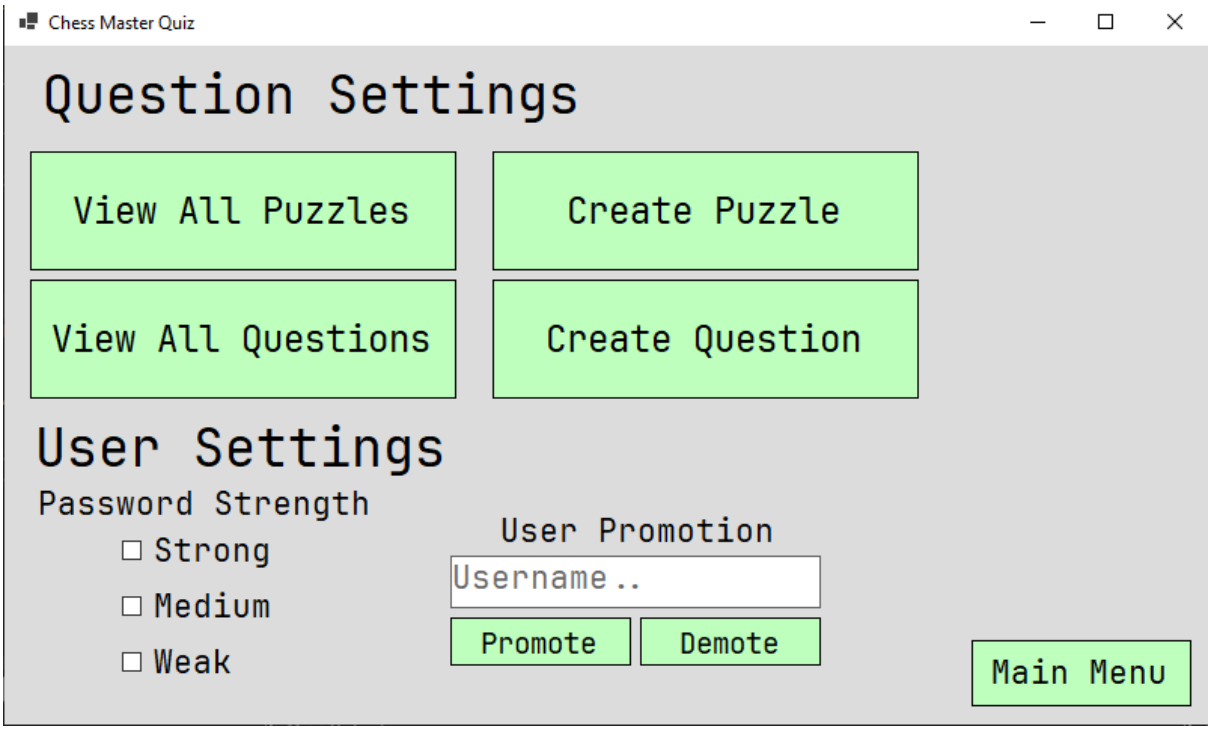
Test 3.11 Failure



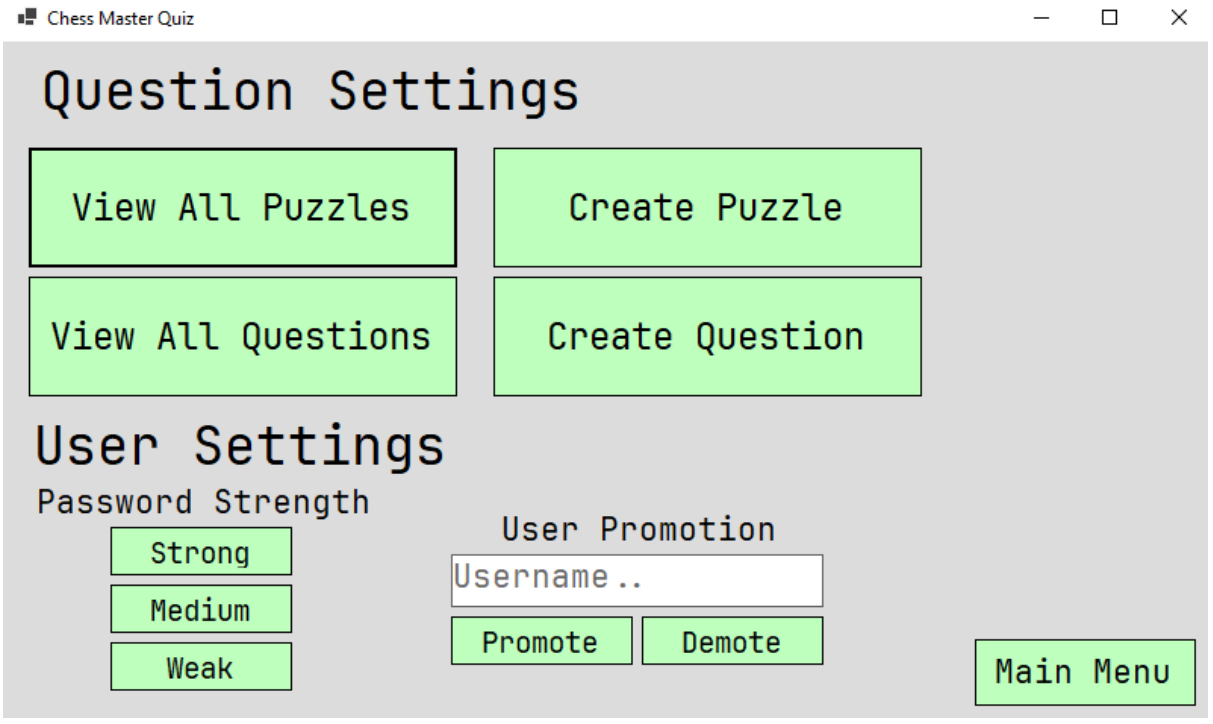
Test 3.11 Success



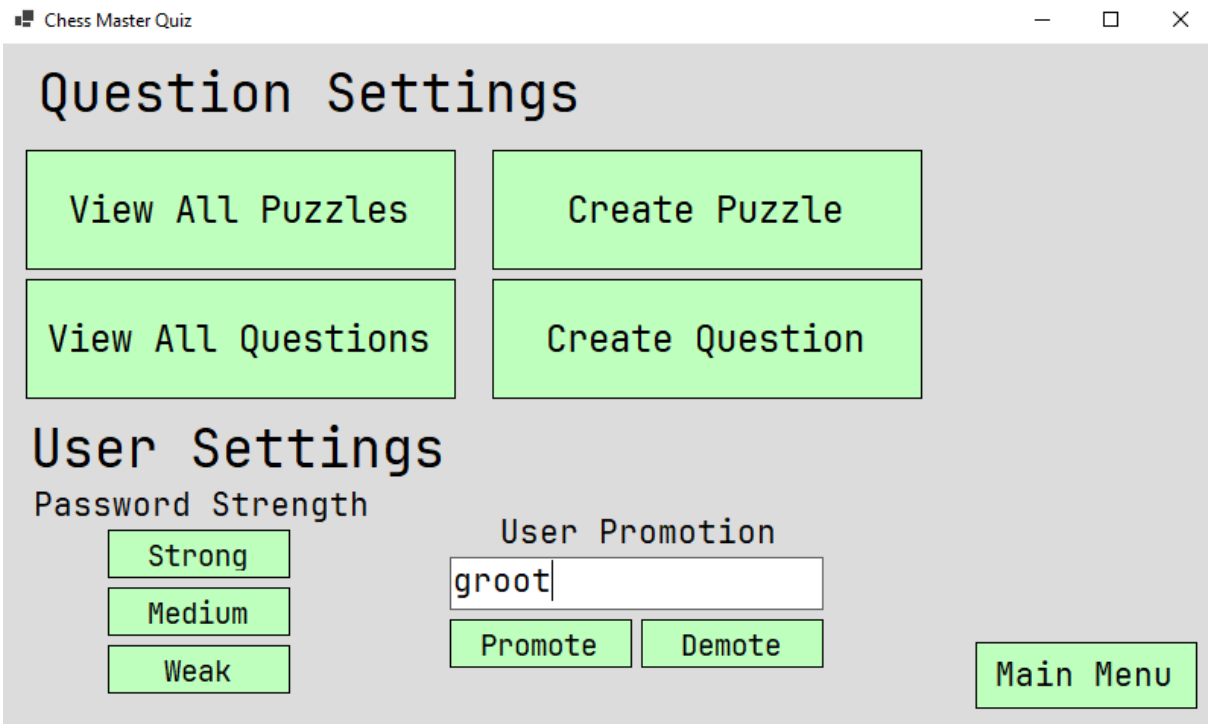
Test 10.04 Failure



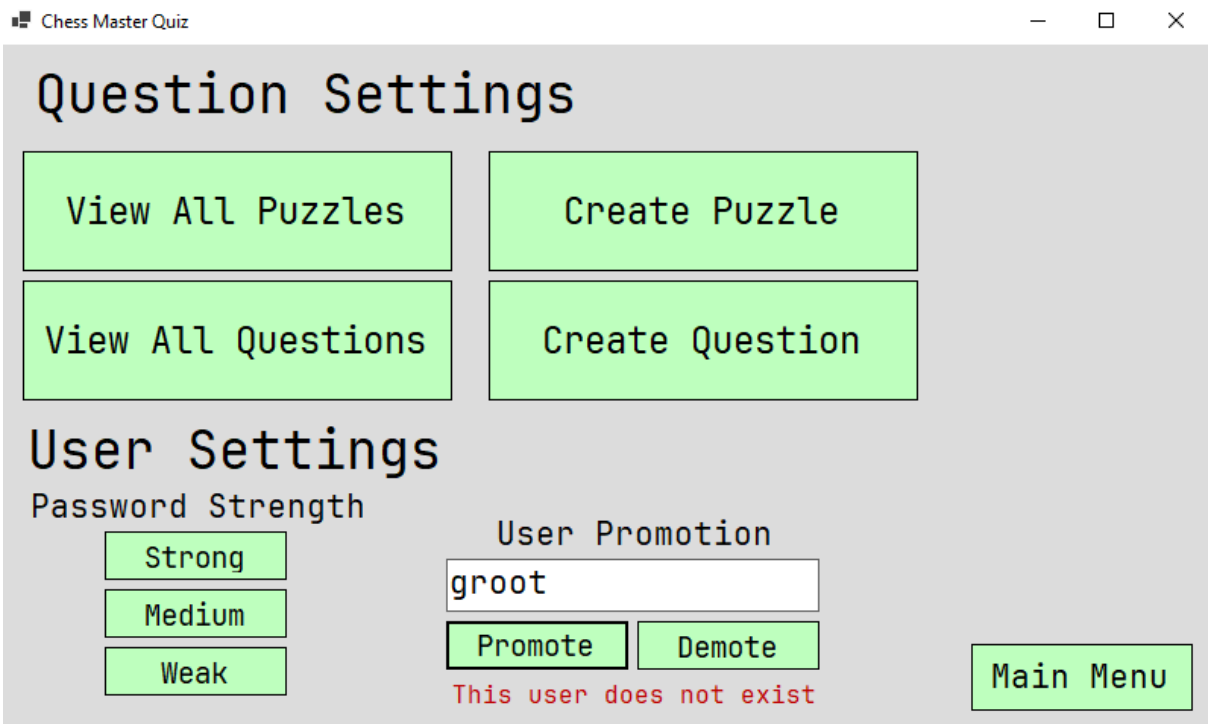
Test 10.04 Success



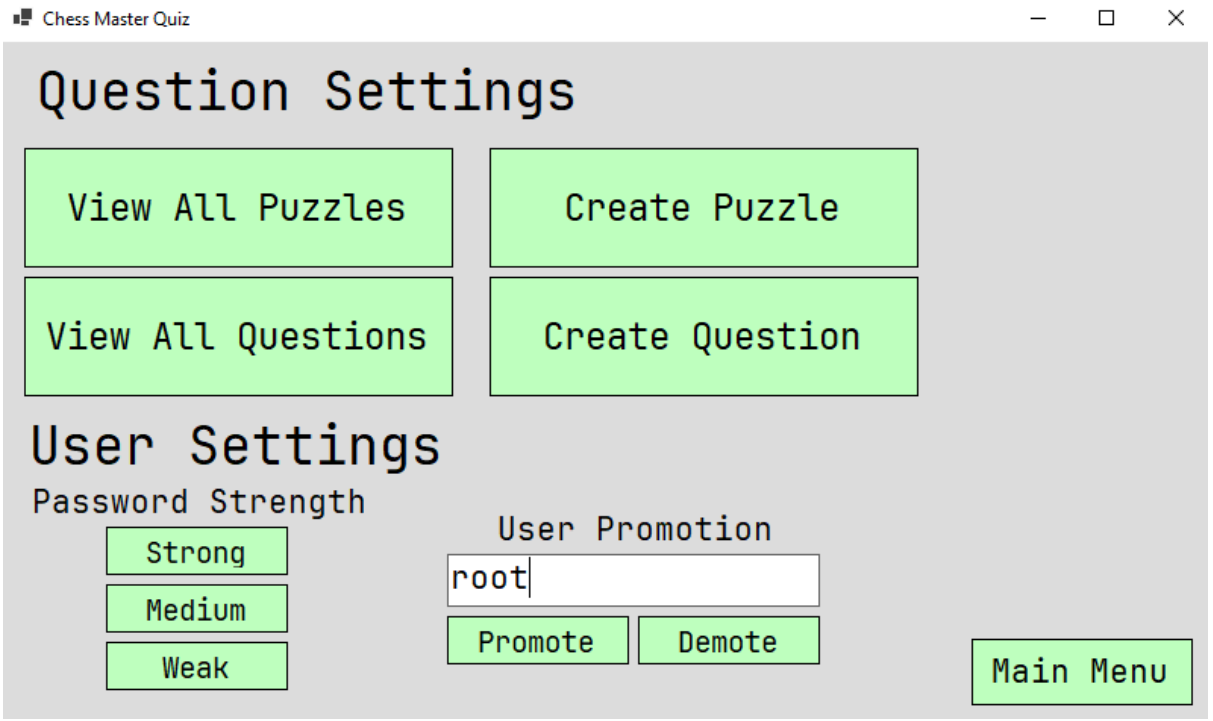
Test 10.08 Failure



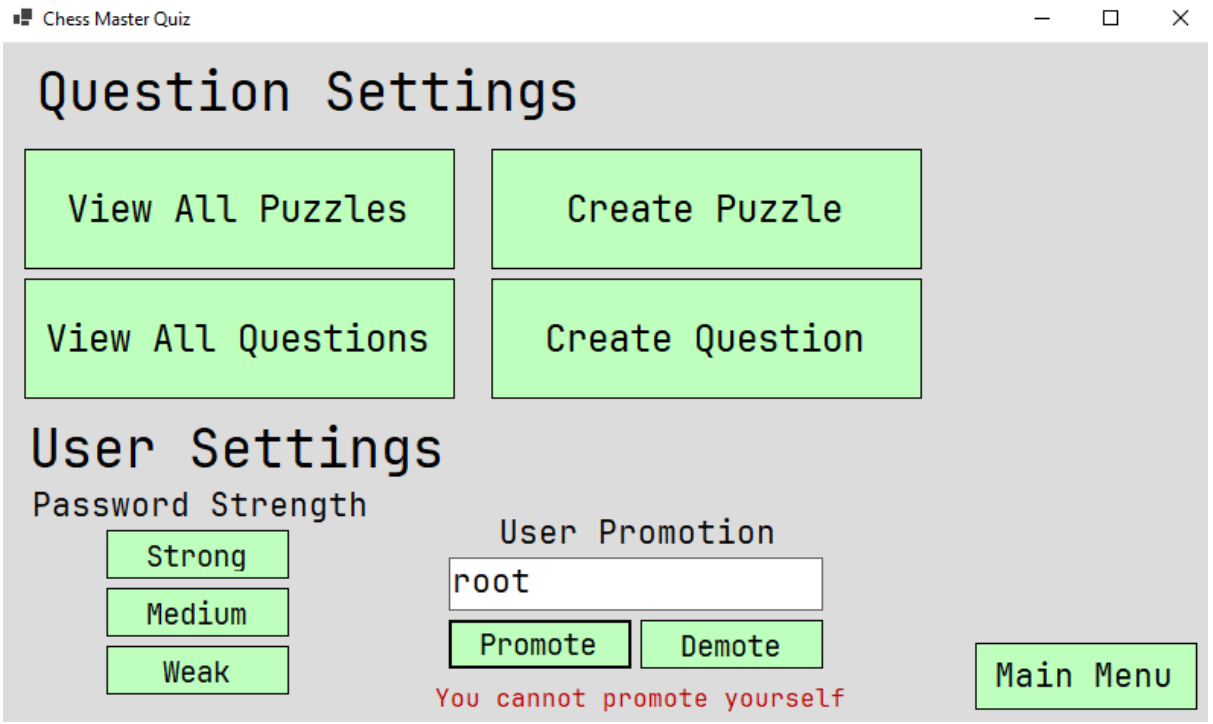
Test 10.08 Success



Test 10.09 Failure

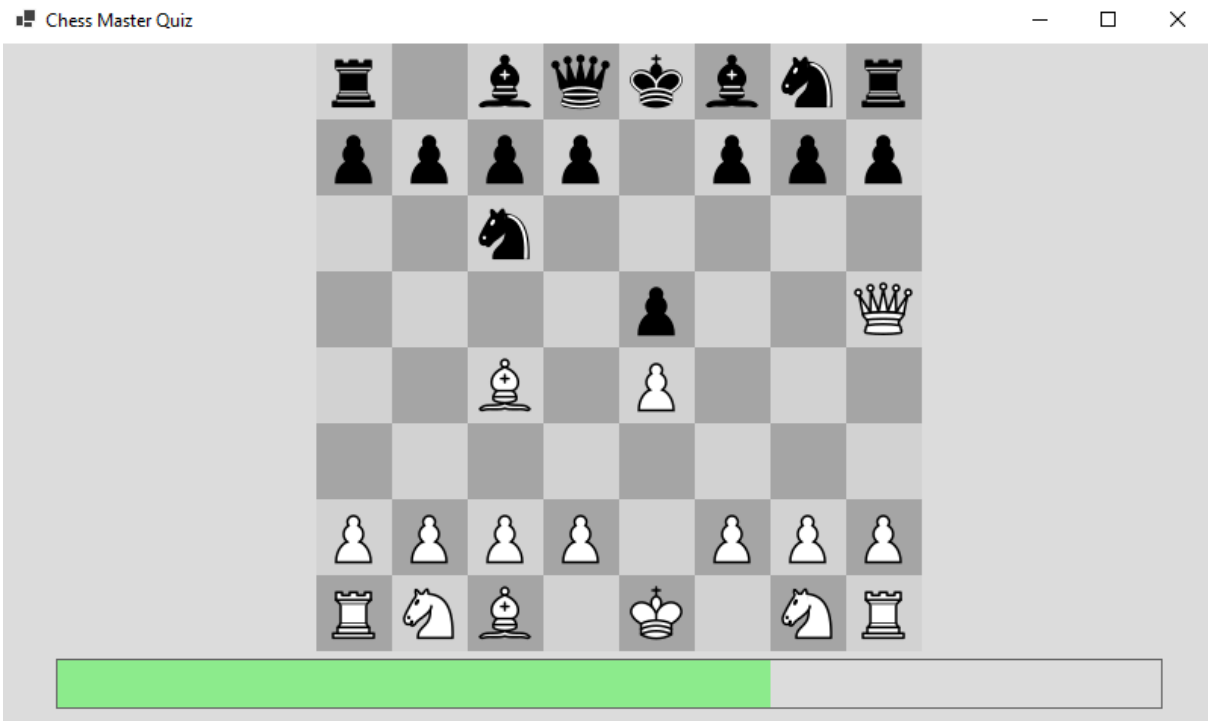


Test 10.09 Success



Application Walkthrough

Splash Screen - `formSplashScreen.cs`



The splash screen is the first form displayed to the user. It takes approximately 3 seconds to finish loading. The splash screen also includes a short chess game while the progress bar is loading. This chess game is a well-known opening called the *Scholar's Mate.*

These features have been implemented through the use of a custom chess board panel, and a secondary panel for the progress bar. I used asynchronous programming to tackle the task of loading the next form when the progress bar completes. Due to the nature of how I designed my form management system, the system is unable to open a form from another thread than the one it was created on. This poses a problem as the timer for the progress bar creates a

new thread to handle its `Elapsed` calls. Asynchronous programming allows the same thread to wait for a period of time, while keeping the chessboard and progress bar moving synchronously.

When the progress bar finishes loading, the next form, *formLogin* will be displayed, which allows the user to either login to the application, or register instead.

Login Form - `formLogin.cs`

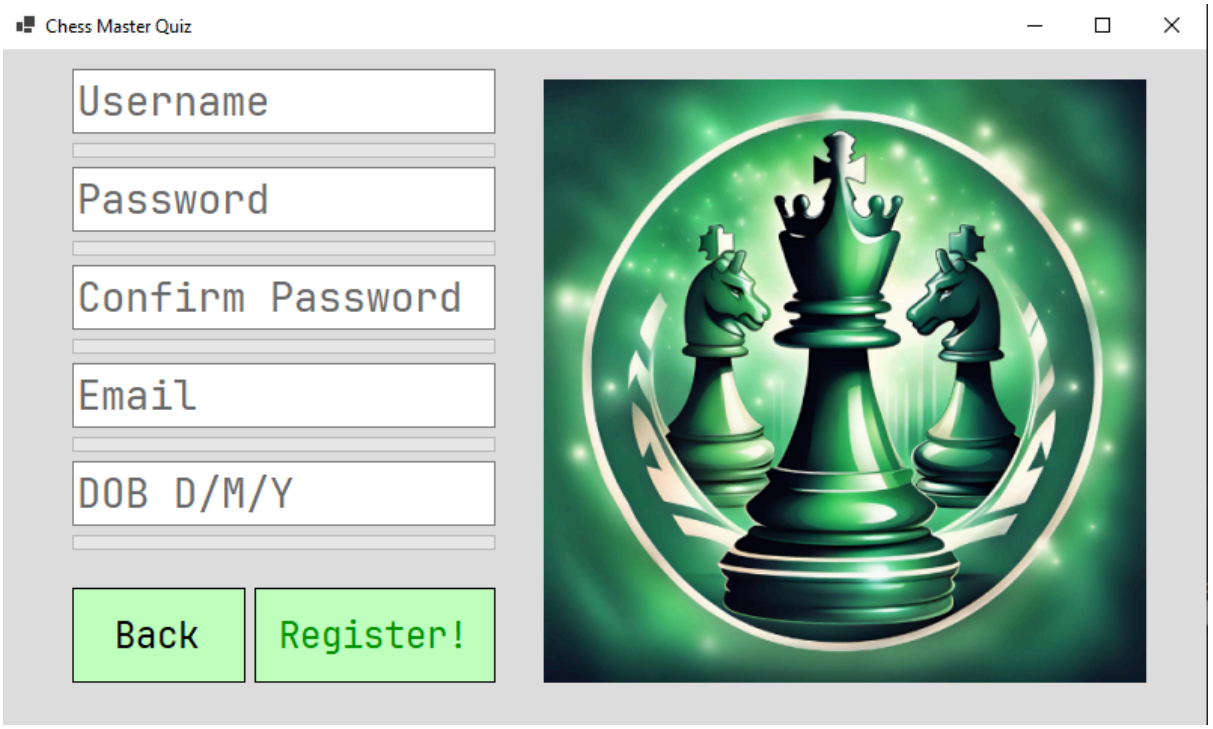
The login form is the second form displayed to the user. Here the user is able to sign into a preexisting account, or alternatively create a new account by navigating to the registration page. The user can also leave the entire application here by clicking on the `Exit` button.

A prominent part of the login form is the large logo on the right side. This logo is randomly chosen every time the form is loaded. In total there are about 35 different possible logos that could be displayed.

When the user enters both their username and password, the username is first checked against the list of users. If the username doesn't match any, then

this is displayed to the user through a label. However, if it does match, then the password field is hashed and checked against the hashed password of the user. If the hashed password matches, then both of the details are correct and the user will be redirected to the *Main Menu*. Behind the scenes, the user is then logged in and set to be the `ActiveUser`. All forms can then reference this `ActiveUser`, or their behaviour can be changed to fit a specific user which can be passed from the *Context System.*

Register Form - `formRegister.cs`

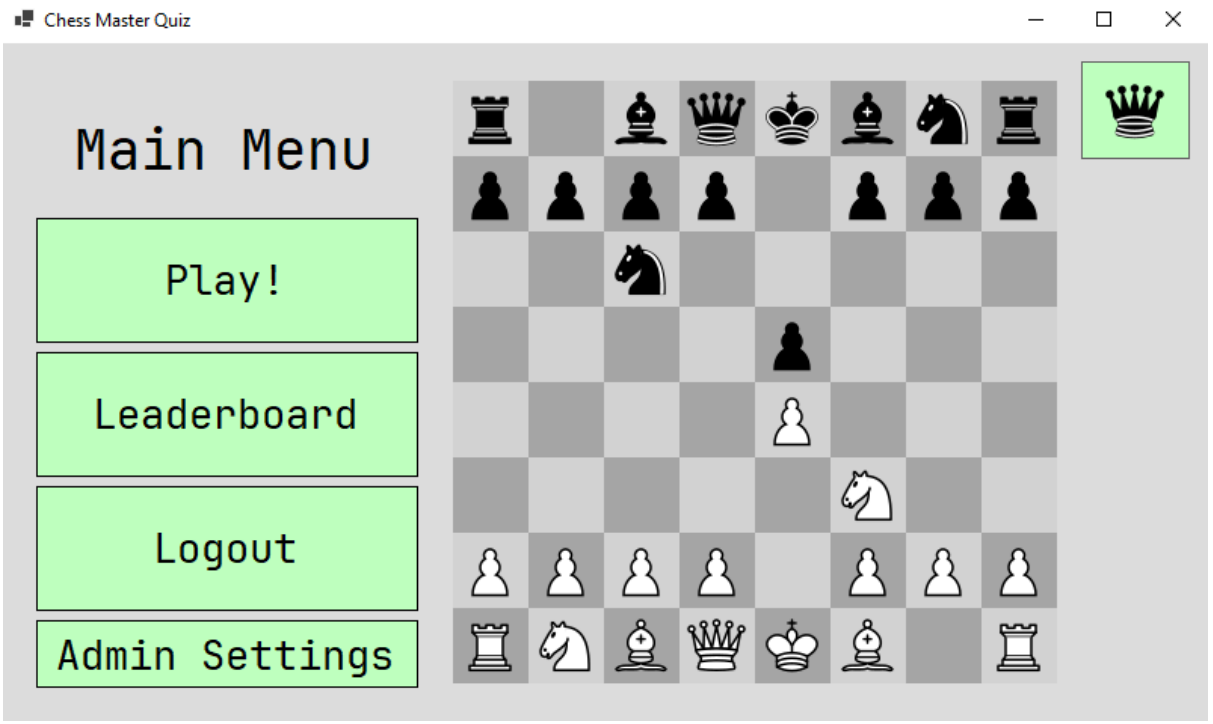


The Registration form can be accessed through the login form. Here, users are able to enter their information and sign up to the quiz. The user will then be able to *register* and create their new account. The user will only be able to register if all fields are validated properly. A progress bar under the appropriate fields will show the user how close they are to completing the respective validation check. For example, a username must be longer than 3 characters, and less than 18. It also cannot contain special characters or spaces, etc. To validate these fields, a mixture of regex expressions and hard coded checks are used.

A design decision I made was to prevent the user from choosing to become an admin in the registration process. Instead, a preexisting admin can promote a normal user into an admin.

In addition, admin users can also change the password requirement strength. A *weak* requirement means that passwords do not need much validation, but a *strong* requirement means that passwords need excessive validation. A good middle ground is the *medium* requirement.

Main Menu - `formMenu.cs`

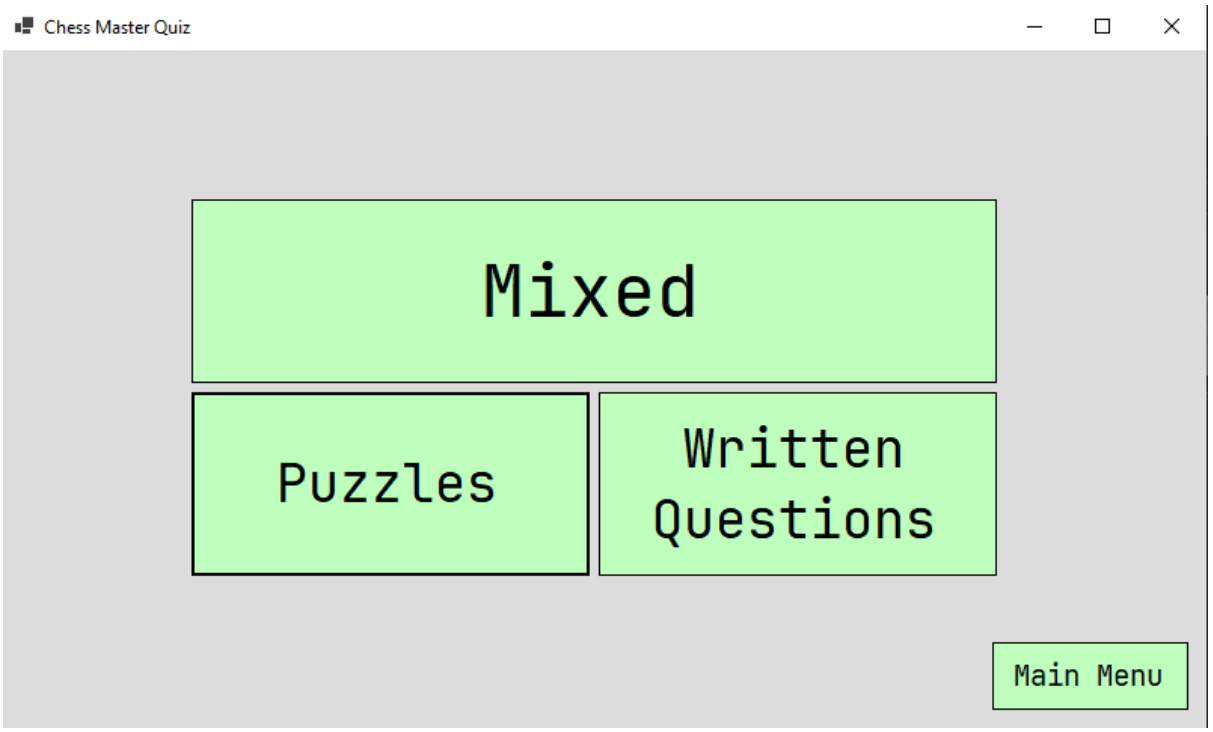


The main menu is the heart of the application. It provides quick paths to every major part of the quiz. These quick links include, the *Play* menu, the *Leaderboard*, the *Admin Menu*, the *User Profile*, and Logging out of the application. Pressing on any of these buttons will simply activate the form while passing any necessary state (context system) to the representative form.

A large part of the main menu, much like the *Splash Screen*, is the chess board that plays a famous game by itself. The games that could be played on the board are famous games from the chess player, *Wilhelm Steinitz*.

Clicking on the user's profile picture at the top right of the screen will open the *User Profile* of the currently logged in user.

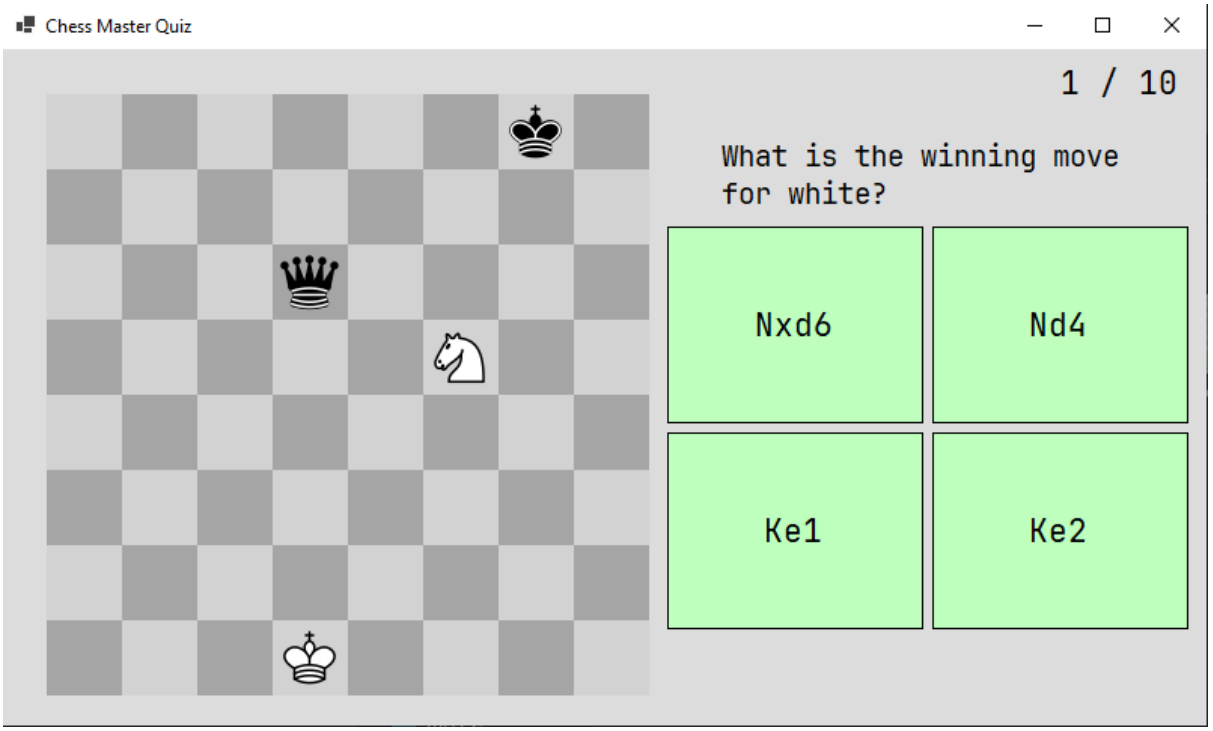
Choose Question Menu - `formChooseQuestion.cs`



This form allows the user to choose a type of question to answer. Each of the buttons, Mixed, Puzzles and Written Questions redirect the user to the representative form. The user can also return to the main menu if they want to.

When the user chooses a quiz type to proceed with, a callback function is generated. This callback function is then passed to the form with the context system.

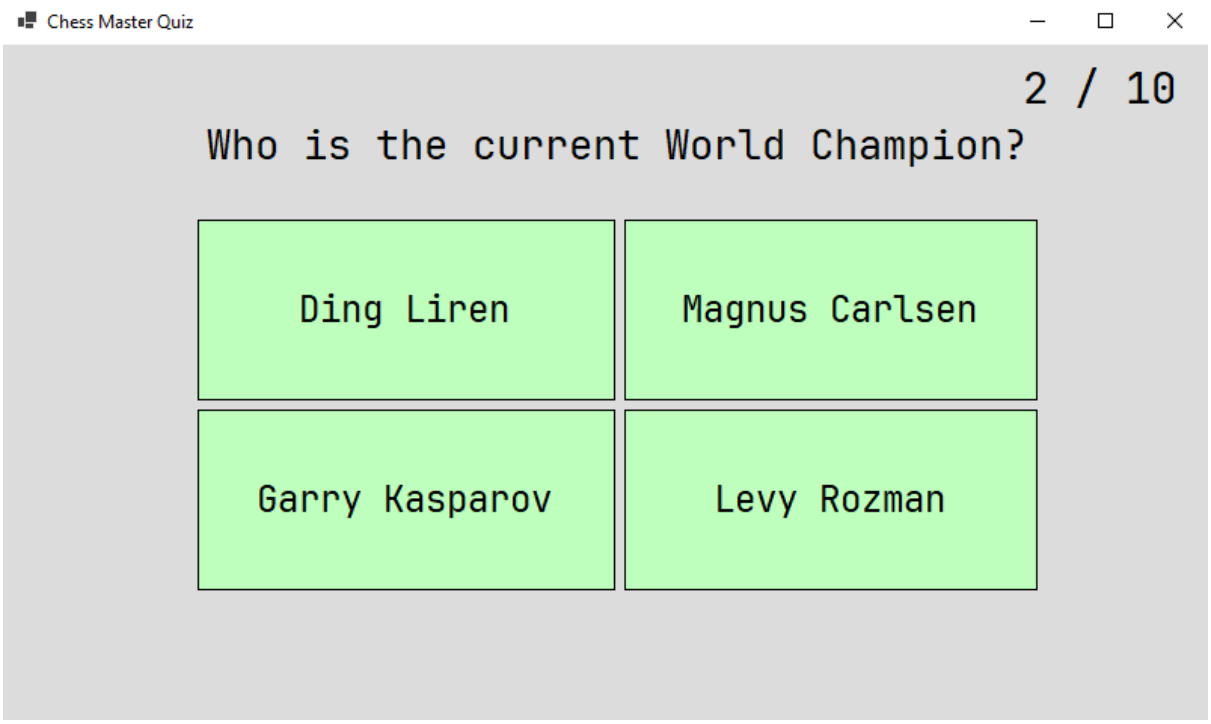
Puzzle Question - `formPuzzleQuestion.cs`



This form can display a chess puzzle to the user. The user is then able to select the move they think will win.

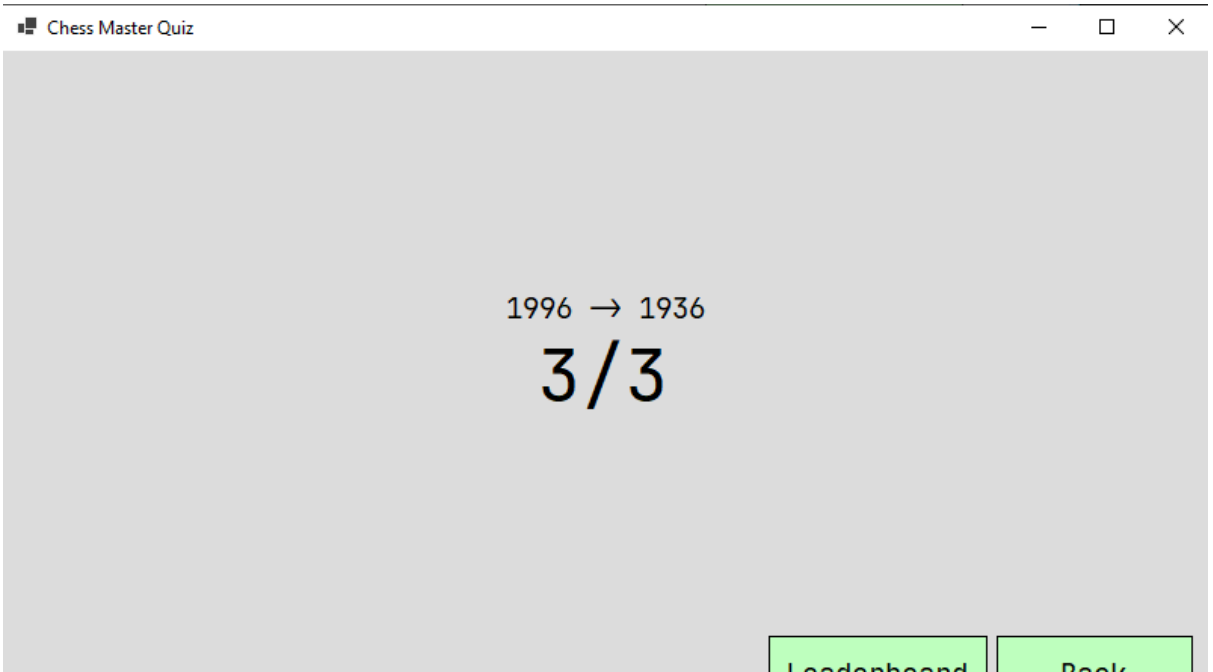
This form has no knowledge of the other forms in the list of questions the user is being asked. Instead, a callback function is passed into the form from the *Choose Question Menu.* This callback function is then called when the user clicks on one of the answer buttons. The callback takes in the answer that the user chose as a parameter, which allows the user's elo and accuracy to be updated based on if the user got the question correct or not. Once the user's stats are updated, the callback creates a new form with the next question.

Written Question - `formTextQuestion.cs`



The *Written Question* asks the user a question about chess and gives them four different options to choose from. Under the hood, this form works virtually identically to *PuzzleQuestion*, the only difference being a slight change in the callback function passed through.

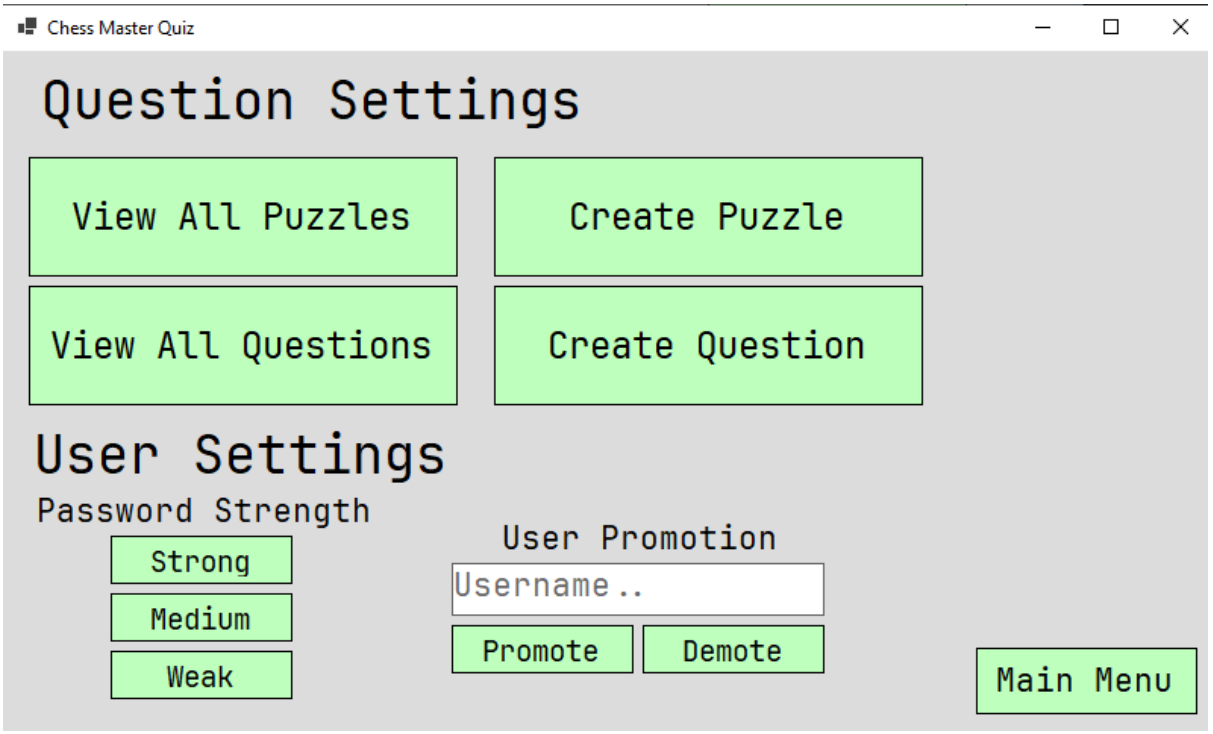
Result form - `formResult.cs`



This form shows the user their stats after answering a round of questions. The main number shows how many questions the user got right out of the number of questions they were asked. The arrow shows the ELO growth of the user. This is obtained by looking back by the appropriate number of elo values.

From the results form, the user can then go straight to the leaderboard, to view their new standings against other users. Alternatively, the user can return to the main menu and answer more questions if they want.

Admin Page - `formAdminPage.cs`



This admin page allows the admin to create new questions and view all current questions. It also allows the admin to change the password strength rules. These rules apply to all users. Another thing that the admin can do is that they can promote a normal user to an admin. They can also demote other admins to regular users. However, the admin cannot demote themselves, as there must always be one admin.

Evaluation

User Requirements

User Requirement	Achieved?	Comment
UR1: Application should have a splash screen	Yes	The application has a fully functioning splash screen, with a chess game playing as the quiz loads.
UR2: Application should have a login screen	Yes	The application has a fully functioning login screen, where the user can login, or be directed to a register screen
UR3: Application should have a register screen	Yes	The application has a register screen which can create a new user account.
UR4: The application should have a main menu with easy navigation to all other menus	Yes	No menus are more than 3 clicks away from the main menu.
UR5: The application should have a leaderboard to display in relation to other users	Yes	The application does have a leaderboard which can sort users by various stats
UR6: The application should have different types of questions, both text based and puzzle based questions	Yes	The application contains both puzzles written questions
UR7: The application should keep track of the user's stats, such as their accuracy of gettings questions right	Yes	Many different stats for each user are tracked, such as Elo, Accuracy and High score.
UR8: The application should tailor the difficulty of questions to the skill ability of the user	No	The questions are presented to the user randomly
UR9: The User should be able to logout and into the application without exiting	Yes	The user can logout and log back in from the main menu and login menu

the application		
UR10: The application should have a settings screen where they are able to change their password and profile picture	Yes	The user is able to change both their password and profile picture
UR11: An admin should be able to create new admins	Yes	An admin can create new admins, but also can demote other admins to regular users
UR12: An admin should be able to create new puzzles and questions	Yes	The admin can create both puzzles and written questions
UR13: An admin should be able to view all puzzles and questions, and delete them	Yes	The admin is able to view all puzzles and questions, but not delete them

Time Management

I believe my time management for this project was one of the weakest parts. Planning my application and how I would tackle the writeup alongside the code and application development would have greatly improved the workflow of development. While I was confident in my ability to finish the coding part of the coursework, I neglected the writeup portion until near the end of development, which meant I had to work a lot more near the end rather than having my effort spread out throughout the development process.

When I encountered problems, I was able to solve them quickly, usually from my own knowledge, however sometimes I consulted websites such as Stack Overflow for a more experienced developer’s take on the problem at hand. Overall, I do not believe the problems I encountered created an issue with my time management.

I found it much easier to write code in longer blocks of time, as I often knew exactly what I wanted to do, and how to do it. This allowed me to sit down and implement the feature quickly, and meant that I could work for longer periods of time as I was less fatigued from mental strain. In terms of the write-up, I

found that I found it much harder to sit down and work on it for longer periods of time.

When developing the application, I did not have a set plan for what hours I would work, which meant I was not steadily working through the workload. If I were to create this application again, I would definitely create a timetable or at minimum a loose plan for when I would work on the code and the write-up. This way I would be more organised and finish the MVP before the target, allowing me to focus on other features which would provide more functionality to the quiz.

Testing Procedure

For testing my application, I used a variety of different methods. Most of my testing was Unit Tests, where I could validate that the different parts of the application worked as I expected them to. Each unit test was tested twice to ensure the reliability and robustness of the program.

If I were to redo the testing for my application, I would code my unit tests instead. Coding my unit tests would allow me to ensure that the behaviour I was experiencing would always happen. In addition, coding my unit tests would create many more tests than I could do in a much shorter amount of time. I could also use a unit testing framework for C# that would distinctly cut down on the time required to create the unit tests.

In addition to unit tests, I also did User Acceptance Testing. These UA tests allowed me to gain a better understanding of how people experience interacting with the application.

I believe that I could have incorporated end users in my testing more, as the application is ultimately for them, they are arguably the most important type of testing. Due to the lack of consultation of an end user, I had to make decisions by myself, without considering what an end user might want in the application. Fortunately, I would be an end user of my own application as I am very interested in chess, which is what the quiz is based on. However, only having myself as an end user prevents me from collecting other people’s opinions and preferences, which meant that the application would tend to mostly cater towards what I personally would want in a chess quiz.

Project Management

I think that I could have managed my project better. I did not have a clear idea of what I wanted my application to be or look like from the beginning, which meant that I was “freestyling” as I created the application. However, I believe I achieved the vast majority of what I wanted to achieve from the start. If I had more time, I could have finished all of the features I would have wanted in the application. I believe a key factor to the success of implementing many of these features was genuine interest into my application and excitement for developing the application.

After developing the application, I would say that I have a much better understanding of the time scales needed to complete the application, and the various subtasks within it. I also believe that setting milestones for specific dates would have improved the steady growth of my application and helped me keep on track to finish my program well before the deadline.

Self Management

I believe that I worked extremely effectively and efficiently on my application, however, if I had a good plan paired with those qualities, my application would have doubtlessly been much better, in both design and implementation. I could have better prepared myself by looking at other applications with the same concept as Chess Master, and also looking at popular websites and taking notes on how they design their UI.

While I believe I am proficient in both C# and my IDE of choice, Visual Studio, I did ensure that I was as efficient as possible while coding, by installing a *Vim Motions* plugin for Visual Studio. Vim motions allow me to edit text, and code in general, extremely quickly, which meant I could take the ideas from my mind and get them into working code very quickly. This ensured that there was a very short time between idea and working prototype, which I would say was beneficial to the overall development of my application,

I did not keep a document of the day to day process of developing the application, which would have helped me to reflect on how I tackled and solved the previous problems I encountered while developing the application. If I were to create the application again, I would definitely take 10 minutes each day and write about what I did that day and what problems I overcame.

Application Strengths and Weaknesses

I believe that my application is extremely strong in some areas, but it can also be quite weak in other areas. I think that in terms of technical implementation, my application is quite strong. I also think that my application contains some impressive features, such as a working chess board, which greatly improves the application.

I am also proud of my rating system, which is based on the Elo system from chess. This rating system is robust and promotes progression through the leaderboard. Another strength which I believe my application has is its navigation through different menus. Navigating around menus is quick and seamless, and very easy through code.

Another key strength to my application is the variety of difficulty of questions in it. It can range from very simple questions to very hard puzzles and trivia that even experienced chess players may not know.

However, my application was also weak in a few areas. For example, in terms of being aesthetically pleasing, some people may not appreciate the green colour scheme. If I had time, I would have changed the colour scheme to be a more neutral colour.

I also believe that my application was weak in terms of the variety of the question types. My application only has a couple types of questions which means that after a couple plays of them, they can get quite stale. However, on the flip side of this argument, my application has a large quantity of questions, which can help prevent the app from becoming stale over time as the probability of the user encountering the same question is very low.

Future Improvements

If I had more time to improve my application I would make a variety of different changes and improvements.

- I would add multiple more different types of questions. These new questions would help to stop the quiz becoming stale and they could add exciting new features.

- If I had the option to change the programming language, I definitely would. While C# is a fine, middle of the road programming language, Windows Forms is extremely sub par, and is not a pleasant development experience. If I were to keep the application as a Desktop App, I would consider changing the language to *Rust,* using *SDL* to create a window and display the information to the screen. Rust is a great language due to its memory safety guarantee (through a *borrow checker)* without needing a *Garbage Collector*! However, If I had the choice, I would change the application from a desktop app to a web app. Web apps are increasingly becoming the norm and porting my quiz to one would definitely be beneficial for the development and ease of access. I would use the language, *Go,* with *HTMX*, to create the web app.

- In addition to changing the programming language, I would most definitely stop using Visual Studio as my IDE. Instead, I would switch to *Neovim*. Neovim is part of the new wave of editors, which are not IDEs, but rather PDEs, (Personal Development Environment). Personal Development Environments are specific to the user, and have vast customizability options, which allow the experience to be tailored to the user. Using Neovim, the entire development process can be much faster, due to my comfort with the text editor. Neovim can also be used with any language, due its *Treesitter* and *LSP* plugins.

- Another improvement I would definitely make is to use a database instead of storing information on disk. I would either use a SQL or a NoSQL database. While relational databases such as SQL can be quick and allow for complex relations between tables, which can improve querying data, NoSQL and non-relational databases like *MongoDB* have a document based approach to storing data, which could be useful for storing information about users and questions.

- In addition, another improvement I would like to make would be to implement online functionality. This would allow multiple users to connect from across the world and share their chess knowledge. Having the quiz be online would also greatly improve the leaderboard functionality as having many players would create a competitive nature for being at the top of the leaderboard. Users could also compete directly against each other, competing to see who could answer the most correct questions in 30 seconds etc.

- Ideally, I would also like to add a fully functional chess playing section to the application, where a user could play against someone from across the world, or against a friend on the same computer. If I were to implement this functionality, I believe also porting the quiz to a mobile app would greatly improve the user experience as they would be able to play chess, and quiz themself online wherever they go

- Another improvement I would make would be accessibility features. Having a range of accessibility features such as a dialogue option which would ask the questions to the user through the speakers would go a long way in making the application more accessible.