2021

Gymnastics Membership System

NEA PROJECT

FINLAY GRAY

Table of Contents

Analysis

The Problem

I am both a gymnast and a coach at a gymnastics club . We have a membership system, but it is very outdated. It very simply links gymnasts to classes, allowing a register to be taken by the coach and does not give much further functionality. This led me to doing my NEA on this topic creating a membership system for my gym.

As a gymnast what I would like to see in a membership system is a way in which I could view my timetable, my coaches, and any kind of time changes.

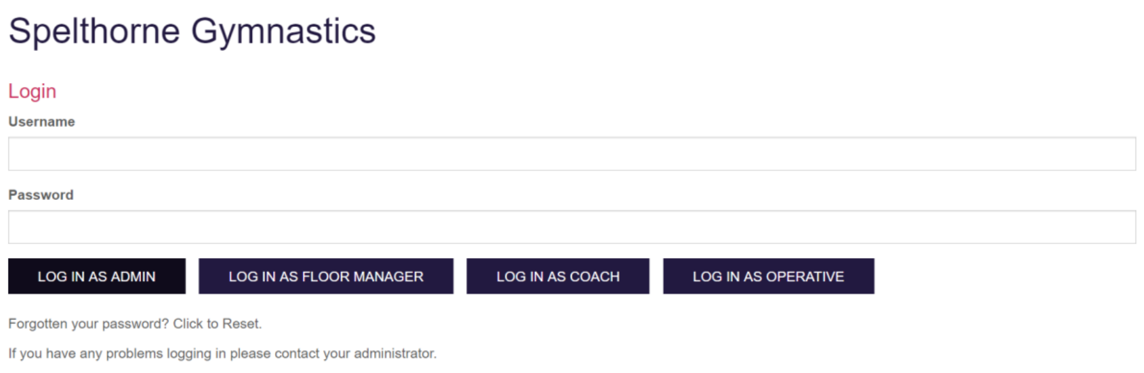
As a coach I would like to see a way in which I could see what classes I am coaching and which gymnasts are in those classes, allowing me to complete a register at the start of every class.

My initial thoughts were to create a system where every coach and gymnast has a specific log in allowing them to access a timetable, and if they are a gymnast, they could see who their coaches are, whereas if they are a coach, they can access registers. As well as this I think there should be some kind of admin log in that could change logistical times, such as the dates/times of any events such as a Christmas display or a competition and have the highest access level.

Analysing the Old System

The old system is purely for coaches and other staff at the gymnastic club ,not for members, and I think this is a big flaw as I believe they are missing out on a big target audience that would love a system like this.

The look of the login page


Spelthorne Gymnastics

Login

Username

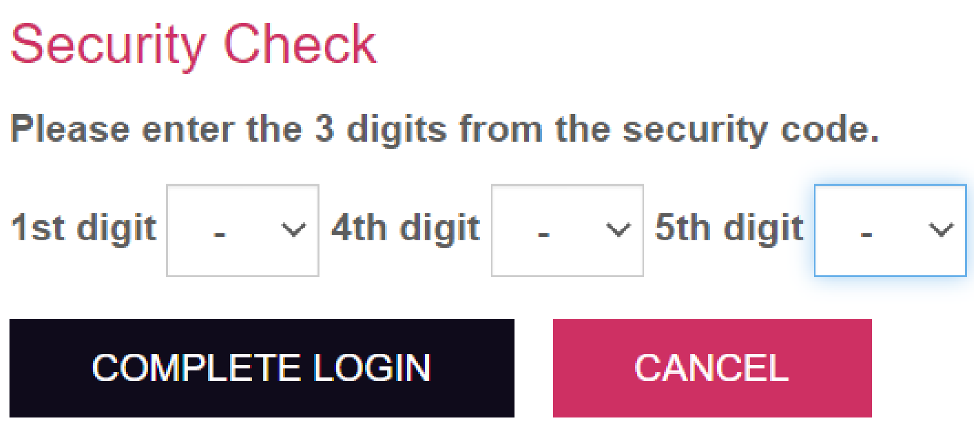
Password

LOG IN AS ADMIN | LOG IN AS FLOOR MANAGER | LOG IN AS COACH | LOG IN AS OPERATIVE

Forgotten your password? Click to Reset.

If you have any problems logging in please contact your administrator.

This login page has a very clean look to it. It is very simple and easy to navigate. I am going to take the same kind of approach when it comes to my login page as I believe, being the first thing, a user sees when they start the program, it must look clean and appealing. However, I do not like the fact that there are four different buttons to log in as this can be quite confusing to new users who will generally tend to press the first log in button by default. Due to this I will have only have one login button for a more user-friendly UI.



Security Check

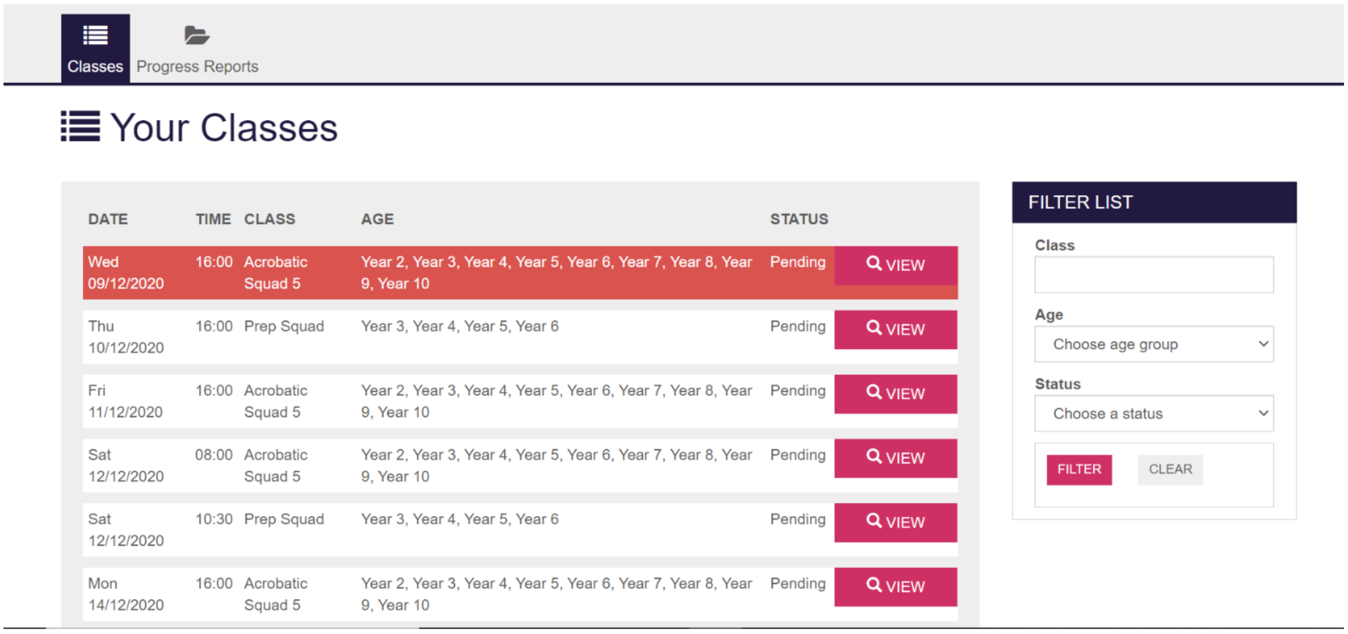
Please enter the 3 digits from the security code.

1st digit [-] 4th digit [-] 5th digit [-]

COMPLETE LOGIN   CANCEL

After logging in the site has a secondary security check which you must fill out each time.

This uses a security code which is known to all staff and requires you to enter 3 digits of it. This is a good extra security tool when you have a relatively small number of users, however with a larger number of users this becomes a problem as you will have to distribute this code to every user. This then damages the integrity and the usefulness of this security check and is why I will not be implementing this is my program. As well as this it is not very user-friendly to the younger ages as they might not be able to pass this relatively confusing task.



Classes | Progress Reports

Your Classes

DATE	TIME	CLASS	AGE	STATUS	
Wed 09/12/2020	16:00	Acrobatic Squad 5	Year 2, Year 3, Year 4, Year 5, Year 6, Year 7, Year 8, Year 9, Year 10	Pending	VIEW
Thu 10/12/2020	16:00	Prep Squad	Year 3, Year 4, Year 5, Year 6	Pending	VIEW
Fri 11/12/2020	16:00	Acrobatic Squad 5	Year 2, Year 3, Year 4, Year 5, Year 6, Year 7, Year 8, Year 9, Year 10	Pending	VIEW
Sat 12/12/2020	08:00	Acrobatic Squad 5	Year 2, Year 3, Year 4, Year 5, Year 6, Year 7, Year 8, Year 9, Year 10	Pending	VIEW
Sat 12/12/2020	10:30	Prep Squad	Year 3, Year 4, Year 5, Year 6	Pending	VIEW
Mon 14/12/2020	16:00	Acrobatic Squad 5	Year 2, Year 3, Year 4, Year 5, Year 6, Year 7, Year 8, Year 9, Year 10	Pending	VIEW

FILTER LIST

Class: [ ]

Age: Choose age group

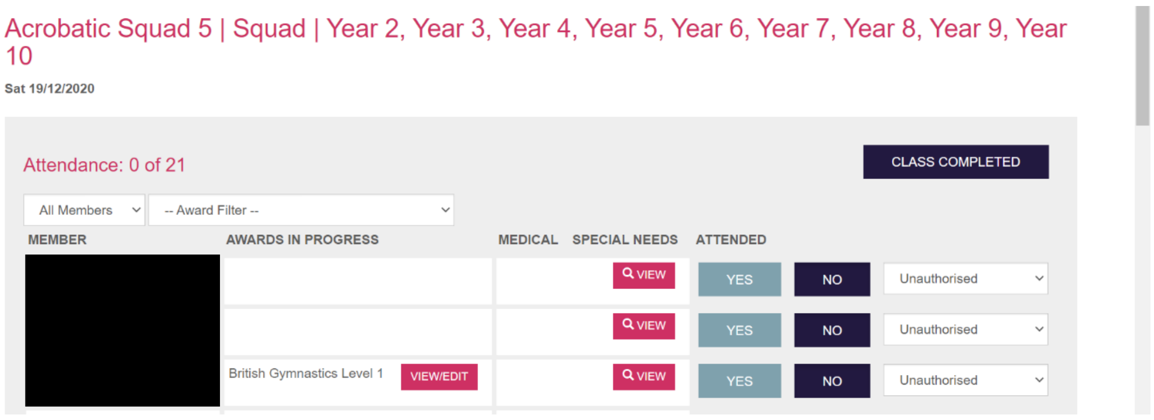
Status: Choose a status

FILTER CLEAR

This is how the page looks for a coach after logging on. It defaults to showing you the registers you have taken, and your pending or overdue registers are in red. It is also showing very clear details on each class, such as date, time, class name and the range of school years of the gymnasts in the class.

On the right there is also a filter list where you can search for different classes if you coach many different ones. This can be very helpful and is a very good feature that I could use in my project.

I also like the layout of how the registers are shown as they are very easy to access, read and show a lot of details about each class before you have even clicked on them. However, I do not know how necessary it is to include so much detail on this page as all that is needed is the date, time and class on this page.


Acrobatic Squad 5 | Squad | Year 2, Year 3, Year 4, Year 5, Year 6, Year 7, Year 8, Year 9, Year 10

Sat 19/12/2020

Attendance: 0 of 21

MEMBER	AWARDS IN PROGRESS	MEDICAL	SPECIAL NEEDS	ATTENDED		
[redacted]			VIEW	YES	NO	Unauthorised
			VIEW	YES	NO	Unauthorised
	British Gymnastics Level 1 VIEW/EDIT		VIEW	YES	NO	Unauthorised

When you click on the register for a class it takes you to this screen. This is just a small section of the page and it lists every member down the page. On the very left it shows the names of the gymnasts in the class which I have blurred out. It then gives a couple of details such as medical issues or special needs i. On the right it gives you an option to mark if the gymnast is present or not. Once you have marked all the gymnasts present or not you press class completed in the top corner and it submits the register into the system.

Again, this looks clean, simple and for taking a register is a good system.

This, however good at being a register taking program, is the extent of what is possible for a coach to do.

The Interview

After thinking more about this I decided to conduct an interview with the General Manager of my gym to ask her what she would like in a membership system.

Q1: What are the main features that you think are needed in a membership system?

A1: Well, the fundamentals of a membership system are the ability to store and access data about gymnasts, classes, waiting lists. You should also be able to calculate and collect data so that you can work out facts such as gender percentages. As well as this I think there should be an easy way of sending mass emails, taking registers, and allowing clients to pay fees.

Q2: Personally, what additional features would you like to see in the system?

A2: For additional features I would like to see a way in which the system could create a community between gymnasts, parents, coaches, and admin to allow everyone to feel more of a part of the club. For example, some sort of communication system.

Q3: What about from an admin standpoint?

A3: From an admin standpoint I would like to see a feature that can create automatic time sheets from coaches' hours as this would allow the admin to free themselves up for other more pressing and less boring tasks.

Q4: As the General Manager what access levels would you like your coaches to have?

A4: As coaches they should, of course, be able to access registers and certain important details about the gymnasts they coach such as any medical information, gymnastic badges that the gymnast has earned, the gymnasts name, DOB, and contact number. As well as this they should only be able to access only their personal timesheets and perhaps be able to request the moving of a gymnast, however, they should not be able to move them themselves as this should be something only admins are capable of.

Q5: When a user logs on to the system what would you like to be the standout thing they see?

A5: As soon as they have logged on, they should see a welcome screen, showing their name at the top and the class they are in. It should also show their timetable and any upcoming events. I think it would be good if they had two options after that, one which leads to all the details of the membership (more for the parents), including when the next payment is due, all their contact details.

The other option (more for the gymnast) takes you to a page which show their class, their coach, a way to communicate with their class, a little game, and a place here they can view which badges they have.

Q6: Finally, from a design perspective, how would you like the system to look, clean and simple or lots of information on each page?

A6: I would rather have it looking clean and simple as this makes easier to navigate, which is especially helpful for the gymnasts and a lot of them are quite young.

Conclusion of interview

This interview with the General manager opened my eyes to a couple additional extras I could add, as well as more of an insight to what the most important factors should be. One important detail that came out of this interview was the fact that coaches should not have as high of an access level as I originally believed, and that every change should go through the admin before it happens. As well as this another detail that she added was that the gymnast login page should have two different pages, for child and parent. This will make it more user friendly as the child will not want to see all the "boring" details of the site.

High Level Requirements for the System

Must Have Requirements

- Each gymnast/coach/admin must have a log in which gives them access to the system.
- When creating an account there must be a way to fill in details about the person that need to be stored.
- There must be a timetable displayed to gymnasts and coaches when they log on showing their classes.
- There must be a way coaches and admin can view and take registers.
- There must be a way admin can edit classes by adding and removing both gymnasts and coaches to and from classes.
- There must be a way admin can change class times and requirements for classes such as max number of gymnasts.
- Admins must be able to create the accounts.

    - Must be a way for them to easily fill in all required details.

- It must be user friendly.
    - Suitable for primary school children to adults

- There must be a way that gymnasts can view all their details as well as what class there are in and their coaches.

- There must be a way that users can create their password when first logging on given their already known membership id.

Should Have Requirements

- There should be an automatic system which works out how many hours a coach has worked and how much money they will get at their specific rate.

- Admin should be able to change the pay rate for all staff.

- There should be a way admin can give out badges.

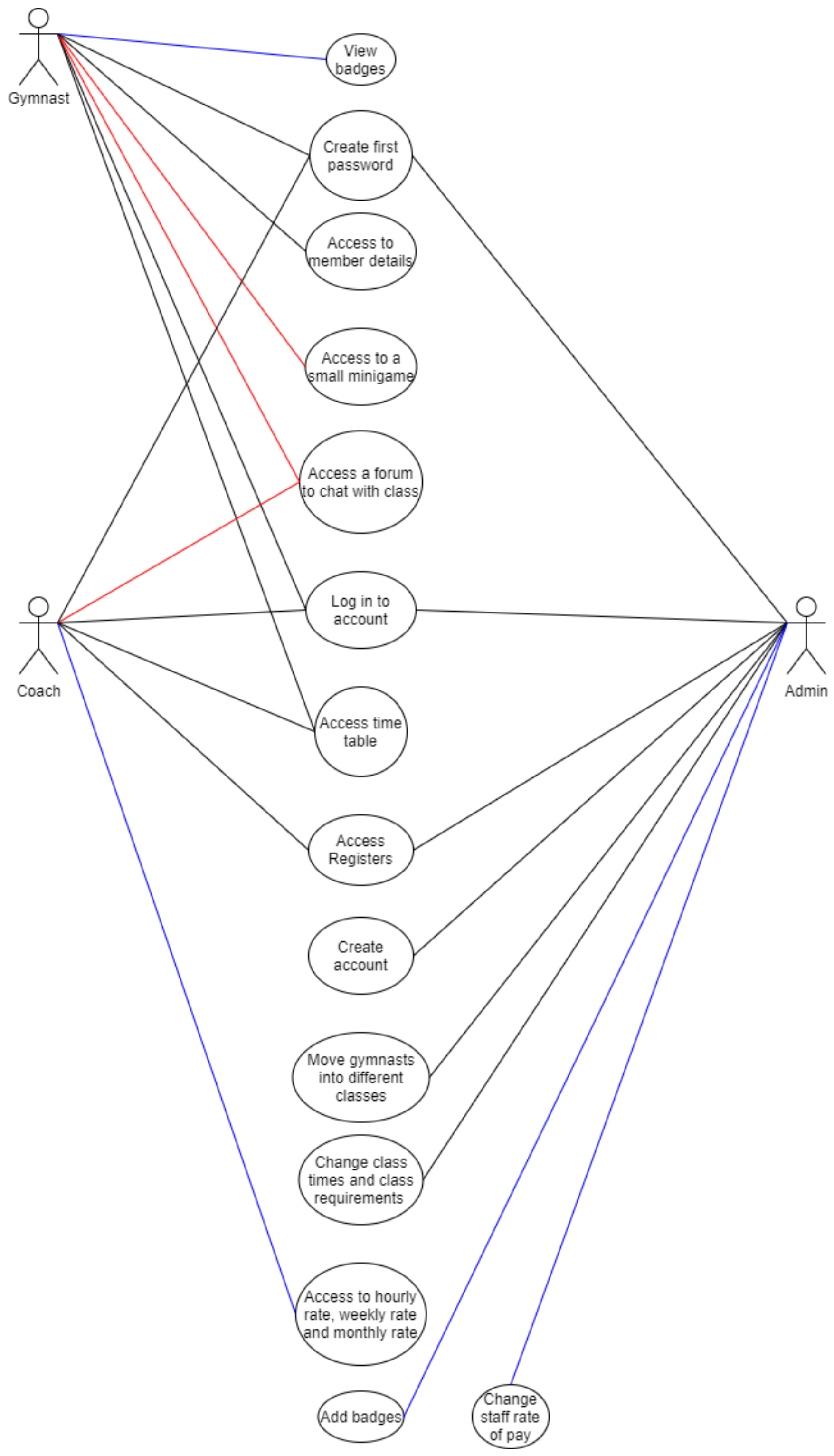
- There should be a way the gymnasts can view their badges they have earned.

Could Have Requirements

- There could be small minigame available to the gymnasts, such as a simple 2d run and jump game with the character doing a flip or something gymnastics related.

- There could be a way gymnast can speak to their coaches and other gymnasts in a forum type chat and vice versa.

Use Case Diagram


Gymnast
View badges
Create first password
Access to member details
Access to a small minigame
Access a forum to chat with class
Coach
Log in to account
Admin
Access time table
Access Registers
Create account
Move gymnasts into different classes
Change class times and class requirements
Access to hourly rate, weekly rate and monthly rate
Add badges
Change staff rate of pay

Key:

Black line: Must have

Blue line: Should have

Red line: Could have

Low Level Requirements

L1 When the program is run the user should be presented with a login in screen with clear instruction on where to input Membership ID and password

L2 If it is their first-time logging in there will be a button they can press, the New Member button.

N1 After pressing this button, they will be taken to a screen where they will be able to enter their Membership ID which if first run is set as a default 0

N2 If their ID is valid, they will be taken to another page in which they must set their new password. If the password is not valid then they cannot move on

N3 They then must set up a password which fulfills a certain given criterion. If it passes this then the password will be saved, and they will be sent back to the login page.

L3 Once they have inputted their ID and password, they should press the submit button which will check their details against a database of details and decide first whether the input is valid, and then whether they are a gymnast, a coach or an admin.

L4 On all pages there will be a 'quit' button which will exit the program.

L4 There will also be a 'back button' On every page except from the login pages which will take them back to the previous page they were on.

Gymnast

G1 If they are a gymnast, it will take them to the gymnast base page.

G2 From here they should have the option to click either of three buttons

G3 If the "member details" button is clicked the screen will change to show them all their details, such as id number, postcode etc.

G4 If the “general” button is pressed, they will be taken to a page in which they will be able to view the badges that they have earned, as well as a forum chat in which the gymnasts can chat to coaches and other gymnasts in their class. As well as this it will show the name of their class, the name of their coaches and any badges they have.

G5 If they press the console chat button, then a chat server will run in the console.

G6 There will also be a ‘game’ button on this screen.

G6.1 If the “game” button is clicked, the screen will be cleared, and the game will start.

G6.2 User will be shown game start screen.

G6.3 They will be able to choose between play game and view high scores or exiting.

G6.4 If they press view high scores, they will be shown a page which shows the current high score.

G6.5 If they press the play game button, the game will start.

G6.6 When they press the space button the character will jump.

G6.7 If they hit a spike, then then the game ends. Their score is checked against the stored high score and if it is greater than it will become the new high score.

G6.8 After this check has completed the user is taken back to the game start screen

Coach

C1 If they are a coach, after logging in they will be taken to the coach base page.

C2 This will take them to a page in which they should be able to view their timetable of classes, take register button, a view register button, a timesheet button or a forum chat.

C3 If they press the register button, they will be shown a list of the classes they coach, allowing them to pick a class by clicking it.

C3.1 After clicking this class, it will show the days they train as a button allowing you to pick one.

C3.2 After picking a day/time it will take you to a page in which you can take the register by marking each gymnast as present or absent and then pressing a submit button

C4 If they press the timesheet button, they will be taken to page in which they will be able to view their total hours worked, their rate of pay and how much they will make per week and per month.

C5 If they press the console chat page, then a chat server will run in the console.

C6 If they press the view register button, they will get the same screens as if they pressed the take register button however, there will be no present or absent buttons as it will just show if the user was present or absent.

Admin

A1 After logging in they will be taken to the admin base page in which they will be presented with a screen of buttons allowing them to access many different things

A2 If they press the 'edit' button they will be taken to a page in which they can view all classes

   a. After clicking on this button, they will be taken to a screen in which they can choose to edit the details of the class or the member in the class.
   b. If they press the edit details button then they will be able to edit the details of the class such as days, times, max kids in class and class name.
   c. If they press the edit class button, they will be presented with the names of everyone in the class and the ability to remove any member from the class.

A3 If they press the 'add member' button they will be taken to a page in which they will fill in details to create a new member

   a. They will input data such as:
      i. First name
      ii. Surname
      iii. Membership type
      iv. DOB
      v. Telephone number
      vi. Postcode
      vii. Medical information
      viii. Gender

A4 If they press the 'badges' button, then they will be taken to a page in which they can add a badge to a certain member.

A5 If they press the 'pay' button they will be taken to a page where they can choose a member of staff's rate of pay

A6 If they press the 'create class' button they will be taken to a page in which they will fill out the name of the class, they number of days per week that class will run, and the max number of kids allowed in the class. They will then press a next button.

   a. Once they press the next button, they will be presented with a certain number of input boxes to fill out the days and times of the classes depending on how many days they said the class would run per week.

    b. They will then press a submit button which will input this into the database.

A7 If they press the 'add to class' button they will be taken to a page where they can input an id number of a member. Once they inputted this and pressed the next button, they will be shown a screen of classes that it is possible to add this gymnast to.

    a. Whichever button they press it will add that member to that specific class.

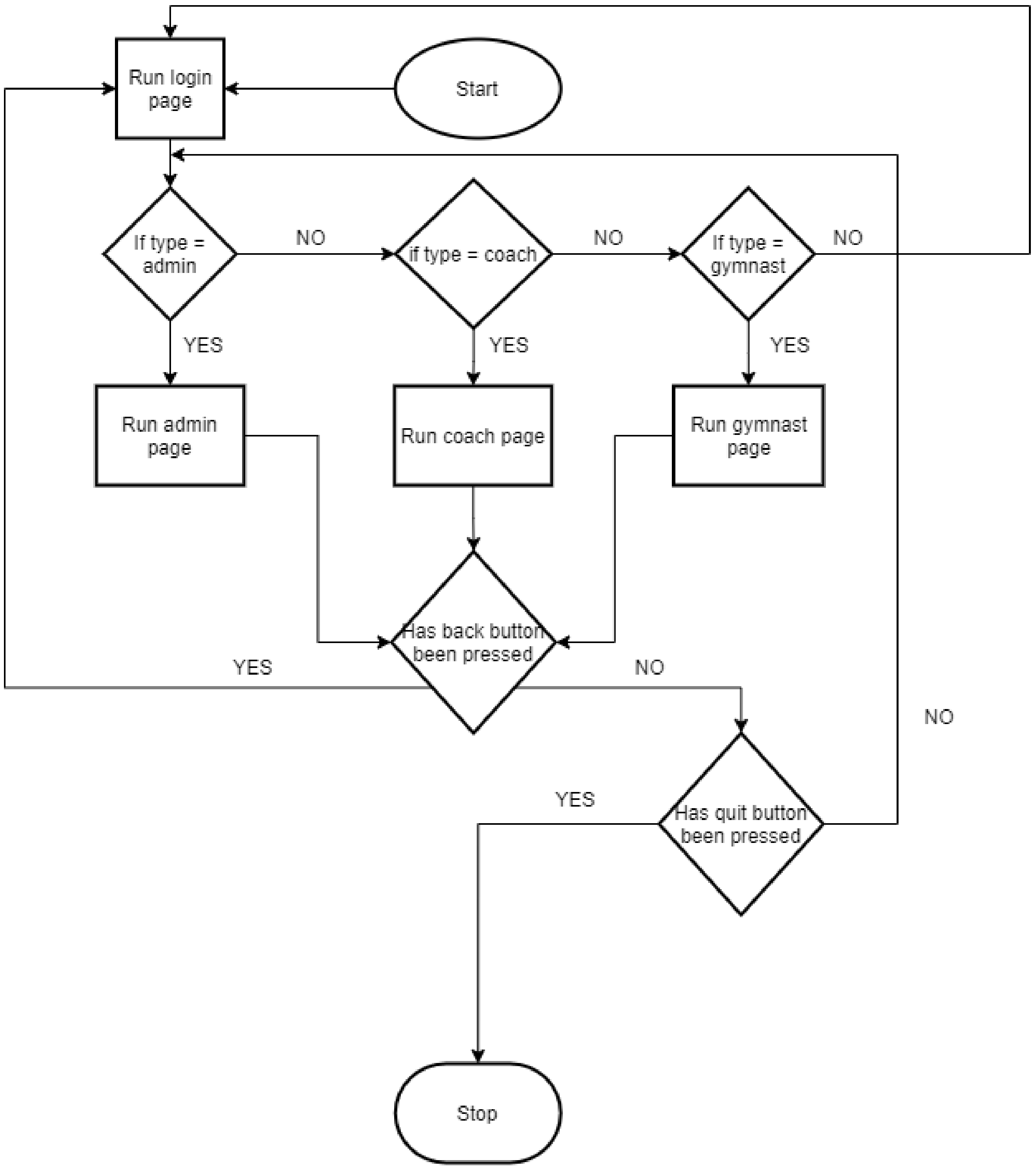
A8 If they press the 'take register' button, they will be shown a list of the classes they coach, allowing them to pick a class by clicking it.

    a. After clicking this class, it will show the days they train as a button allowing you to pick one.
    b. After picking a day/time it will take you to a page in which you can take the register by marking each gymnast as present or absent and then pressing a submit button

A9 If they press the 'view register' button, they will get the same screens as if they pressed the take register button however, there will be no present or absent buttons as it will just show if the user was present or absent.

Design

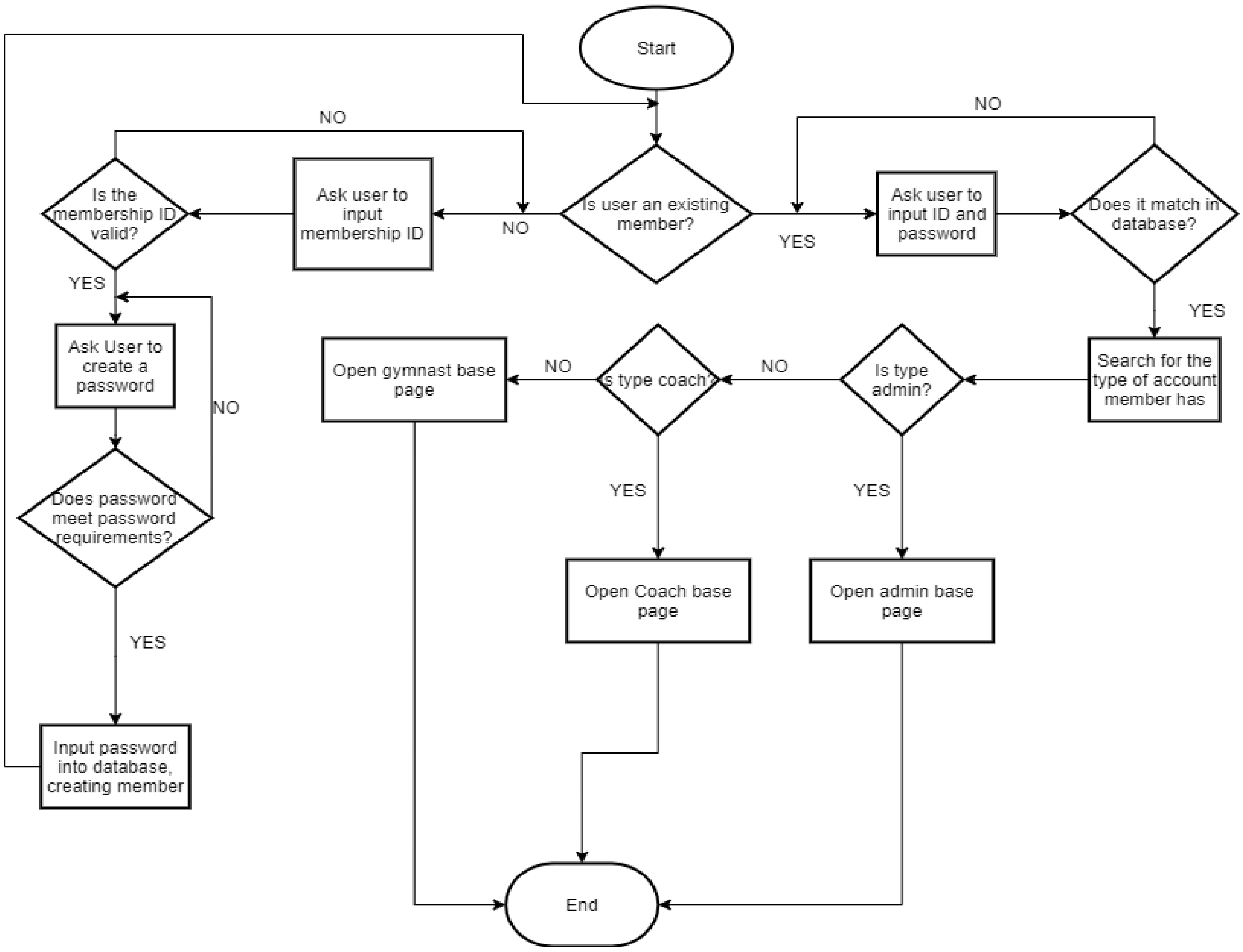
Flow diagram for entire system



This is the general flow diagram for the system. Each type of member will have its own run process, and the system will check for the back or quit buttons.

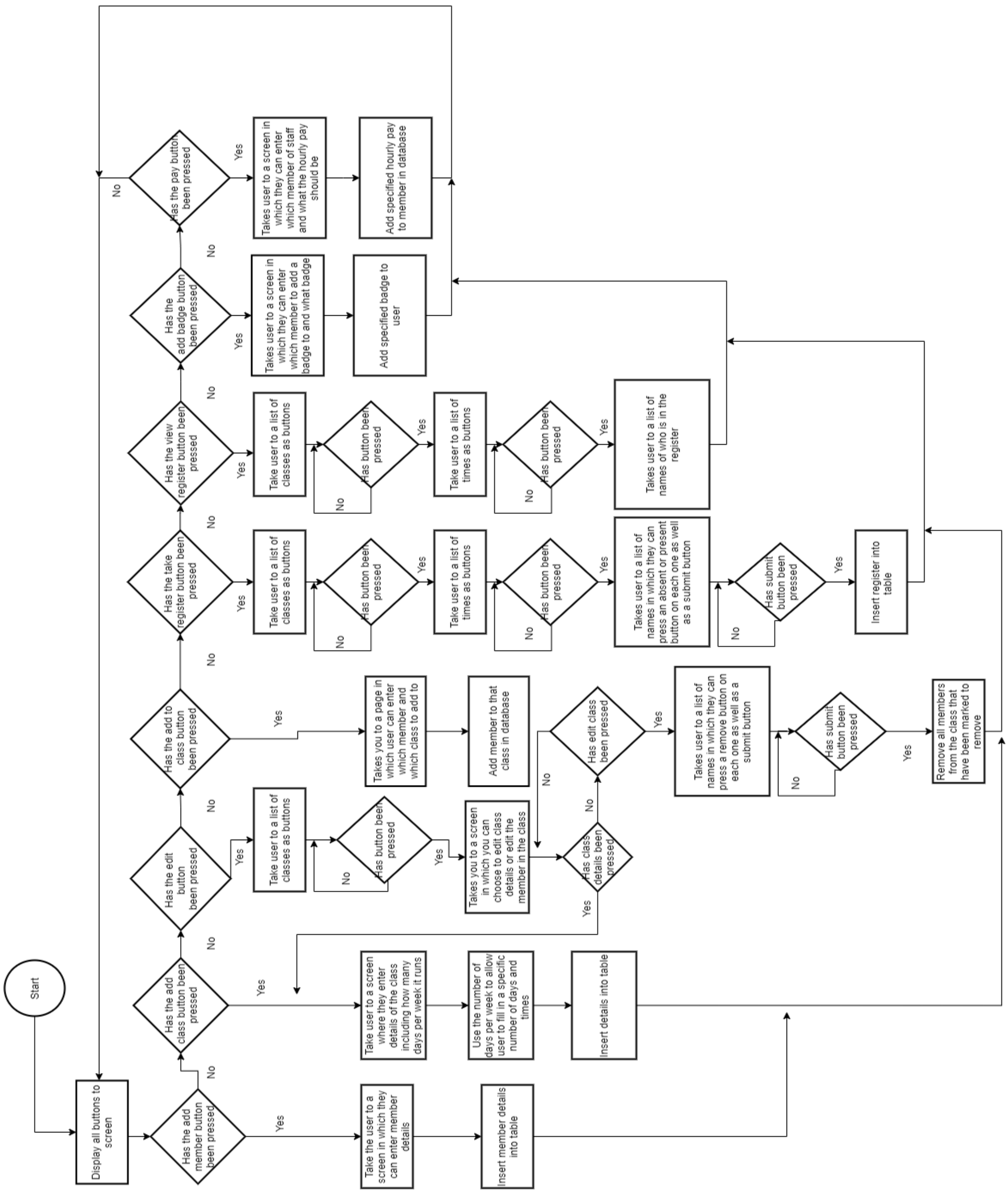
Flow diagram for log on page

Below is the flow diagram for the log on page which will be the first thing the user sees as they log on.



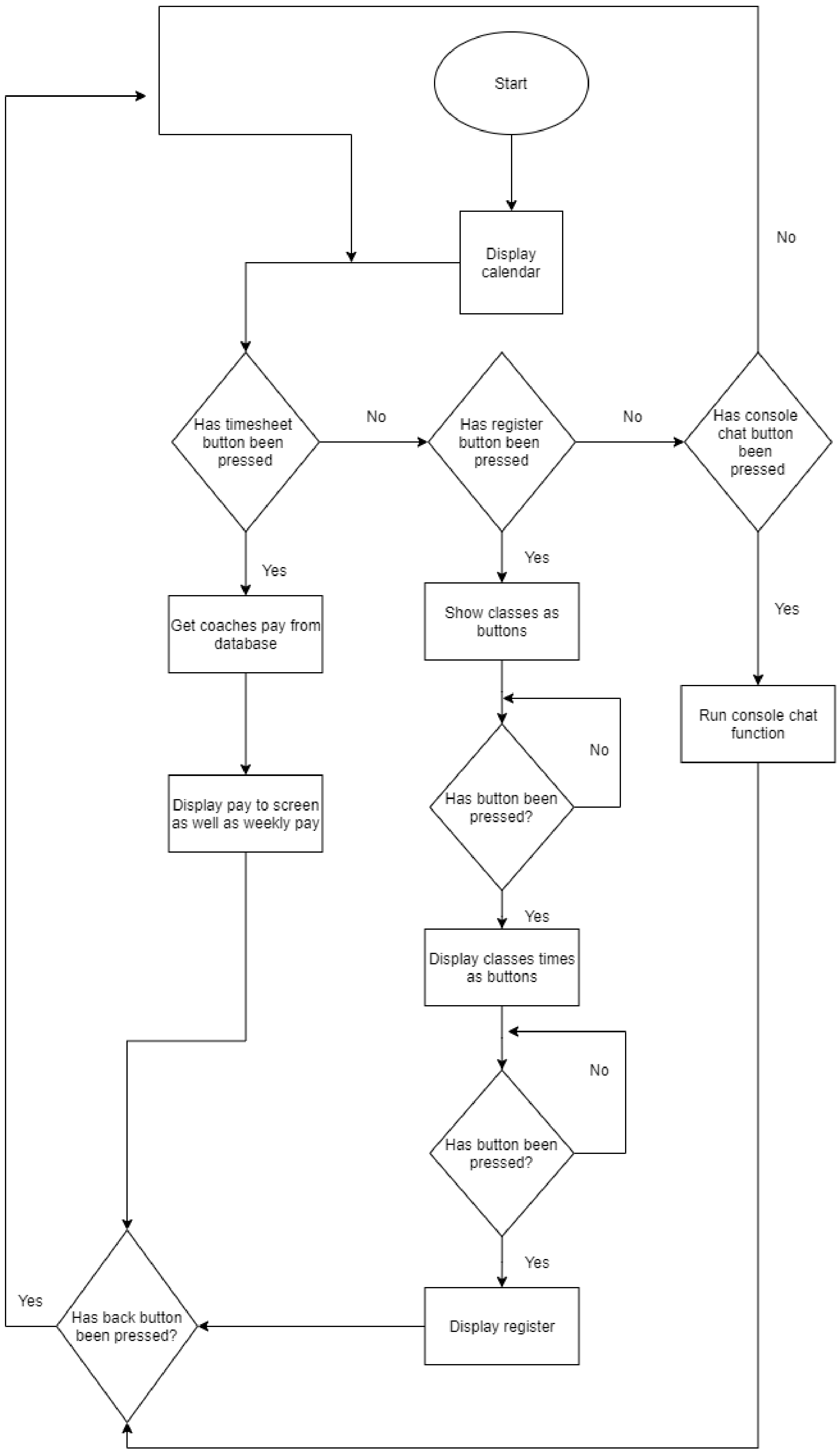
Start
Is user an existing member?
NO
Ask user to input membership ID
Is the membership ID valid?
NO
YES
Ask User to create a password
Does password meet password requirements?
NO
YES
Input password into database, creating member
YES
Ask user to input ID and password
Does it match in database?
NO
YES
Search for the type of account member has
Is type admin?
NO
YES
Open admin base page
Is type coach?
NO
YES
Open Coach base page
Open gymnast base page
End

Flow diagram for admin page



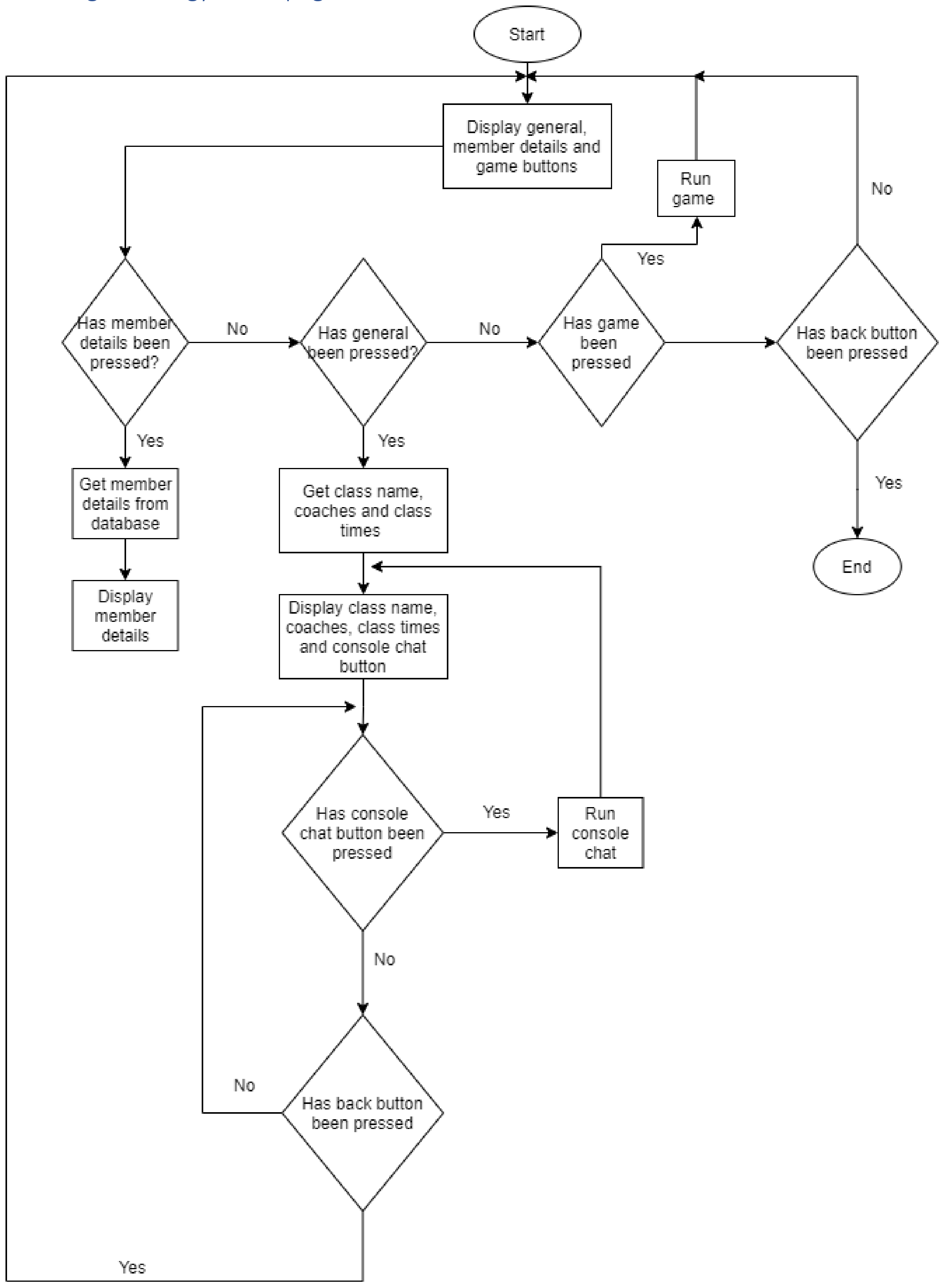
Start
Display all buttons to screen
Has the add member button been pressed
Yes
Take the user to a screen in which they can enter member details
Insert member details into table
No
Has the add class button been pressed
Yes
Take user to a screen where the they enter details of the class including how many days per week it runs
Use the number of days per week to allow user to fill in a specific number of days and times
Insert details into table
No
Has the edit button been pressed
Yes
Take user to a list of classes as buttons
Has button been pressed
No
Yes
Takes you to a screen in which you can choose to edit class details or edit the class member in the class
Has class details been pressed
Yes
No
Has edit class been pressed
No
Yes
Takes user to a list of names in which they can press a remove button on each one as well as a submit button
Has submit button been pressed
No
Yes
Remove all members from the class that have been marked to remove
No
Has the add to class button been pressed
Yes
Takes you to a page in which user can enter which member and which class to add to
Add member to that class in database
No
Has the take register button been pressed
Yes
Take user to a list of classes as buttons
Has button been pressed
No
Yes
Take user to a list of times as buttons
Has button been pressed
No
Yes
Takes user to a list of names in which they can press an absent or present button on each one as well as a submit button
Has submit button been pressed
No
Yes
Insert register into table
No
Has the view register button been pressed
Yes
Take user to a list of classes as buttons
Has button been pressed
No
Yes
Take user to a list of times as buttons
Has button been pressed
No
Yes
Takes user to a list of names of who is in the register
No
Has the add badge button been pressed
Yes
Takes user to a screen in which they can enter which member to add a badge to and what badge
Add specified badge to user
No
Has the pay button been pressed
Yes
Takes user to a screen in which they can enter which member of staff and what the hourly pay should be
Add specified hourly pay to member in database
No

Flow diagram for coach page



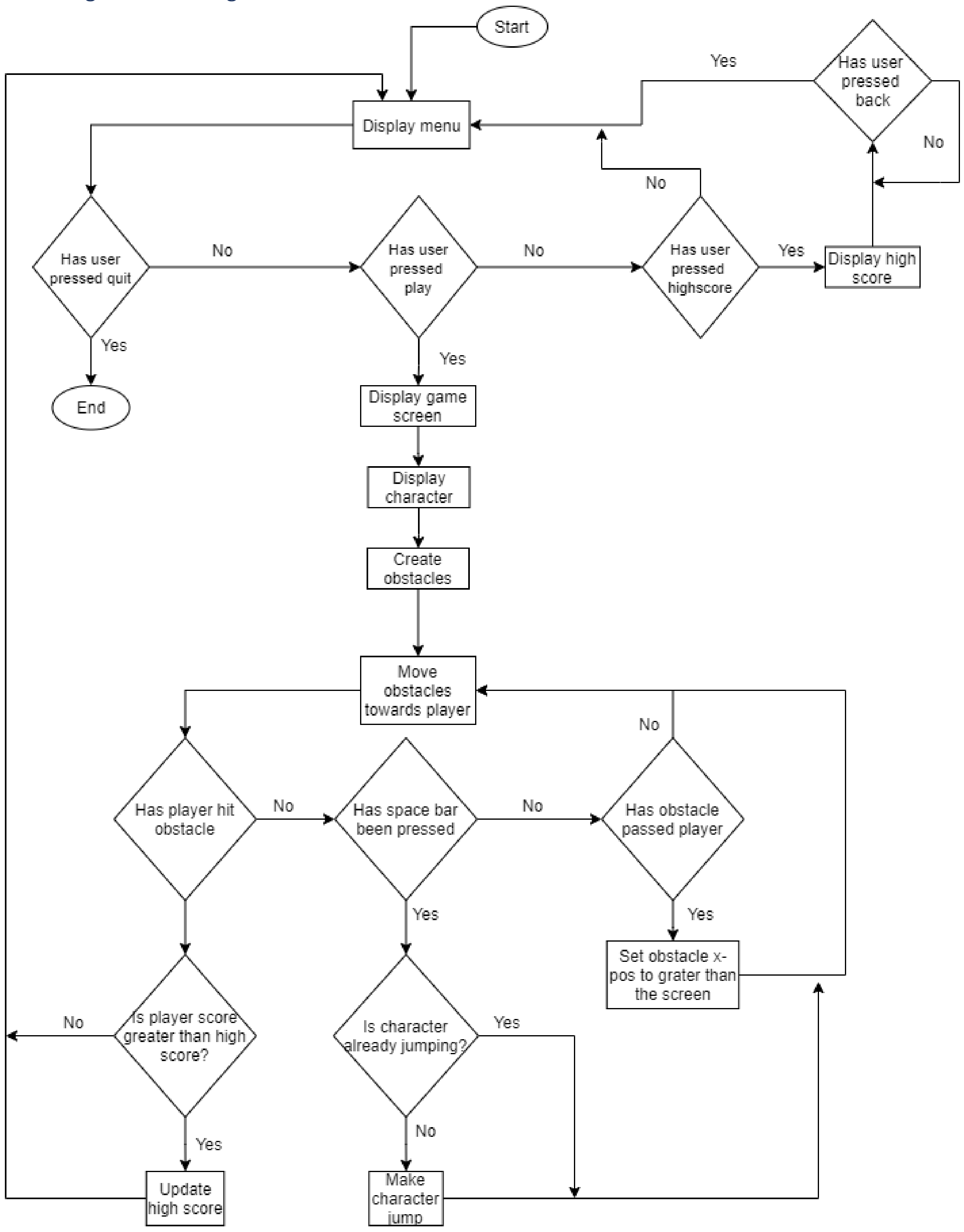
Start
Display calendar
Has timesheet button been pressed
No
Has register button been pressed
No
Has console chat button been pressed
No
Yes
Get coaches pay from database
Display pay to screen as well as weekly pay
Yes
Show classes as buttons
Has button been pressed?
No
Yes
Display classes times as buttons
Has button been pressed?
No
Yes
Display register
Yes
Run console chat function
Has back button been pressed?
Yes

Flow diagram for gymnast page

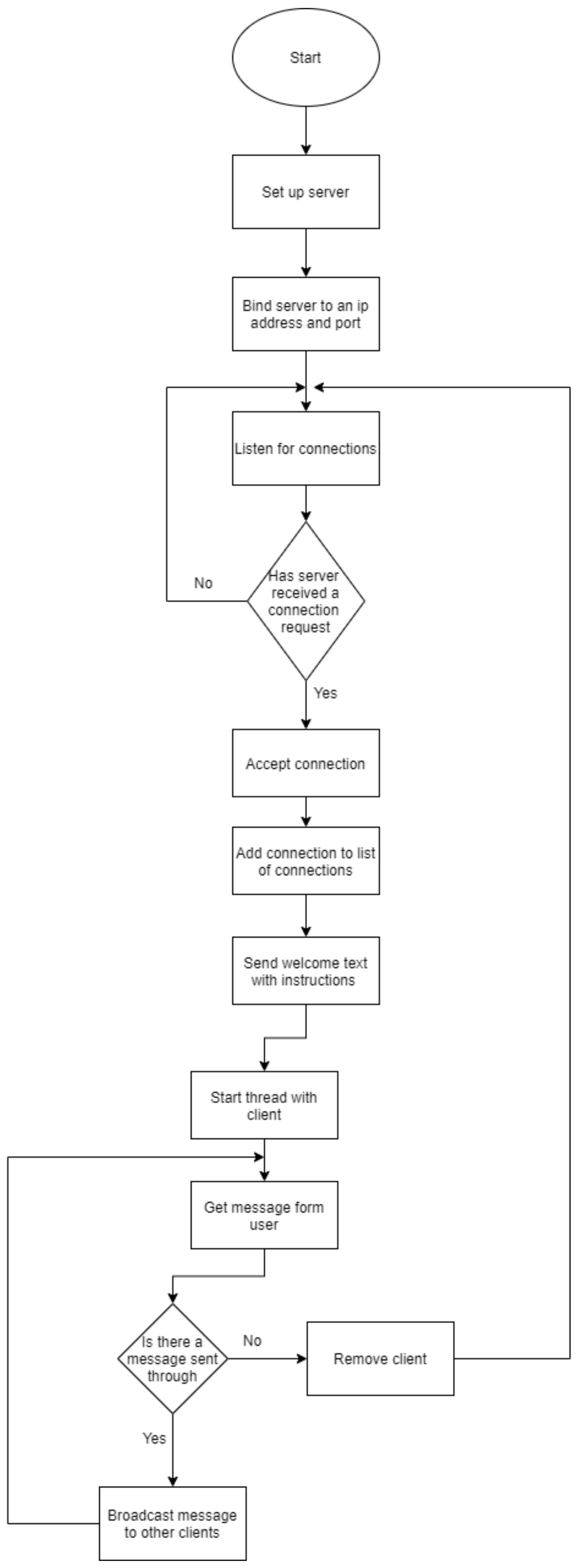


Start
Display general, member details and game buttons
Run game
No
Yes
Has member details been pressed?
No
Has general been pressed?
No
Has game been pressed
Has back button been pressed
Yes
Yes
Get member details from database
Get class name, coaches and class times
Yes
End
Display member details
Display class name, coaches, class times and console chat button
Has console chat button been pressed
Yes
Run console chat
No
No
Has back button been pressed
Yes

Flow Diagram for minigame



Flow diagram for server

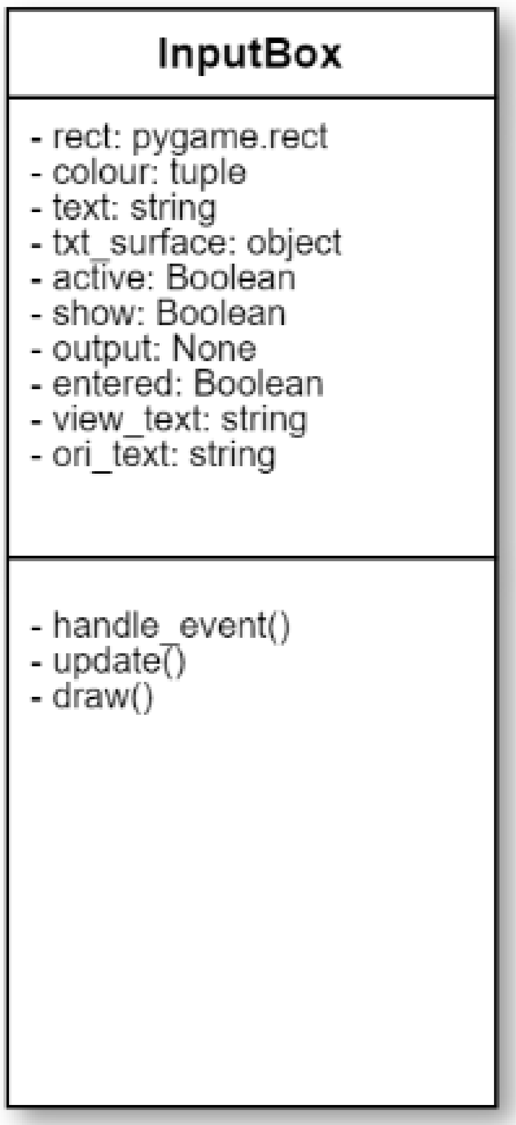


Start
Set up server
Bind server to an ip address and port
Listen for connections
Has server received a connection request
No
Yes
Accept connection
Add connection to list of connections
Send welcome text with instructions
Start thread with client
Get message form user
Is there a message sent through
No
Remove client
Yes
Broadcast message to other clients

Most important classes

Input Box

Class diagram

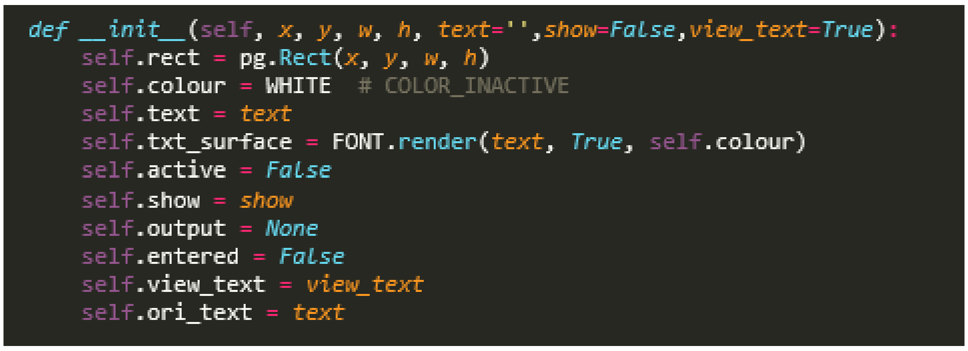


Code

```python
# Text box class
class InputBox:

    def __init__(self, x, y, w, h, text='',show=False,view_text=True):
        self.rect = pg.Rect(x, y, w, h)
        self.colour = WHITE  # COLOR_INACTIVE
        self.text = text
        self.txt_surface = FONT.render(text, True, self.colour)
        self.active = False
        self.show = show
        self.output = None
        self.entered = False
        self.view_text = view_text
        self.ori_text = text

    def handle_event(self, event):
        if event.type == pg.MOUSEBUTTONDOWN:
            # If the user clicked on the input box rect.
            if self.rect.collidepoint(event.pos):
                # Toggle the active variable.
                self.active = not self.active
                self.text = ''
            else:
                self.active = False
            # Change the current colour of the input box.
            self.colour = COLOUR_ACTIVE if self.active else COLOUR_INACTIVE
        if event.type == pg.KEYDOWN:
            if self.active:
                if event.key == pg.K_BACKSPACE:
                    self.text = self.text[:-1]
                else:
                    self.text += event.unicode
                    self.entered = False
                # Re-render the text.
                length = len(self.text)
                star = '*' * length
                if self.view_text:
                    self.txt_surface = FONT.render(self.text, True, self.colour)
                else:
                    self.txt_surface = FONT.render(star, True, self.colour)

    def update(self):
        # Resize the box if the text is too long.
        width = max(200, self.txt_surface.get_width() + 10)
        self.rect.w = width

    def draw(self, screen):
        # Blit the text.
        screen.blit(self.txt_surface, (self.rect.x + 5, self.rect.y + 5))
        # Blit the rect.
        pg.draw.rect(screen, self.colour, self.rect, 2)
```

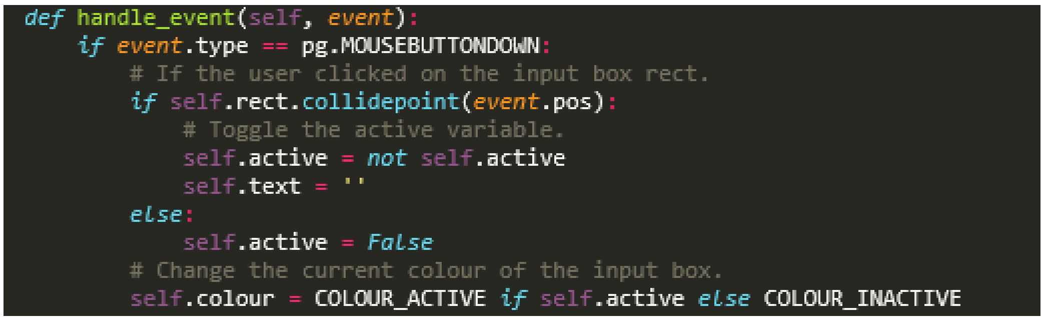
This is the code for the class for the input boxes.

We start by declaring all the variables we will need for the class using the __init__ function.



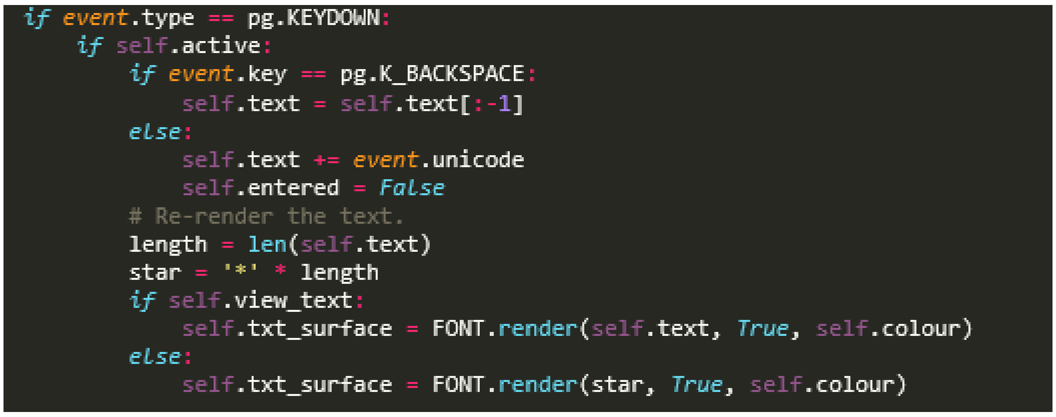
```
def __init__(self, x, y, w, h, text='',show=False,view_text=True):
    self.rect = pg.Rect(x, y, w, h)
    self.colour = WHITE  # COLOR_INACTIVE
    self.text = text
    self.txt_surface = FONT.render(text, True, self.colour)
    self.active = False
    self.show = show
    self.output = None
    self.entered = False
    self.view_text = view_text
    self.ori_text = text
```

We then have the handle event function which is split into two halves by if statements. It first checks whether the user has clicked on the input box or not. If they have it changes the colour of the border and moves of to the second part of the function.



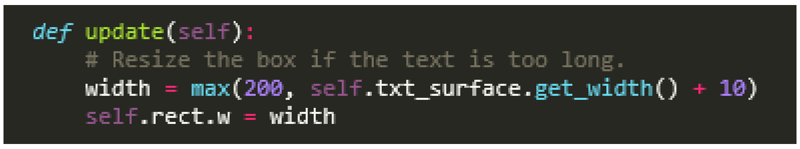
```
def handle_event(self, event):
    if event.type == pg.MOUSEBUTTONDOWN:
        # If the user clicked on the input box rect.
        if self.rect.collidepoint(event.pos):
            # Toggle the active variable.
            self.active = not self.active
            self.text = ''
        else:
            self.active = False
        # Change the current colour of the input box.
        self.colour = COLOUR_ACTIVE if self.active else COLOUR_INACTIVE
```

This then detects the keystrokes of the user to see what they are typing and adds this to the input boxes text. It also checks whether the input box is a password box or not, and if it is a password box it changes the view of each letter to a *.



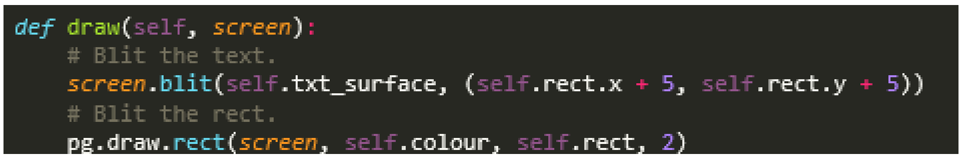
```
if event.type == pg.KEYDOWN:
    if self.active:
        if event.key == pg.K_BACKSPACE:
            self.text = self.text[:-1]
        else:
            self.text += event.unicode
            self.entered = False
        # Re-render the text.
        length = len(self.text)
        star = '*' * length
        if self.view_text:
            self.txt_surface = FONT.render(self.text, True, self.colour)
        else:
            self.txt_surface = FONT.render(star, True, self.colour)
```

After this there is the update function which resizes the length of the input box as the text gets longer.



```
def update(self):
    # Resize the box if the text is too long.
    width = max(200, self.txt_surface.get_width() + 10)
    self.rect.w = width
```

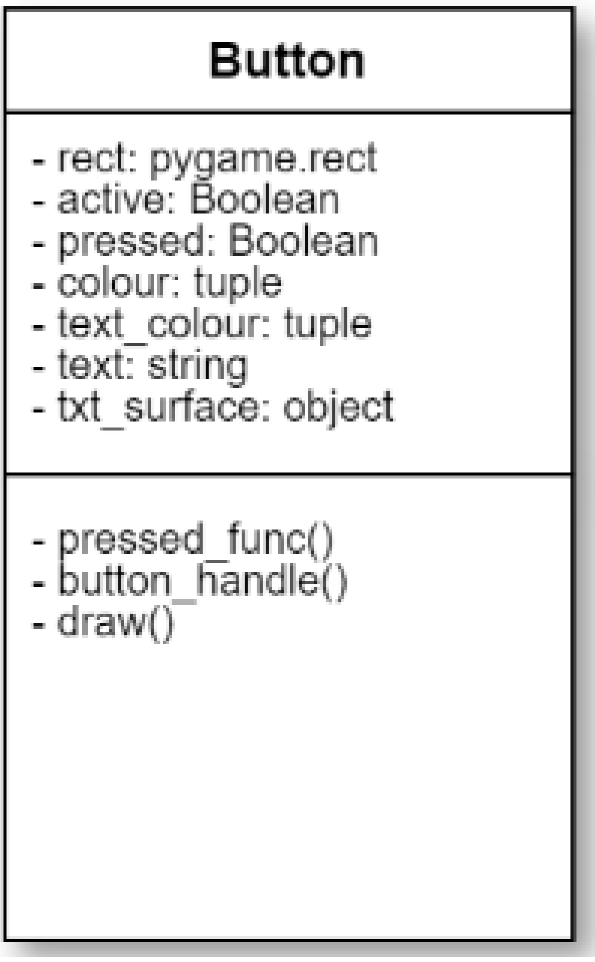
Finally, there is the draw function which simply draws the input box onto the screen.



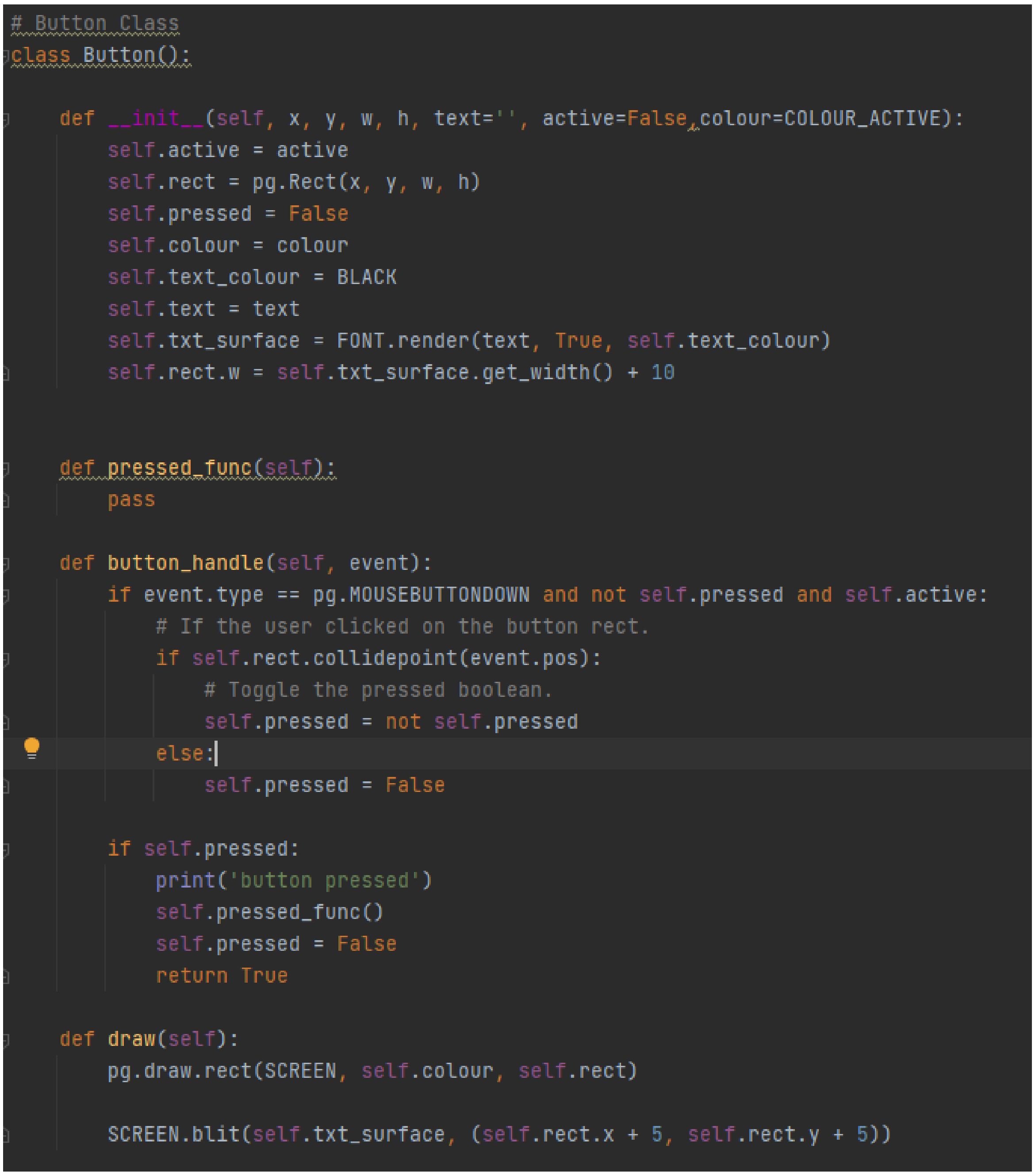
```
def draw(self, screen):
    # Blit the text.
    screen.blit(self.txt_surface, (self.rect.x + 5, self.rect.y + 5))
    # Blit the rect.
    pg.draw.rect(screen, self.colour, self.rect, 2)
```

Button

Class Diagram

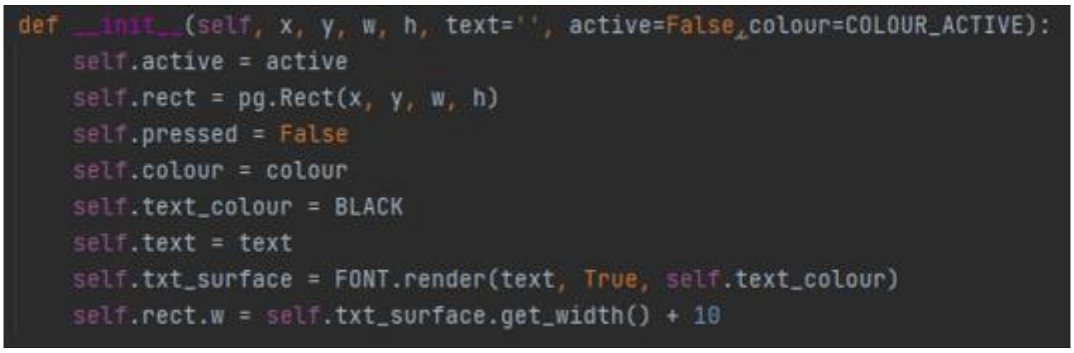


Code


```
# Button Class
class Button():

    def __init__(self, x, y, w, h, text='', active=False,colour=COLOUR_ACTIVE):
        self.active = active
        self.rect = pg.Rect(x, y, w, h)
        self.pressed = False
        self.colour = colour
        self.text_colour = BLACK
        self.text = text
        self.txt_surface = FONT.render(text, True, self.text_colour)
        self.rect.w = self.txt_surface.get_width() + 10


    def pressed_func(self):
        pass

    def button_handle(self, event):
        if event.type == pg.MOUSEBUTTONDOWN and not self.pressed and self.active:
            # If the user clicked on the button rect.
            if self.rect.collidepoint(event.pos):
                # Toggle the pressed boolean.
                self.pressed = not self.pressed
            else:
                self.pressed = False

        if self.pressed:
            print('button pressed')
            self.pressed_func()
            self.pressed = False
            return True

    def draw(self):
        pg.draw.rect(SCREEN, self.colour, self.rect)

        SCREEN.blit(self.txt_surface, (self.rect.x + 5, self.rect.y + 5))
```

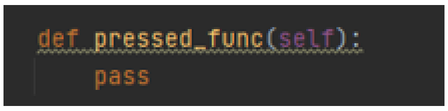
This is the general class for a button.

First, we declare all the variables we need.



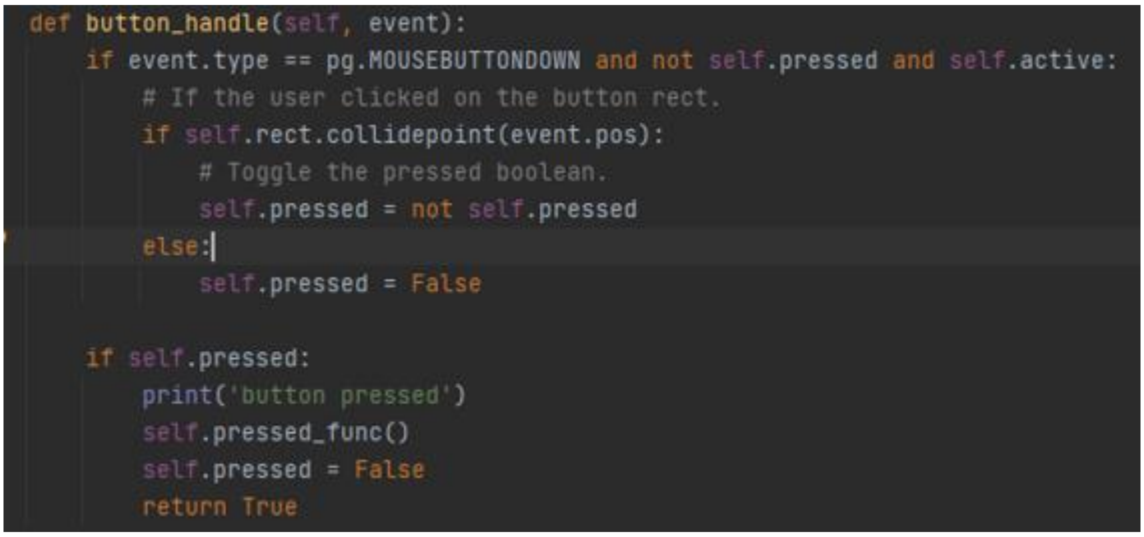
```python
def __init__(self, x, y, w, h, text='', active=False,colour=COLOUR_ACTIVE):
    self.active = active
    self.rect = pg.Rect(x, y, w, h)
    self.pressed = False
    self.colour = colour
    self.text_colour = BLACK
    self.text = text
    self.txt_surface = FONT.render(text, True, self.text_colour)
    self.rect.w = self.txt_surface.get_width() + 10
```

We then have the pressed func function. This is the function that we change when we instantiate a new button class and this runs when the button is pressed.



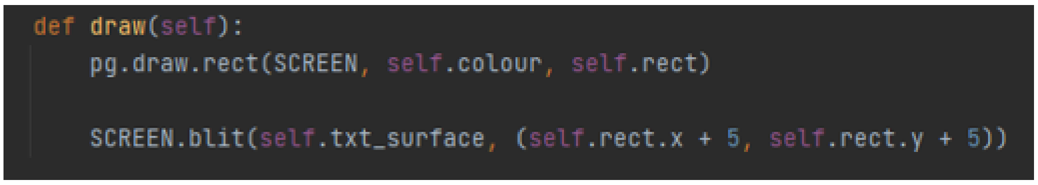
```python
def pressed_func(self):
    pass
```

Next, we have the button handle function which first checks if the button has been pressed and if so runs the pressed_func() function.



```python
def button_handle(self, event):
    if event.type == pg.MOUSEBUTTONDOWN and not self.pressed and self.active:
        # If the user clicked on the button rect.
        if self.rect.collidepoint(event.pos):
            # Toggle the pressed boolean.
            self.pressed = not self.pressed
        else:
            self.pressed = False

    if self.pressed:
        print('button pressed')
        self.pressed_func()
        self.pressed = False
        return True
```

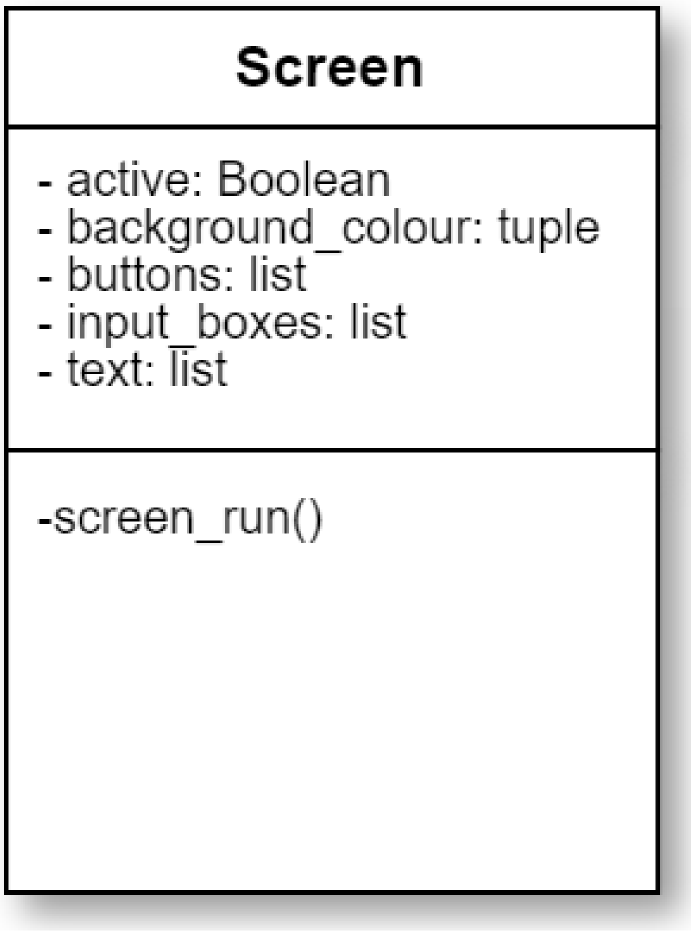
Finally, we have the draw function which draws the button onto the screen.



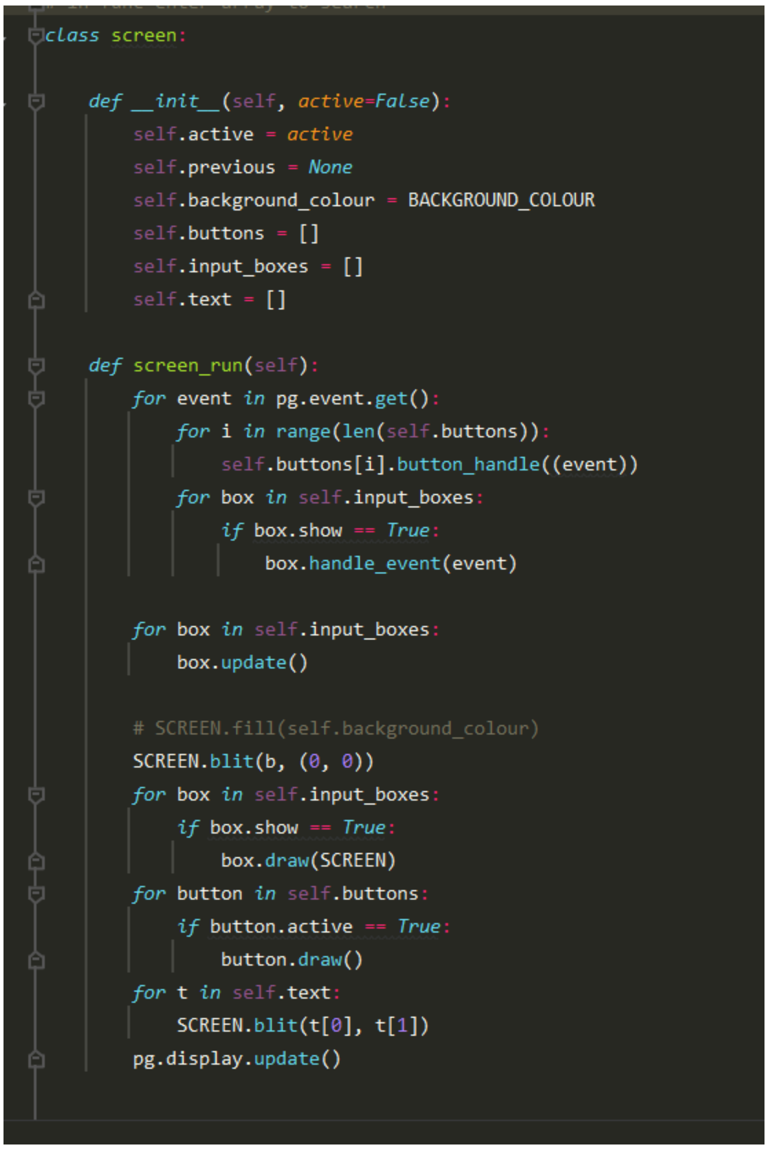
```python
def draw(self):
    pg.draw.rect(SCREEN, self.colour, self.rect)

    SCREEN.blit(self.txt_surface, (self.rect.x + 5, self.rect.y + 5))
```

Screen

Class Diagram



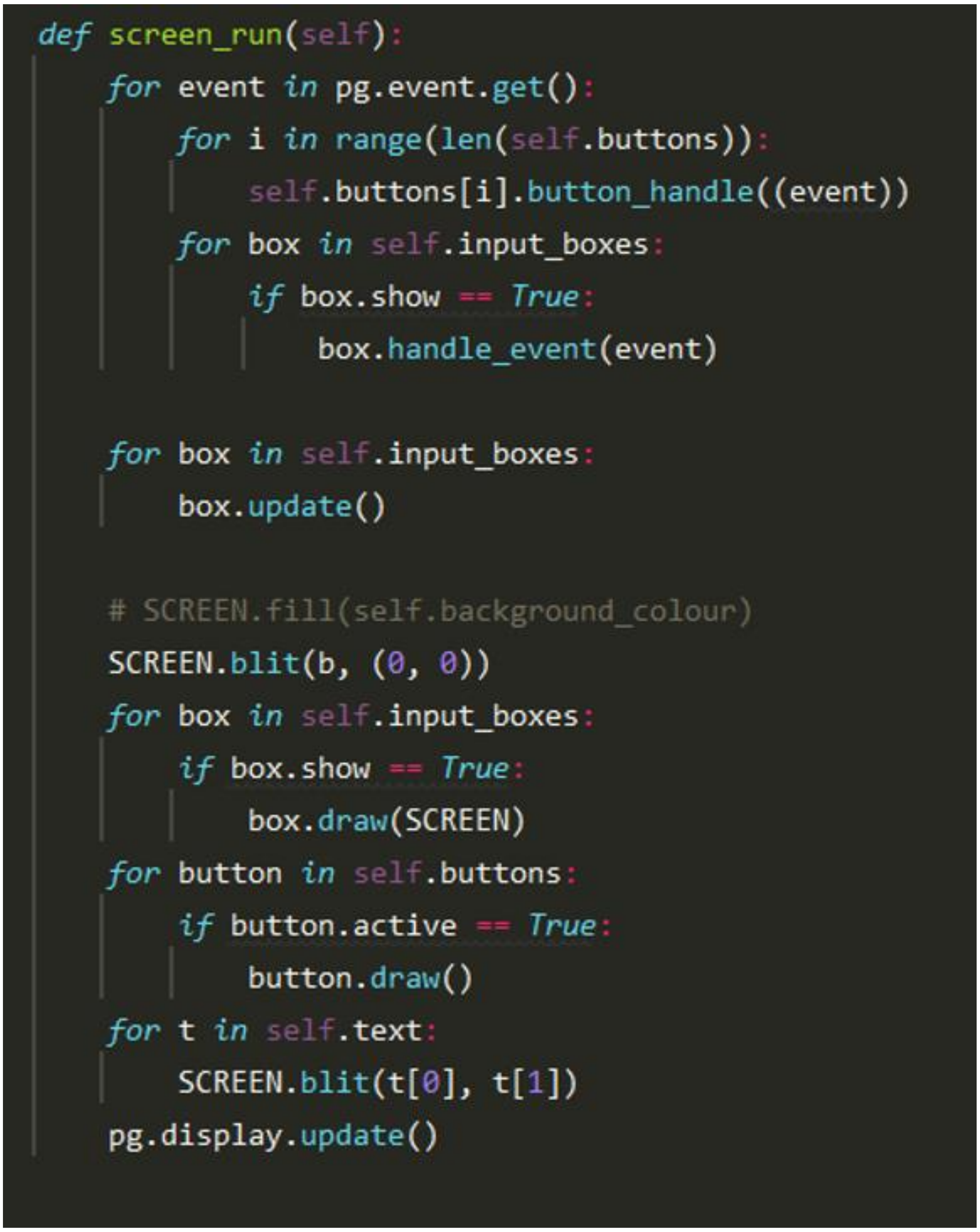
Code



```python
class screen:

    def __init__(self, active=False):
        self.active = active
        self.previous = None
        self.background_colour = BACKGROUND_COLOUR
        self.buttons = []
        self.input_boxes = []
        self.text = []

    def screen_run(self):
        for event in pg.event.get():
            for i in range(len(self.buttons)):
                self.buttons[i].button_handle((event))
            for box in self.input_boxes:
                if box.show == True:
                    box.handle_event(event)

        for box in self.input_boxes:
            box.update()

        # SCREEN.fill(self.background_colour)
        SCREEN.blit(b, (0, 0))
        for box in self.input_boxes:
            if box.show == True:
                box.draw(SCREEN)
        for button in self.buttons:
            if button.active == True:
                button.draw()
        for t in self.text:
            SCREEN.blit(t[0], t[1])
        pg.display.update()
```

This is the main class that all the different screens inherit from.

It is made of one main function named screen_run.



```python
def screen_run(self):
    for event in pg.event.get():
        for i in range(len(self.buttons)):
            self.buttons[i].button_handle((event))
        for box in self.input_boxes:
            if box.show == True:
                box.handle_event(event)

    for box in self.input_boxes:
        box.update()

    # SCREEN.fill(self.background_colour)
    SCREEN.blit(b, (0, 0))
    for box in self.input_boxes:
        if box.show == True:
            box.draw(SCREEN)
    for button in self.buttons:
        if button.active == True:
            button.draw()
    for t in self.text:
        SCREEN.blit(t[0], t[1])
    pg.display.update()
```

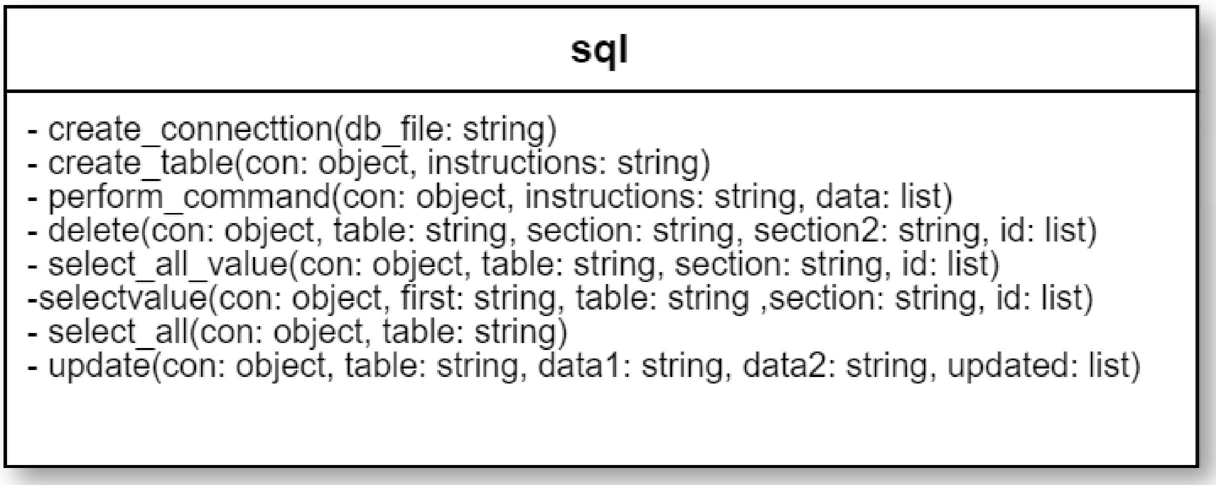
This function first goes through all the buttons in the buttons list of the screen and runs their handle function. It then does the same for all the input boxes.

Next it updates the size of the input box depending on the length of the text it contains.

It then blits the background which is stored in the variable ‘b’, followed by all the active input boxes, buttons and text to the screen.

Sql class

Class diagram

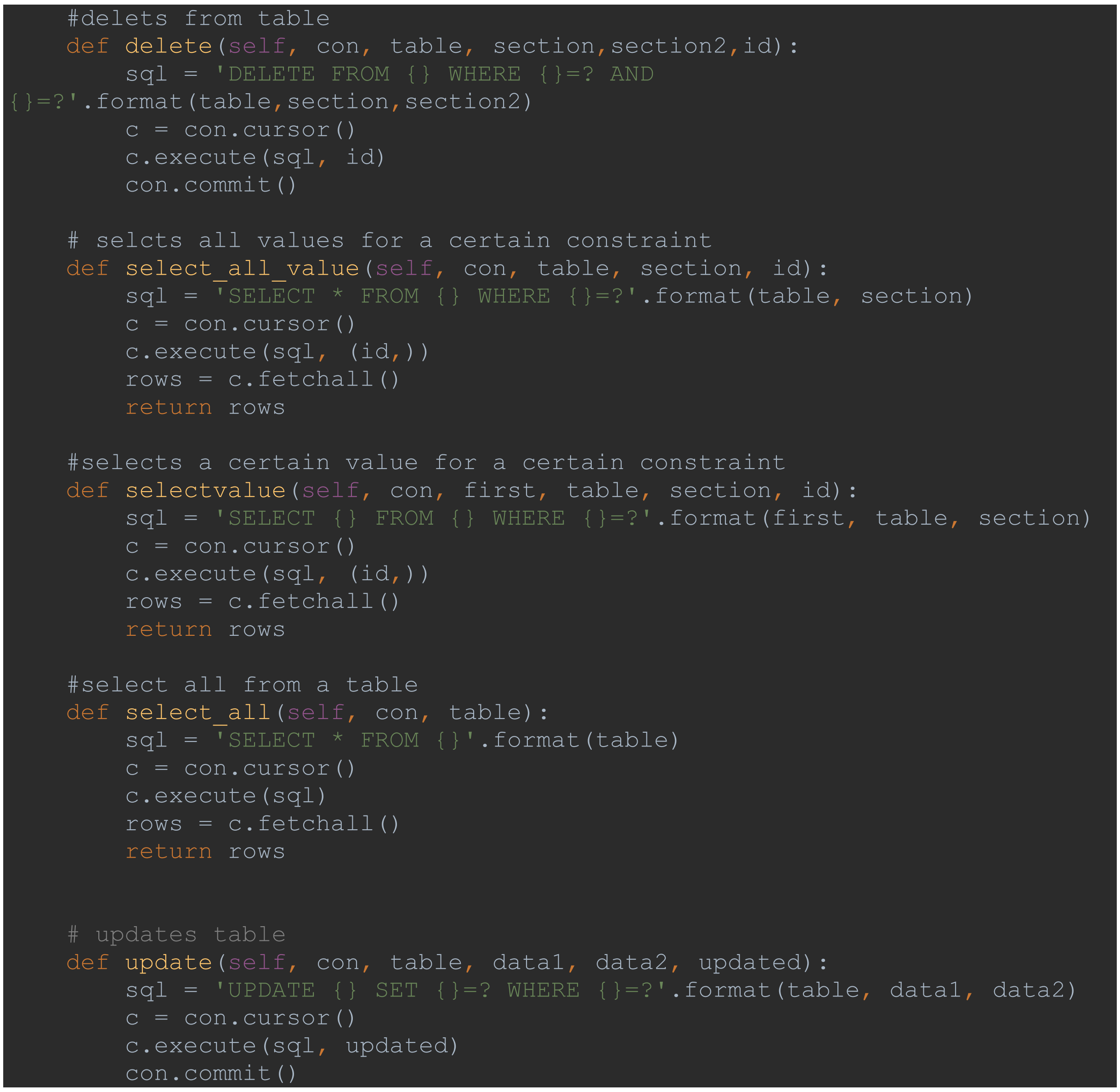


```
sql

- create_connecttion(db_file: string)
- create_table(con: object, instructions: string)
- perform_command(con: object, instructions: string, data: list)
- delete(con: object, table: string, section: string, section2: string, id: list)
- select_all_value(con: object, table: string, section: string, id: list)
-selectvalue(con: object, first: string, table: string ,section: string, id: list)
- select_all(con: object, table: string)
- update(con: object, table: string, data1: string, data2: string, updated: list)
```

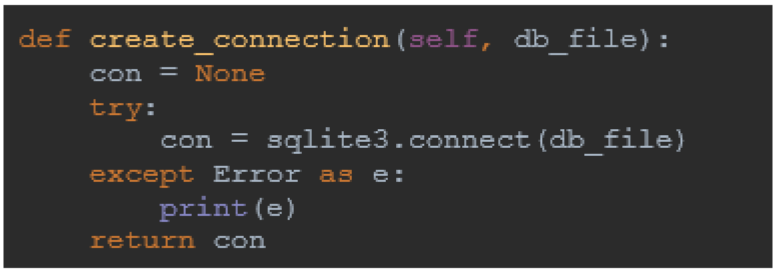
Code

This is the sql class that all database connections run through.

```python
class sql:
    def __init__(self):
        pass


    def create_connection(self, db_file):
        con = None
        try:
            con = sqlite3.connect(db_file)
        except Error as e:
            print(e)
        return con

    # use create table if not exists
    def create_table(self, con, instructions):
        try:
            c = con.cursor()
            c.execute(instructions)
        except Error as e:
            print(e)

    #performs command give via instructions
    def perform_command(self, con, instructions, data):
        c = con.cursor()
        c.execute(instructions, data)
        con.commit()
```

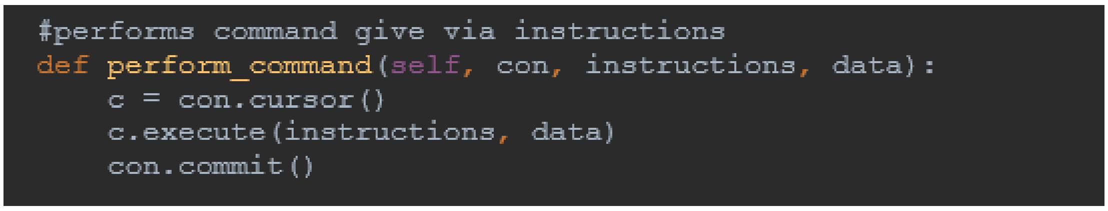


```
    #delets from table
    def delete(self, con, table, section,section2,id):
        sql = 'DELETE FROM {} WHERE {}=? AND
{}=?'.format(table,section,section2)
        c = con.cursor()
        c.execute(sql, id)
        con.commit()

    # selcts all values for a certain constraint
    def select_all_value(self, con, table, section, id):
        sql = 'SELECT * FROM {} WHERE {}=?'.format(table, section)
        c = con.cursor()
        c.execute(sql, (id,))
        rows = c.fetchall()
        return rows

    #selects a certain value for a certain constraint
    def selectvalue(self, con, first, table, section, id):
        sql = 'SELECT {} FROM {} WHERE {}=?'.format(first, table, section)
        c = con.cursor()
        c.execute(sql, (id,))
        rows = c.fetchall()
        return rows

    #select all from a table
    def select_all(self, con, table):
        sql = 'SELECT * FROM {}'.format(table)
        c = con.cursor()
        c.execute(sql)
        rows = c.fetchall()
        return rows


    # updates table
    def update(self, con, table, data1, data2, updated):
        sql = 'UPDATE {} SET {}=? WHERE {}=?'.format(table, data1, data2)
        c = con.cursor()
        c.execute(sql, updated)
        con.commit()
```



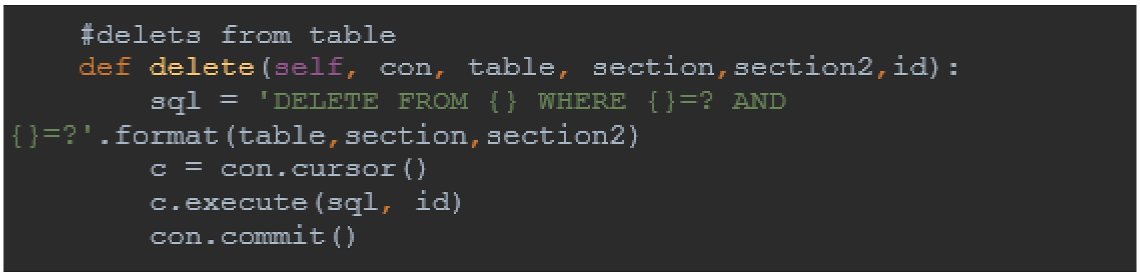
```
def create_connection(self, db_file):
    con = None
    try:
        con = sqlite3.connect(db_file)
    except Error as e:
        print(e)
    return con
```

This function attempts to create a connection to the database with file name as the variable db_file



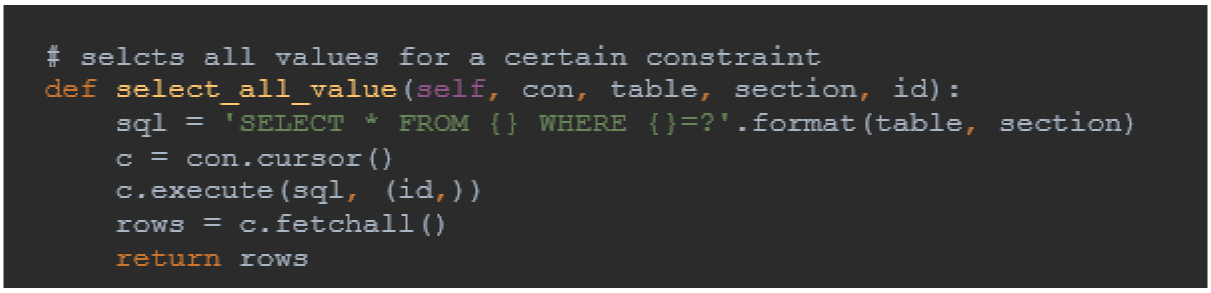
```
#performs command give via instructions
def perform_command(self, con, instructions, data):
    c = con.cursor()
    c.execute(instructions, data)
    con.commit()
```

This function performs the command as given in the instruction's variable. It executes this command along with the data given alongside it.



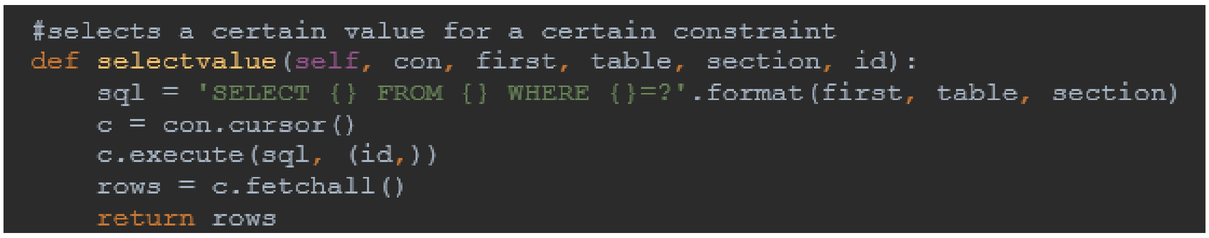
```
    #delets from table
    def delete(self, con, table, section,section2,id):
        sql = 'DELETE FROM {} WHERE {}=? AND
{}=?'.format(table,section,section2)
        c = con.cursor()
        c.execute(sql, id)
        con.commit()
```

This function performs the delete command on the specified table where section is equal to the first number in the array id, and section2 equals the second number in the array id.



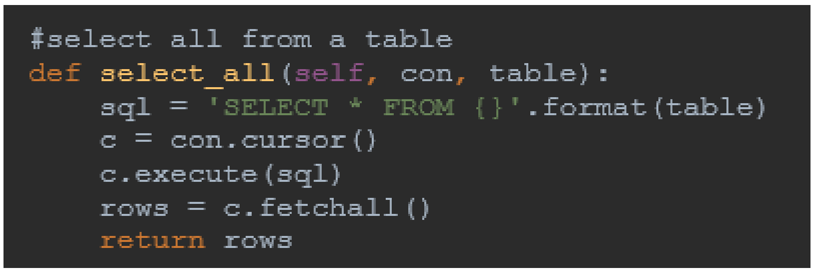
```
# selcts all values for a certain constraint
def select_all_value(self, con, table, section, id):
    sql = 'SELECT * FROM {} WHERE {}=?'.format(table, section)
    c = con.cursor()
    c.execute(sql, (id,))
    rows = c.fetchall()
    return rows
```

This function selects all the values from table where variable section = the variable id



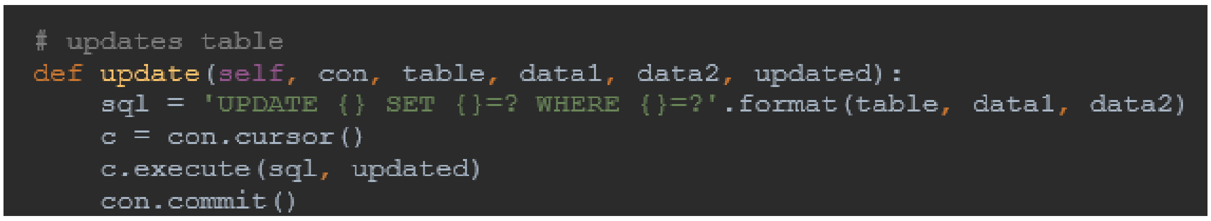
```
#selects a certain value for a certain constraint
def selectvalue(self, con, first, table, section, id):
    sql = 'SELECT {} FROM {} WHERE {}=?'.format(first, table, section)
    c = con.cursor()
    c.execute(sql, (id,))
    rows = c.fetchall()
    return rows
```

This function selects the value in variable first from table where section = id



```python
#select all from a table
def select_all(self, con, table):
    sql = 'SELECT * FROM {}'.format(table)
    c = con.cursor()
    c.execute(sql)
    rows = c.fetchall()
    return rows
```

This function selects all the values from a table.



```python
# updates table
def update(self, con, table, data1, data2, updated):
    sql = 'UPDATE {} SET {}=? WHERE {}=?'.format(table, data1, data2)
    c = con.cursor()
    c.execute(sql, updated)
    con.commit()
```

This function updates table, setting data1 = the first value in the array updated where data2 = the second value in updated

UI

When creating my UI, I decided I wanted a very simple yet effective look.

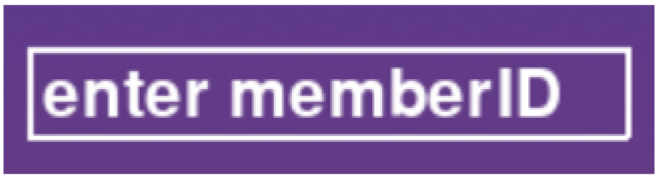
Background

I tried making the background a simple screen filled with one colour however this looked very mundane. I then came up with the idea to mix two colours to add a little complexity without losing the clean and simple affect.

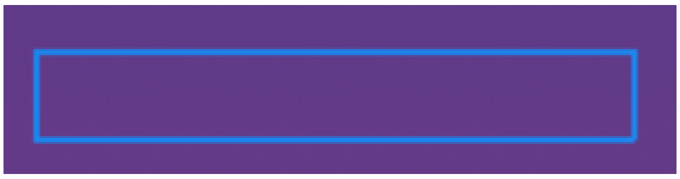
This is what I came up with as an image that I can blit to the screen.

Textboxes

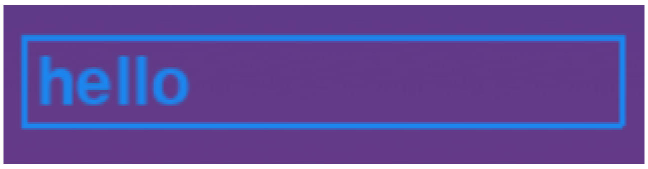
When creating the look for the Input box I wanted to create a way in which users would know whether they had clicked on the Input box in order to type or not. Due to this I came up with the idea to change the colour of the outline of the Input box when the user clicks on it. This is what I came up with.



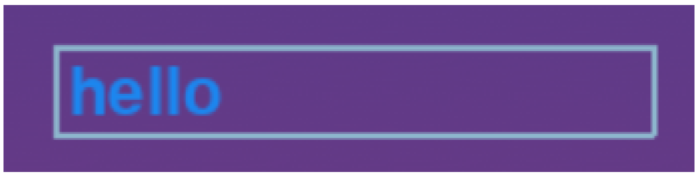
This is how an input box will first look before any actions have taken place, the text will disappear as soon as you start typing. The border is white when no changes have been made.



After clicking within the box the border will change to a blue showing the user that they have clicked the textbox and that they can now type.



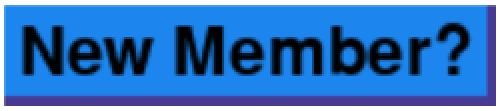
As you type the border will stay blue.



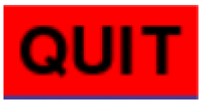
After clicking away from the Input box the colour of the outline changes to a lighter colour displaying that this Input box is no longer active.

Buttons

The look of the button is a very simple one however I wanted to have different colours of buttons depending on what the button does. Therefore, for a standard button I kept with the colour scheme of blue. I use a simple pygame Rect to create this.



However, with some specific buttons I have used different colours to make them stand out more, for example the quit button which I have made red.

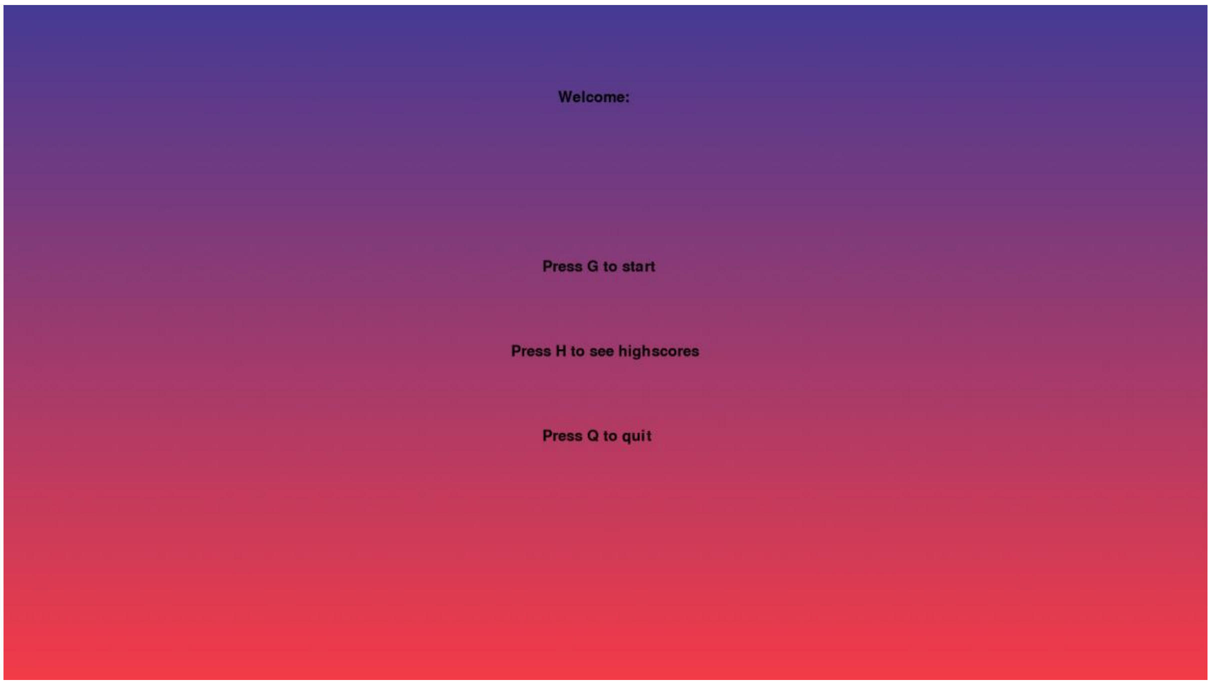


Game

Menu and high score screen

For the menu and high score screen I went with the same theme as the rest of the program so it would not look out of place. I wanted it to look very simple so I could cater to the younger demographic.

Here is what I came up with for these screens.

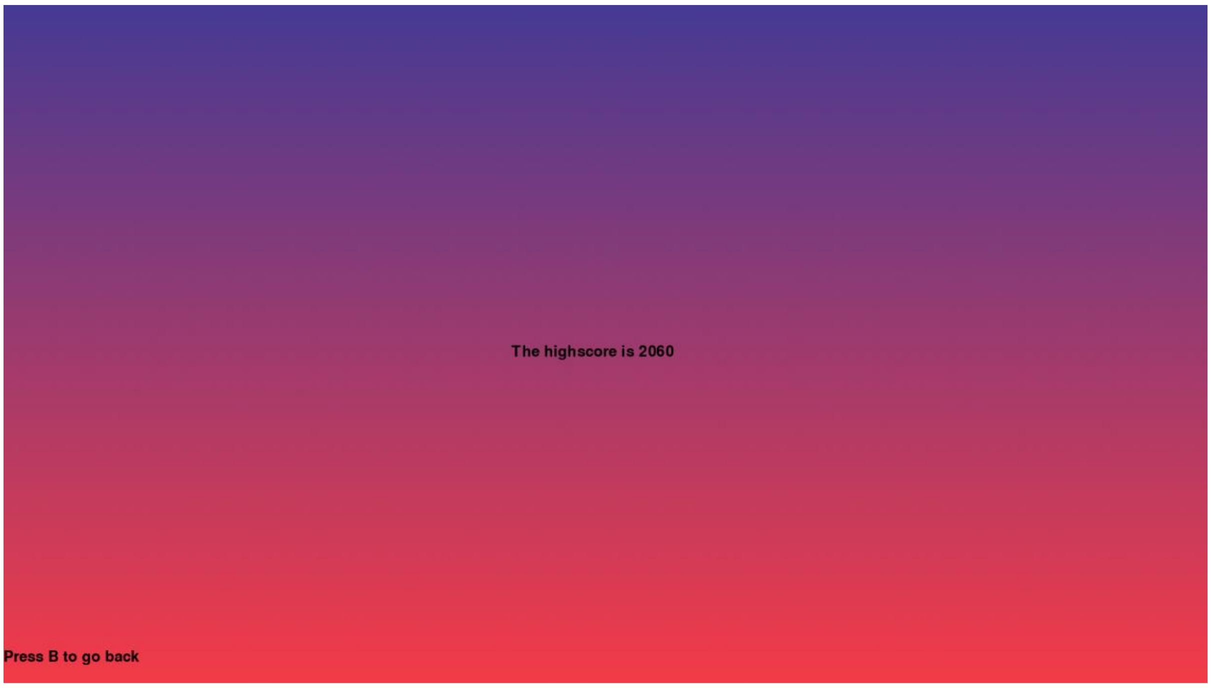

Welcome:

Press G to start

Press H to see highscores

Press Q to quit

(this is an example high score)


The highscore is 2060

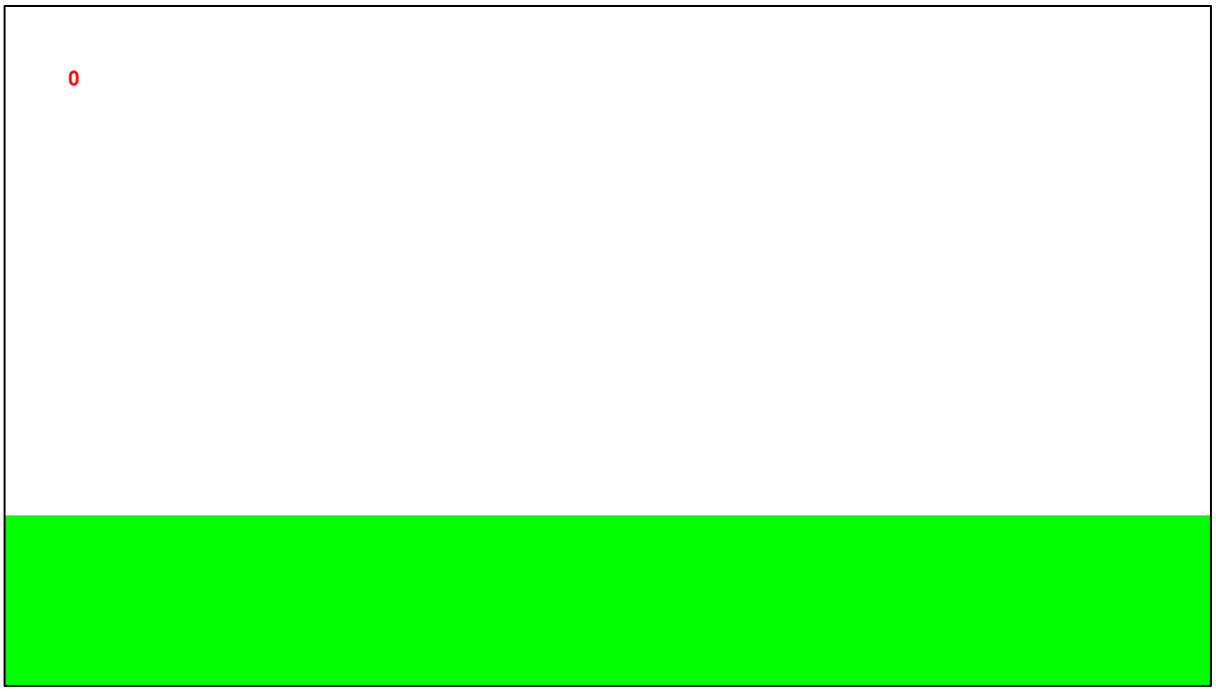
Press B to go back

Game Screen and objects

Screen

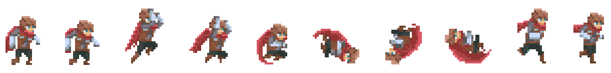
For the game screen I wanted something a bit different and so I went with a pure white screen with a green platform as a grass type surface. Here is what I came up with

(I have created a black border around this picture in order to allow the screen to be more visible)



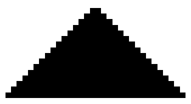
Character

For the character I used a set of images that I found royalty free on the internet. I used these images to allow me to animate the character to make it look like he is running and doing a flip when he jumps. Here are the sprites used.

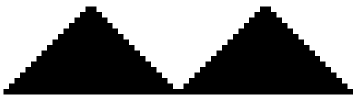


Spikes

For the bad blocks that kill the character I choose to use spikes. I drew these spikes on paint and they look like this.



There is also a double spike to challenge the player a little more which looks like this.



Database

I am using SQLite 3 in python to form a SQL database.

Table: Members

Column	Data type	Description
Id	Integer	Primary key for members. Foreign key for: class, register, payrate
Password	Text	Unique password stored as a hash
First_name	Text	First name of member
Last_name	Text	Last name of member
Type	Text	Type of member: gymnast, coach or admin
DOB	Date	Date of Birth of member
Date_joined	Date	Date joined of member
Mobile	Text	Mobile number of member
Medical	Text	Any medical information about member
Postcode	Text	Postcode of member
Gender	Text	Gender of member
Badges	Text	Contains all badges members own

Table: class_details

Column	**Data type**	**Description**
Classid	Integer	Primary key for class_details. Foreign key for: class, register
Name	Text	Name of the class
No_days	Text	Number of days in the class
MaxKidsInClass	Integer	Max number of kids allowed in the class
kidsInClass	Integer	Number of kids currently in the class
Days_times	Text	The days and time per day that the class runs

Table: class

Column	**Data type**	**Description**
Memberid	Integer	Foreign key from table: members
Classid	Integer	Foreign key from table: class_details

Table: register

Column	**Data type**	**Description**
Memberid	Integer	Foreign key from table: members
Classid	Integer	Foreign key from table: class_details
Day	Text	Day the register is taken on
Present	Integer	Marks a 0 if absent or a 1 if present

Table: payrate

Column	Data type	Description
id	Integer	Foreign key from table: members
pay	real	Rate of pay per hour for staff member

Hashing

I have decided to store all passwords in the database as hash numbers. This provides addition layers of security as even if someone where to get their hands on the data, they would still not be able to decrypt it.

I have opted to use pythons prebuilt hashing function as I believe this is the most efficient and safest algorithm to sue. Therefore, I am using the sha 256 hash from the library hashlib.

Technical solution

main.py

```python
# NEA Project: Gymnastics Membership System
# Name: Finlay Gray



# import statements
import pygame as pg
import sys
import sqlite3
from sqlite3 import Error
import re
import hashlib
import socket
import threading
import pickle
import os

# initialise pygame
pg.init()

# Setting up variables
SCR_W, SCR_H = pg.display.Info().current_w, pg.display.Info().current_h
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
BACKGROUND_COLOUR = (146, 168, 209)
FONT = pg.font.Font(None, 32)
b = pg.image.load('test_back.png')
pic = pg.transform.scale(b, (SCR_W, SCR_H))
font = pg.font.Font('freesansbold.ttf', 32)

COLOUR_INACTIVE = pg.Color('lightskyblue3')
COLOUR_ACTIVE = pg.Color('dodgerblue2')


# regular expression for password check
def password_check(password):
    valid = False
    length_check = re.search('.{8}', password)
    if length_check:
        upper_case_check = re.search('[A-Z]', password)
        if upper_case_check:
            special_check = re.search('\W', password)
            if special_check:
                number_check = re.search('[0-9]', password)
                if number_check:
                    valid = True

    return valid


# Setting up the screen

SCREEN = pg.display.set_mode((SCR_W, SCR_H))


# create sql class which all sql queries run through
class sql:
    def __init__(self):
        pass

    def create_connection(self, db_file):
        con = None
        try:
            con = sqlite3.connect(db_file)
        except Error as e:
            print(e)
        return con

    # use create table if not exists
    def create_table(self, con, instructions):
        try:
            c = con.cursor()
            c.execute(instructions)
        except Error as e:
            print(e)

    # perform a sql command given
    def perform_command(self, con, instructions, data):
        c = con.cursor()
        c.execute(instructions, data)
        con.commit()

    def delete(self, con, table, section, section2, id):
        sql = 'DELETE FROM {} WHERE {}=? AND {}=?'.format(table, section, section2)
        c = con.cursor()
        c.execute(sql, id)
        con.commit()

    # select all values given a certain criteria
    def select_all_value(self, con, table, section, id):
        sql = 'SELECT * FROM {} WHERE {}=?'.format(table, section)
        c = con.cursor()
        c.execute(sql, (id,))
        rows = c.fetchall()
        return rows

    # select a certain value (first) given a certain criteria (section)
    def selectvalue(self, con, first, table, section, id):
        sql = 'SELECT {} FROM {} WHERE {}=?'.format(first, table, section)
        c = con.cursor()
        c.execute(sql, (id,))
        rows = c.fetchall()
        return rows

    def select_all(self, con, table):
        sql = 'SELECT * FROM {}'.format(table)
        c = con.cursor()
        c.execute(sql)
        rows = c.fetchall()
        return rows
    def update(self, con, table, data1, data2, updated):
        sql = 'UPDATE {} SET {}=? WHERE {}=?'.format(table, data1, data2)
        c = con.cursor()
        c.execute(sql, updated)
        con.commit()


# instantiate sql class and set up tables
db = sql()
# create first connection with file
con = db.create_connection('file.db')
# create members table
db.create_table(con, '''CREATE TABLE IF NOT EXISTS members(
                        id integer PRIMARY KEY AUTOINCREMENT,
                        password text NOT NULL,
                        first_name text NOT NULL,
                        last_name text NOT NULL,
                        type text NOT NULL,
                        DOB date NOT NULL,
                        joined_date date NOT NULL,
                        mobile text NOT NULL,
                        medical text NOT NULL,
                        postcode text NOT NULL,
                        gender text NOT NULL,
                        badges text
                        )''')

# create class table
db.create_table(con, '''CREATE TABLE IF NOT EXISTS class(
                        memberid integer NOT NULL,
                        classid integer NOT NULL,
                        FOREIGN KEY (memberid) REFERENCES members (id)
                            ON DELETE CASCADE ON UPDATE NO ACTION,
                        FOREIGN KEY (classid) REFERENCES class_details (classid)
                            ON DELETE CASCADE ON UPDATE NO ACTION)''')

# create class_details table
db.create_table(con, '''CREATE TABLE IF NOT EXISTS class_details(
                        classid integer PRIMARY KEY AUTOINCREMENT,
                        name text NOT NULL,
                        no_days text NOT NULL,
                        MaxKidsInClass integer NOT NULL,
                        kidsInClass integer NOT NULL,
                        days_times text NOT NULL
                        )''')
# create register table
db.create_table(con, '''CREATE TABLE IF NOT EXISTS register(
                        memberid integer NOT NULL,
                        classid integer NOT NULL,
                        day text NOT NULL,
                        present integer NOT NULL,
                        FOREIGN KEY (memberid) REFERENCES members (id)
                            ON DELETE CASCADE ON UPDATE NO ACTION,
                        FOREIGN KEY (classid) REFERENCES class_details (classid)
                            ON DELETE CASCADE ON UPDATE NO ACTION)''')
# create payrate table
db.create_table(con, '''CREATE TABLE IF NOT EXISTS payrate(
                        id integer NOT NULL,
                        pay real NOT NULL,
                        FOREIGN KEY (id) REFERENCES members (id)
                            ON DELETE CASCADE ON UPDATE NO ACTION)''')

# insert automatic admin for first run
try:
    db.perform_command(con, '''INSERT INTO members
VALUES(?,?,?,?,?,?,?,?,?,?,?,?)''', (
        0, 'xxxx', 'admin', 'admin', 'admin', '20-09-2002', '20-05-2020',
'07455469158', 'N/A', 'TW167PN', '', ''))
```

```
except:
    pass

con.close()


# Button Class
class Button:

    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        self.active = active
        self.rect = pg.Rect(x, y, w, h)
        self.pressed = False
        self.colour = colour
        self.text_colour = BLACK
        self.text = text
        self.txt_surface = FONT.render(text, True, self.text_colour)
        self.rect.w = self.txt_surface.get_width() + 10

    def pressed_func(self):
        pass

    def button_handle(self, event):
        if event.type == pg.MOUSEBUTTONDOWN and not self.pressed and self.active:
            # If the user clicked on the button rect.
            if self.rect.collidepoint(event.pos):
                # Toggle the pressed boolean.
                self.pressed = not self.pressed
            else:
                self.pressed = False
        # runs pressed function when the button is clicked on
        if self.pressed:
            self.pressed_func()
            self.pressed = False
            return True

    def draw(self):
        pg.draw.rect(SCREEN, self.colour, self.rect)

        SCREEN.blit(self.txt_surface,
                    (self.rect.x + 5, self.rect.y + (self.rect.h / 2) -
(self.txt_surface.get_height() / 2)))


# submit button for login screen
class Submit_Button(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        memberid = login_page.input_boxes[0].text
        id_check = re.match('[0-9]*', str(memberid))
        # hashes password
        password = hashlib.sha256(login_page.input_boxes[1].text.encode('utf-
8')).hexdigest()
        # checks if id already has password
        if id_check:
            if password == 'xxxx':
                id_check = False
        send = True
        text = []
        if id_check:
            con = db.create_connection('file.db')
            ids = db.select_all_value(con, 'members', 'id', memberid)
            con.close()
            # searches through all ids to see if any match id and password
            if ids:
                for id in ids:
                    if password == id[1]:
                        send = False
                        # sends user to different pages depending on what member
type they are
                        if id[4] == 'gymnast':
                            login_page.active = False
                            gymnast.active = True
                            gymnast.memberid = memberid
                            gymnast_general_screen.days = calender(memberid)
                            for i in range(len(gymnast_general_screen.days)):
                                text = '{}'.format(gymnast_general_screen.days[i])
                                text = text[2:-4]
                                text = font.render(text, True, WHITE)
                                text = [text, (SCR_W * (1 / 8), (SCR_H * ((3) /
16)))]

                                gymnast_general_screen.text.append(text)
                        elif id[4] == 'coach':
                            login_page.active = False
                            coach_base_page.active = True
                            coach_base_page.memberid = memberid
                            coach_base_page.days = calender(memberid)
                            for i in range(len(coach_base_page.days)):
                                text = '{}'.format(coach_base_page.days[i])
                                text = text[2:-4]
                                text = font.render(text, True, WHITE)
                                text = [text, (SCR_W * (1 / 8), (SCR_H * ((i + 3) /
16)))]

                                coach_base_page.text.append(text)
                        elif id[4] == 'admin':
                            login_page.active = False
                            admin_base_page.active = True
        # blits to screen if invalid data is entered
        text = font.render('Enter a vaild ID and password', True, WHITE)
        to_screen = [text, (SCR_W * (6 / 16), SCR_H - 100)]
        if send:
            login_page.text.append(to_screen)

        for i in login_page.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
        pg.display.update()


class Close_Button(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        print('bye bye')
        pg.quit()
        sys.exit()


##Takes user to new member page
class New_Member_Button(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        login_page.active = False
        for i in new_password.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
        new_password.active = True


# Checks if id has password linked to it
class New_Member_Button_submit(Button):
```

```python
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        id = new_password.input_boxes[0].text
        con = db.create_connection('file.db')
        rows = db.select_all_value(con, 'members', 'id', id)
        if rows:
            for row in rows:
                if row[1] == 'xxxx':
                    new_password.active = False
                    for i in new_password1.input_boxes:
                        i.text = i.ori_text
                        i.txt_surface = FONT.render(i.text, True, i.colour)
                    new_password1.active = True

        con.close()


# if password meets requirements updates database with new password
class New_Member_Button_submit2(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        id = new_password.input_boxes[0].text
        con = db.create_connection('file.db')
        rows = db.select_all_value(con, 'members', 'id', id)
        pass1 = new_password1.input_boxes[0].text
        pass2 = new_password1.input_boxes[1].text
        send = False
        text = font.render('You have not entered a valid Password', True, WHITE)

        to_screen = [text, ((SCR_W * (1 / 2)) - (text.get_width() / 2), SCR_H -
100)]
        if pass1 == pass2:
            password = password_check(pass1)
            if password:
                for row in rows:
                    # update function
                    # hashes password so it is stored as a hash
                    p = hashlib.sha256(pass1.encode('utf-8')).hexdigest()

                    db.update(con, 'members', 'password', 'id', (p, id))
                    new_password1.active = False
                    login_page.active = True

            else:
                send = True
        else:
            send = True
        if send:
            new_password1.text.append(to_screen)

        con.close()
        login_page.text = []
        new_password1.text = [new_password1.pass_text_help_to_screen]


# sends user back to screen linked via a back_relationships dictionary
class Back_Button(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        go = False
        for i in back_relationships.keys():
            if i.active == True:
                to_back = i
                go = True
        if go:
            to_back.active = False
            for i in back_relationships[to_back].input_boxes:
                i.text = i.ori_text
                i.txt_surface = FONT.render(i.text, True, i.colour)
            back_relationships[to_back].active = True


# Admin page buttons

# takes user to add member screen
class add_member(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        create_member_screen.active = True
        create_member_screen.submit.active = False
        create_member_screen.next.active = True


# add member button submit

class add_member_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        input_text = []
        for i in create_member_screen.input_boxes:
            input_text.append(i)
        con = db.create_connection('file.db')
        # inserts data from text boxes into table members
        db.perform_command(con,
                           'INSERT INTO
members(first_name,last_name,password,type,DOB,joined_date,mobile,medical,postcode,
gender,badges) VALUES(?,?,?,?,?,?,?,?,?,?,?)',
                           (input_text[0].text, input_text[1].text, 'xxxx',
input_text[2].text, input_text[3].text,
                            input_text[4].text,
                            input_text[5].text, input_text[6].text,
input_text[7].text, input_text[8].text, ','))
        # blits id number just created onto the admin base screen
        c = con.cursor()
        c.execute('SELECT * FROM members ORDER BY id DESC LIMIT 1')
        rows = c.fetchall()
        con.close()
        data = rows[0]

        id_number = data[0]

        text = font.render('Most recent ID number:' + str(id_number), True, WHITE)
        to_screen = [text, (SCR_W * (2 / 6), SCR_H * (1 / 16))]
        admin_base_page.text = []
        admin_base_page.text.append(to_screen)
        for i in create_member_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
            if i.show == True:
                i.show = False
            elif i.show == False:
                i.show = True
        create_member_screen.active = False
        admin_base_page.active = True
```

```
# allows the user to enter more details
class add_member_next(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        for i in create_member_screen.input_boxes:
            if i.show == True:
                i.show = False
            elif i.show == False:
                i.show = True
        create_member_screen.submit.active = True
        create_member_screen.next.active = False


class add_class_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        create_class_screen.active = True
        create_class_screen.next.active = True
        create_class_screen.submit.active = False


# button that sets up input boxes depending on the number of days the class runs.
class add_class_next(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        for i in create_class_screen.input_boxes:

            if i.show == True:
                i.show = False

        no_days = int(create_class_screen.input_boxes[1].text)
        for i in range(1, no_days + 1):
            day = InputBox((SCR_W / 4) - 100, (SCR_H * (i / 8)), 100, 30, 'enter
day', True)
            time = InputBox((SCR_W * (3 / 4)) - 100, (SCR_H * (i / 8)), 100, 30,
'enter time', True)
            create_class_screen.input_boxes.append(day)
            create_class_screen.input_boxes.append(time)
        create_class_screen.next.active = False
        create_class_screen.submit.active = True


# Take all values from input boxes and put into table
class add_class_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):

        day_time = ''
        for i in range(3, 3 + (len(create_class_screen.input_boxes) - 3), 2):
            day = create_class_screen.input_boxes[i].text
            time = create_class_screen.input_boxes[(i + 1)].text
            day_time += day + '/' + time + ','
        con = db.create_connection('file.db')
        db.perform_command(con,
                           '''INSERT INTO
class_details(name,no_days,MaxKidsInClass,kidsInClass,days_times)
VALUES(?,?,?,?,?)''',
                           (create_class_screen.input_boxes[0].text,
create_class_screen.input_boxes[1].text,
                            create_class_screen.input_boxes[2].text, 0, day_time))
        to_remove = []
        con.close()
        for i in create_class_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
            if i.show == True:
                i.show = False
                to_remove.append(i)
            elif i.show == False:
                i.show = True

        # removes all input boxes just created to allow the button to be pressed
again
        for i in to_remove:
            create_class_screen.input_boxes.remove(i)

        # reset
        admin_base_page.active = True
        create_class_screen.active = False


# takes user to the add to class screen
class add_to_class_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        add_to_class_screen.class_id = []
        add_to_class_screen.classes = []
        add_to_class_screen.buttons = [add_to_class_screen.back_button,
add_to_class_screen.close_button,
                                       add_to_class_screen.next]
        add_to_class_screen.active = True


# Button which when pressed adds the member id specified in last screen to class
which name is the same as the name
# of the button
class class_name_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):

        class_name = self.text
        member_Id = add_to_class_screen.id_search.text
        con = db.create_connection('file.db')
        data = db.select_all_value(con, 'class_details', 'name', class_name)
        data = data[0]
        num_kids = data[4]
        # increases the number in the class by 1
        num_kids += 1
        db.update(con, 'class_details', 'kidsInClass', 'classid', (num_kids,
data[0]))
        db.perform_command(con, '''INSERT INTO class VALUES(?,?)''', (member_Id,
data[0]))
        con.close()
        # resets to allow function to be run again
        for i in add_to_class_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
            if i.show == True:
                i.show = False
            elif i.show == False:
                i.show = True
        for i in add_to_class_screen.buttons:
            if i.active == True:
                i.active = False
            elif i.active == False:
                i.active = True

        add_to_class_screen.active = False
        admin_base_page.active = True


# Present button in take register
class yes_button_reg(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        # sets colour green if pressed
        self.colour = GREEN
        # checks if absent button is green, if so then it goes back to original
colour
        for i in range(3, len(register_show_screen.buttons), 2):
            if register_show_screen.buttons[i].colour == GREEN:
                register_show_screen.buttons[i + 1].colour = COLOUR_ACTIVE


# Absent button in take register
class no_button_reg(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        self.colour = GREEN
        for i in range(3, len(register_show_screen.buttons), 2):
            if register_show_screen.buttons[i + 1].colour == GREEN:
                register_show_screen.buttons[i].colour = COLOUR_ACTIVE


# button which allows you to pick the time you want to take the register of.
class pick_day_reg(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        register_screen.buttons = [register_screen.close_button,
register_screen.back_button]
        class_name = self.text
        register_show_screen.class_name = class_name
        classes_screen.active = False
        register_screen.active = True
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'days_times', 'class_details', 'name',
class_name)
        con.close()
        data = data[0]
        data = data[0]
        days = data.split(',')
        days.pop(-1)
        for i in range(len(days)):
            but = classes_reg_but((SCR_W / 2) - 100, ((SCR_H * ((i % 8) / 8)) +
25), 50, 30, days[i], True)
            register_screen.buttons.append(but)
        register_show_screen.text = []
        register_show_screen.links = []
        register_show_screen.buttons = [register_show_screen.close_button,
register_show_screen.back_button,
                                        register_show_screen.submit_button]
```

```
# Prints all the gymnasts in the class out as a list with 2 button linked to each
one to allow them to be present
# or absent

class classes_reg_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        # grabs name of day and name of class
        day = self.text
        register_show_screen.day = day
        class_name = register_show_screen.class_name
        # searches for this classes id
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'classid', 'class_details', 'name', class_name)
        data = data[0]
        data = data[0]
        register_show_screen.classid = data
        register_screen.active = False
        # searches for all member ids with this classid
        ids = []
        ids_raw = db.selectvalue(con, 'memberid', 'class', 'classid', data)

        for i in ids_raw:
            for j in i:
                ids.append(j)
        details = []
        for i in ids:
            data = db.select_all_value(con, 'members', 'id', i)
            details.append(data[0])
        # separates chosen members into gymnasts and coaches
        con.close()
        gymnasts = []
        coaches = []
        for i in details:
            if i[4] == 'gymnast':
                gymnasts.append(i)
            else:
                coaches.append(i)
        count = 1
        font = pg.font.Font('freesansbold.ttf', 16)
        # creates text and 2 buttons for each gymnast
        for i in gymnasts:
            yes = yes_button_reg((SCR_W * 2 / 3), (SCR_H * (count / (SCR_H //
30))), 50, 25, 'Present', True)
            no = no_button_reg((SCR_W * 2 / 3 + 100), (SCR_H * (count / (SCR_H //
30))), 50, 25, 'Absent', True)
            register_show_screen.buttons.append(yes)
            register_show_screen.buttons.append(no)
            stri = i[2] + ' ' + i[3]
            text = font.render(stri, True, WHITE)
            text_to_screen = [text, (SCR_W * (1 / 3), (SCR_H * (count / (SCR_H //
30))))]
            register_show_screen.text.append(text_to_screen)
            register_show_screen.links.append([i[0], yes])

            count += 1

        register_show_screen.active = True


# takes the status of the gymnast as edited when taking register and saves them to
the database
class register_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)
    def pressed_func(self):
        con = db.create_connection('file.db')

        for i in register_show_screen.links:
            try:
                db.delete(con, 'register', 'classid', 'memberid',
(register_show_screen.classid, i[0]))
            except:
                pass
            if i[1].colour == GREEN:
                db.perform_command(con, '''INSERT INTO register VALUES(?,?,?,?)''',
                                   (i[0], register_show_screen.classid,
register_show_screen.day, 1))
            else:
                db.perform_command(con, '''INSERT INTO register VALUES(?,?,?,?)''',
                                   (i[0], register_show_screen.classid,
register_show_screen.day, 0))
        con.close()
        register_show_screen.active = False
        back_relationships[classes_screen].active = True


# allows you to pick which class you want to view
class view_register(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        if admin_base_page.active == True:
            back_relationships[reg_view_1_screen] = admin_base_page
            admin_base_page.active = False
        elif coach_base_page.active == True:
            back_relationships[reg_view_1_screen] = coach_base_page
            coach_base_page.active = False
        reg_view_1_screen.active = True
        reg_view_2_screen.buttons = [reg_view_2_screen.back_button,
reg_view_2_screen.close_button]
        con = db.create_connection('file.db')
        rows = db.select_all(con, 'class_details')
        con.close()
        for row in rows:
            reg_view_1_screen.classes.append(row)
        width = 0
        for i in range(len(reg_view_1_screen.classes)):
            if i % 8 == 0:
                width += 1 / 4

            name = reg_view_1_screen.classes[i][1]

            but = pick_day_view((SCR_W * width) - 100, ((SCR_H * ((i % 8) / 8)) +
25), 50, 30, name, True)
            reg_view_1_screen.buttons.append(but)

        reg_view_1_screen.classes = []


# allows you to pick which day you want to view
class pick_day_view(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        reg_view_1_screen.buttons = [reg_view_1_screen.close_button,
reg_view_1_screen.back_button]
        class_name = self.text
        reg_view_1_screen.class_name = class_name
        reg_view_1_screen.active = False
        reg_view_2_screen.active = True
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'days_times', 'class_details', 'name',
class_name)
        con.close()
        data = data[0]
        data = data[0]
        days = data.split(',')
        days.pop(-1)
        for i in range(len(days)):
            but = view_reg_final((SCR_W / 2) - 100, ((SCR_H * ((i % 8) / 8)) + 25),
50, 30, days[i], True)
            reg_view_2_screen.buttons.append(but)
        reg_view_1_screen.text = []
        reg_view_2_screen.text = []
        reg_view_1_screen.buttons = [reg_view_1_screen.close_button,
reg_view_1_screen.back_button]


# Prints the gymnasts to the screen if a register has been taken as well and
whether they were present or absent.
class view_reg_final(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        day = self.text
        reg_view_2_screen.day = day
        class_name = reg_view_1_screen.class_name
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'classid', 'class_details', 'name', class_name)
        data = data[0]
        data = data[0]
        reg_view_1_screen.classid = data
        ids = []
        ids_raw = db.selectvalue(con, 'memberid', 'class', 'classid', data)
        for i in range(2, len(reg_view_2_screen.buttons)):
            reg_view_2_screen.buttons[i].active = False

        for i in ids_raw:
            for j in i:
                ids.append(j)
        details = []
        for i in ids:
            data = db.select_all_value(con, 'members', 'id', i)
            details.append(data[0])
        gymnasts = []
        coaches = []
        for i in details:
            if i[4] == 'gymnast':
                gymnasts.append(i)
            else:
                coaches.append(i)

        count = 1
        font = pg.font.Font('freesansbold.ttf', 16)
        for i in gymnasts:
            stri = i[2] + ' ' + i[3]
            text = font.render(stri, True, WHITE)
            text_to_screen = [text, (SCR_W * (1 / 3), (SCR_H * (count / (SCR_H //
30))))]
            reg_view_2_screen.text.append(text_to_screen)
            data = db.select_all_value(con, 'register', 'memberid', i[0])
            for i in data:
                if i[1] == reg_view_1_screen.classid:
                    if day in i[2]:
                        if i[3] == 1:
                            text = font.render('Present', True, WHITE)
                        elif i[3] == 0:
                            text = font.render('Absent', True, WHITE)
                        text_to_screen = [text, (SCR_W * (2 / 3), (SCR_H * (count /
(SCR_H // 30))))]
                        reg_view_2_screen.text.append(text_to_screen)

            count += 1

        con.close()


# checks all the possible classes the id could be aadded to and puts them on teh
screen as buttons
class add_to_class_id_next(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        con = db.create_connection('file.db')
        rows = db.select_all(con, 'class_details')
        con.close()
        for row in rows:
            if row[4] < row[3]:
                add_to_class_screen.classes.append(row)
        for i in add_to_class_screen.input_boxes:
            if i.show == True:
                i.show = False
            elif i.show == False:
                i.show = True
        width = 0
        for i in range(len(add_to_class_screen.classes)):
            if i % 8 == 0:
                width += 1 / 4

            name = add_to_class_screen.classes[i][1]

            if width % 1 == 0:
                but = class_name_but((SCR_W * width) - 100, ((SCR_H * ((i % 8) /
8)) + 25), 50, 30, name, False)
            else:

                but = class_name_but((SCR_W * width) - 100, ((SCR_H * ((i % 8) /
8)) + 25), 50, 30, name, True)
            add_to_class_screen.buttons.append(but)

        add_to_class_screen.next.active = False


# puts all the classes that can be registered on the screen as buttons
class classes_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        if admin_base_page.active == True:
            back_relationships[classes_screen] = admin_base_page
            admin_base_page.active = False
        elif coach_base_page.active == True:
            back_relationships[classes_screen] = coach_base_page
            coach_base_page.active = False
        classes_screen.active = True
        con = db.create_connection('file.db')
        rows = db.select_all(con, 'class_details')
        con.close()
        for row in rows:
            classes_screen.classes.append(row)
        width = 0
        for i in range(len(classes_screen.classes)):
            if i % 8 == 0:
                width += 1 / 4

            name = classes_screen.classes[i][1]

            if width % 1 == 0:
                but = pick_day_reg((SCR_W * width) - 100, ((SCR_H * ((i % 8) / 8))
+ 25), 50, 30, name, False)
                classes_screen.next_screen.append(but)
            else:
                but = pick_day_reg((SCR_W * width) - 100, ((SCR_H * ((i % 8) / 8))
+ 25), 50, 30, name, True)
            classes_screen.buttons.append(but)

        classes_screen.classes = []


# puts all the classes that can be edited on the screen as buttons
class edit_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        edit_first_screen.active = True
        con = db.create_connection('file.db')
        rows = db.select_all(con, 'class_details')
        con.close()
        for row in rows:
            edit_first_screen.classes.append(row)
        width = 0
        for i in range(len(edit_first_screen.classes)):
            if i % 8 == 0:
                width += 1 / 4

            name = edit_first_screen.classes[i][1]

            if width % 1 == 0:

                but = class_choose((SCR_W * width) - 100, ((SCR_H * ((i % 8) / 8))
+ 25), 50, 30, name, False)

            else:

                but = class_choose((SCR_W * width) - 100, ((SCR_H * ((i % 8) / 8))
+ 25), 50, 30, name, True)
            edit_first_screen.buttons.append(but)

        edit_first_screen.classes = []


# saves the class id and name of the class you want to edit
class class_choose(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        class_name = self.text
        edit_second_screen.name = class_name
        edit_first_screen.active = False
        edit_second_screen.active = True
        con = db.create_connection('file.db')
        class_id = db.selectvalue(con, 'classid', 'class_details', 'name',
edit_second_screen.name)
        edit_second_screen.classid = class_id[0][0]
        con.close()


# Takes you to the edit class details screen
```

```
class class_dets_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        edit_second_screen.active = False
        edit_class_dets_screen.active = True
        con = db.create_connection('file.db')
        data = db.select_all_value(con, 'class_details', 'name',
edit_second_screen.name)
        con.close()


# puts all members in the class whether they are gymnast or coaches on the screen
with a remove button
class edit_class_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        edit_class_screen.buttons = [edit_class_screen.close_button,
edit_class_screen.back_button,
                                     edit_class_screen.submit]
        edit_second_screen.active = False
        edit_class_screen.active = True
        ids = []
        con = db.create_connection('file.db')
        ids_raw = db.selectvalue(con, 'memberid', 'class', 'classid',
edit_second_screen.classid)
        for i in ids_raw:
            for j in i:
                ids.append(j)
        details = []

        for i in ids:
            data = db.select_all_value(con, 'members', 'id', i)

            details.append(data[0])

        con.close()
        gymnasts = []
        coaches = []
        for i in details:
            if i[4] == 'gymnast':
                gymnasts.append(i)
            else:
                coaches.append(i)
        count = 1
        font = pg.font.Font('freesansbold.ttf', 16)
        for i in coaches:
            remove = remove_member_from_class((SCR_W * 2 / 3), (SCR_H * (count /
(SCR_H // 30))), 50, 25, 'Remove',
                                              True)
            edit_class_screen.buttons.append(remove)
            stri = i[2] + ' ' + i[3]
            text = font.render(stri, True, WHITE)
            text_to_screen = [text, (SCR_W * (1 / 3), (SCR_H * (count / (SCR_H //
30))))]
            edit_class_screen.text.append(text_to_screen)
            edit_class_screen.links.append([i[0], remove])

            count += 1
        for i in gymnasts:
            remove = remove_member_from_class((SCR_W * 2 / 3), (SCR_H * (count /
(SCR_H // 30))), 50, 25, 'Remove',
                                              True)
            edit_class_screen.buttons.append(remove)
            stri = i[2] + ' ' + i[3]
            text = font.render(stri, True, WHITE)
            text_to_screen = [text, (SCR_W * (1 / 3), (SCR_H * (count / (SCR_H //
30))))]
            edit_class_screen.text.append(text_to_screen)
            edit_class_screen.links.append([i[0], remove])

            count += 1


# if pressed once, turns red, twice, turn back agiain.
class remove_member_from_class(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        if self.colour == RED:
            self.colour = COLOUR_ACTIVE
        else:
            self.colour = RED


# Takes all users in the class with the remove button linked to them red and
deletes them from the class
class remove_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        con = db.create_connection('file.db')
        for i in edit_class_screen.links:
            if i[1].colour == RED:
                db.delete(con, 'class', 'memberid', 'classid', (i[0],
edit_second_screen.classid))

        edit_class_screen.active = False
        admin_base_page.active = True
        edit_class_screen.text = []
        con.close()


# allows you to edit the days and times the class runs
class edit_class_dets_next(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        for i in edit_class_dets_screen.input_boxes:

            if i.show == True:
                i.show = False

        no_days = int(edit_class_dets_screen.input_boxes[1].text)
        for i in range(1, no_days + 1):
            day = InputBox((SCR_W / 4) - 100, (SCR_H * (i / 8)), 100, 30, 'enter
day', True)
            time = InputBox((SCR_W * (3 / 4)) - 100, (SCR_H * (i / 8)), 100, 30,
'enter time', True)
            edit_class_dets_screen.input_boxes.append(day)
            edit_class_dets_screen.input_boxes.append(time)
        edit_class_dets_screen.next.active = False
        edit_class_dets_screen.submit.active = True


# takes the updates and submits them into the database
class ed_class_dets_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)
```

```python
    def pressed_func(self):

        day_time = ''
        for i in range(3, 3 + (len(edit_class_dets_screen.input_boxes) - 3), 2):
            day = edit_class_dets_screen.input_boxes[i].text
            time = edit_class_dets_screen.input_boxes[(i + 1)].text
            day_time += day + '/' + time + ','
        con = db.create_connection('file.db')
        db.update(con, 'class_details', 'no_days', 'classid',
                  (edit_class_dets_screen.input_boxes[1].text,
edit_second_screen.classid))
        db.update(con, 'class_details', 'name', 'classid',
                  (edit_class_dets_screen.input_boxes[0].text,
edit_second_screen.classid))
        db.update(con, 'class_details', 'MaxKidsInClass', 'classid',
                  (edit_class_dets_screen.input_boxes[2].text,
edit_second_screen.classid))
        db.update(con, 'class_details', 'days_times', 'classid',
                  (day_time, edit_second_screen.classid))
        to_remove = []
        con.close()
        for i in edit_class_dets_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)
            if i.show == True:
                i.show = False
                to_remove.append(i)
            elif i.show == False:
                i.show = True

        for i in to_remove:
            edit_class_dets_screen.input_boxes.remove(i)

        # reset
        admin_base_page.active = True
        edit_class_dets_screen.active = False


# takes you to the set pay screen
class pay_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        pay_screen.active = True


# inserts pay into the database
class pay_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        id = pay_screen.input_boxes[0].text
        pay = pay_screen.input_boxes[1].text
        con = db.create_connection('file.db')
        db.perform_command(con, '''INSERT INTO payrate VALUES(?,?)''', (int(id),
float(pay)))
        con.close()
        pay_screen.active = False
        admin_base_page.active = True
        for i in pay_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)


# takes user to badges screen
```

```python
class badges_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        admin_base_page.active = False
        badge_screen.active = True


# updates members table of specified id number with added badges
class badges_submit(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        id = badge_screen.input_boxes[0].text
        badge = badge_screen.input_boxes[1].text
        con = db.create_connection('file.db')
        value = db.selectvalue(con, 'badges', 'members', 'id', id)
        value = value[0][0]
        string_add = '{} {},'.format(value, badge)
        db.update(con, 'members', 'badges', 'id', (string_add, id))
        con.close()
        badge_screen.active = False
        admin_base_page.active = True
        for i in badge_screen.input_boxes:
            i.text = i.ori_text
            i.txt_surface = FONT.render(i.text, True, i.colour)


# gymnast buttons

# prints all member details onto the member details screen
class member_details_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    # continue
    def pressed_func(self):
        gymnast.active = False
        member_details_screen.active = True
        id = gymnast.memberid
        con = db.create_connection('file.db')
        details = db.select_all_value(con, 'members', 'id', id)
        details = details[0]
        con.close()
        welcome_string = 'Welcome {} {}'.format(details[2], details[3])
        welcome_text = font.render(welcome_string, True, WHITE)
        text_to_screen_1 = [welcome_text, (SCR_W * (1 / 8), (SCR_H * 1 / 16))]
        member_details_screen.text.append(text_to_screen_1)
        id_text = 'ID number: {}'.format(details[0])
        member_text = 'Member Type: {}'.format(details[4])
        dob_text = 'DOB: {}'.format(details[5])
        joined_text = 'Date joined: {}'.format(details[6])
        number_text = 'Phone Number: {}'.format(details[7])
        medical_text = 'Medical Information: {}'.format(details[8])
        postcode_text = 'Postcode: {}'.format(details[9])
        id_text = font.render(id_text, True, WHITE)
        member_text = font.render(member_text, True, WHITE)
        dob_text = font.render(dob_text, True, WHITE)
        joined_text = font.render(joined_text, True, WHITE)
        number_text = font.render(number_text, True, WHITE)
        medical_text = font.render(medical_text, True, WHITE)
        postcode_text = font.render(postcode_text, True, WHITE)

        text_to_screen_2 = [id_text, (SCR_W * (3 / 8), (SCR_H * 1 / 8))]
        text_to_screen_3 = [member_text, (SCR_W * (3 / 8), (SCR_H * 2 / 8))]
        text_to_screen_4 = [dob_text, (SCR_W * (3 / 8), (SCR_H * 3 / 8))]
        text_to_screen_5 = [joined_text, (SCR_W * (3 / 8), (SCR_H * 4 / 8))]
        text_to_screen_6 = [number_text, (SCR_W * (3 / 8), (SCR_H * 5 / 8))]
        text_to_screen_7 = [medical_text, (SCR_W * (3 / 8), (SCR_H * 6 / 8))]
        text_to_screen_8 = [postcode_text, (SCR_W * (3 / 8), (SCR_H * 7 / 8))]
        member_details_screen.text.append(text_to_screen_2)
        member_details_screen.text.append(text_to_screen_3)
        member_details_screen.text.append(text_to_screen_4)
        member_details_screen.text.append(text_to_screen_5)
        member_details_screen.text.append(text_to_screen_6)
        member_details_screen.text.append(text_to_screen_7)
        member_details_screen.text.append(text_to_screen_8)


# takes user to a page in which they can view their class name, coaches, badges and
can press the console chat button
class gymnast_general_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        gymnast.active = False
        gymnast_general_screen.active = True
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'classid', 'class', 'memberid',
gymnast.memberid)
        data = data[0]
        data = data[0]
        dets = db.select_all_value(con, 'class_details', 'classid', data)
        dets = dets[0]
        name = dets[1]
        coaches = []
        all_in_class = db.selectvalue(con, 'memberid', 'class', 'classid', dets[0])
        for i in all_in_class:
            members = db.select_all_value(con, 'members', 'id', i[0])
            for i in members:
                if i[4] == 'coach':
                    coaches.append(i)

                if str(i[0]) == str(gymnast.memberid):
                    badges = i[11]

        try:
            all_badges = badges.split(',')
            for i in range(len(all_badges)):
                try:
                    badges = font.render(all_badges[i], True, WHITE)
                    badges = [badges, (SCR_W * (4 / 8), (SCR_H * (4 + i) / 16))]
                    gymnast_general_screen.text.append(badges)
                except:
                    pass
        except:
            pass
        con.close()

        class_name = 'Class: {}'.format(name)
        class_name_text = font.render(class_name, True, WHITE)
        text_to_screen_1 = [class_name_text, (SCR_W * (1 / 8), (SCR_H * 1 / 16))]
        gymnast_general_screen.text.append(text_to_screen_1)
        coaches_title = 'Coaches:'
        coaches_title = font.render(coaches_title, True, WHITE)
        coaches_title = [coaches_title, (SCR_W * (1 / 8), (SCR_H * 4 / 16))]
        gymnast_general_screen.text.append(coaches_title)
        calender_title = 'Class times:'
        calender_title = font.render(calender_title, True, WHITE)
        calender_title = [calender_title, (SCR_W * (1 / 8), (SCR_H * 2 / 16))]
        gymnast_general_screen.text.append(calender_title)
        badges_title = 'Badges:'
        badges_title = font.render(badges_title, True, WHITE)
        badges_title = [badges_title, (SCR_W * (4 / 8), (SCR_H * 4 / 16))]
        gymnast_general_screen.text.append(badges_title)

        for i in range(len(coaches)):
            text = '{} {}'.format(coaches[i][2], coaches[i][3])
            text = font.render(text, True, WHITE)
            text = [text, (SCR_W * (1 / 8), (SCR_H * ((i + 5) / 16)))]
            gymnast_general_screen.text.append(text)


# runs the game as an external system
class game_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        os.system('python game.py')


# gymnast and coach
# connects to server.py to chat to others
class console_chat_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        chat = True
        client_socket = socket.socket()
        port = 12345
        client_socket.connect(('127.0.0.1', port))
        con = db.create_connection('file.db')
        if gymnast_general_screen.active == True:
            details = db.select_all_value(con, 'members', 'id', gymnast.memberid)
        elif coach_base_page.active == True:
            details = db.select_all_value(con, 'members', 'id',
coach_base_page.memberid)
        details = details[0]
        con.close()
        client_socket.send(pickle.dumps('{} {}'.format(details[2], details[3])))
        recv_msg = client_socket.recv(1024)
        print(pickle.loads(recv_msg))
        data = []

        def get_input():
            data.append(input('enter: '))

        while chat:
            # allows input to be asked for and searching for incoming messages at
the same time
            input_thread = threading.Thread(target=get_input)
            input_thread.start()
            input_thread.join()
            if data[0] == 'exit':
                client_socket.send(pickle.dumps('{} {} has
left'.format(details[2],details[3])))
                break
            else:
                client_socket.send(pickle.dumps(data[0]))
                data.pop(0)
            print(pickle.loads(client_socket.recv(1024)))




        client_socket.close()


# coach only button
# allows coach to view hourly pay, weekly pay and monthly pay
class timesheet_but(Button):
    def __init__(self, x, y, w, h, text='', active=False, colour=COLOUR_ACTIVE):
        super().__init__(x, y, w, h, text, active, colour)

    def pressed_func(self):
        coach_base_page.active = False
        days = []
        for i in coach_base_page.days:
            for j in i:
                temp = j.split(',')
                days.append(temp)
        just_times = []
        for i in days:
            for j in i:
                if j != '':
                    time = j.split('/')

                    just_times.append(time[1])
        total_hours = 0
        new_times = []
        for i in range(len(just_times)):
            new_times.append(just_times[i].replace(':', '.'))
        for i in new_times:
            a, b = i.split('-')
            c = float(b) - float(a)
            hours = c % 12
            total_hours += hours
        hours_text = font.render('Hours Worked per week:
{:.2f}'.format(total_hours), True, WHITE)
        hours_text = [hours_text, (SCR_W * (2 / 8), (SCR_H * 1 / 6))]
        timesheet_screen.text.append(hours_text)
        con = db.create_connection('file.db')
        data = db.selectvalue(con, 'pay', 'payrate', 'id',
coach_base_page.memberid)
        rate = data[0][0]
        con.close()
        rate_text = font.render('Rate of pay: {:.2f}'.format(rate), True, WHITE)
        rate_text = [rate_text, (SCR_W * (2 / 8), (SCR_H * 2 / 6))]
        timesheet_screen.text.append(rate_text)
        money_week = font.render('Money per week: £{:.2f}'.format(total_hours *
rate), True, WHITE)
        money_week = [money_week, (SCR_W * (2 / 8), (SCR_H * 3 / 6))]
        timesheet_screen.text.append(money_week)
        money_month = font.render('Money per month: £{:.2f}'.format(total_hours *
rate * 4.345), True, WHITE)
        money_month = [money_month, (SCR_W * (2 / 8), (SCR_H * 4 / 6))]
        timesheet_screen.text.append(money_month)
        timesheet_screen.active = True


# Text box class
class InputBox:

    def __init__(self, x, y, w, h, text='', show=False, view_text=True):
        self.rect = pg.Rect(x, y, w, h)
        self.colour = WHITE  # COLOR_INACTIVE
        self.text = text
        self.txt_surface = FONT.render(text, True, self.colour)
        self.active = False
        self.show = show
        self.output = None
        self.entered = False
        self.view_text = view_text
        self.ori_text = text

    def handle_event(self, event):
        if event.type == pg.MOUSEBUTTONDOWN:
            # If the user clicked on the input_box rect.
            if self.rect.collidepoint(event.pos):
                # Toggle the active variable.
                self.active = not self.active
                self.text = ''
            else:
                self.active = False
            # Change the current color of the input box.
            self.colour = COLOUR_ACTIVE if self.active else WHITE
        if event.type == pg.KEYDOWN:
            if self.active:
                if event.key == pg.K_BACKSPACE:
                    self.text = self.text[:-1]
                else:
                    self.text += event.unicode
                    self.entered = False
                # Re-render the text.
                length = len(self.text)
                star = '*' * length
                if self.view_text:
                    self.txt_surface = FONT.render(self.text, True, self.colour)
                else:
                    self.txt_surface = FONT.render(star, True, self.colour)

    def update(self):
        # Resize the box if the text is too long.
        width = max(200, self.txt_surface.get_width() + 10)
        self.rect.w = width

    def draw(self, screen):
        # Blit the text.
        screen.blit(self.txt_surface, (self.rect.x + 5, self.rect.y + 5))
        # Blit the rect.
        pg.draw.rect(screen, self.colour, self.rect, 2)


# main class screen

class screen:

    def __init__(self, active=False):
        self.active = active
        self.previous = None
        self.background_colour = BACKGROUND_COLOUR
        self.buttons = []
        self.input_boxes = []
        self.text = []
    # runs the screen checking if button  or input boxes are clicked on
    def screen_run(self):
        for event in pg.event.get():
            for i in range(len(self.buttons)):
                self.buttons[i].button_handle((event))
            for box in self.input_boxes:
                if box.show == True:
                    box.handle_event(event)

        for box in self.input_boxes:
            box.update()

        SCREEN.blit(b, (0, 0))
        for box in self.input_boxes:
            if box.show == True:
                box.draw(SCREEN)
        for button in self.buttons:
            if button.active == True:
                button.draw()
        for t in self.text:
            SCREEN.blit(t[0], t[1])
        pg.display.update()


## login screen class
class login(screen):

    def __init__(self, active):
        super().__init__(active)
        self.user_text = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 50, 100, 30,
'enter memberID', True)
        self.password_text = InputBox((SCR_W / 2) - 100, (SCR_H / 2), 100, 30,
'enter password', True, False)
        self.input_boxes = [self.user_text, self.password_text]
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.submit_login_button = Submit_Button((SCR_W / 2) - 50, (SCR_H / 2) +
50, 50, 30, 'SUBMIT', True)
        self.new_member_button = New_Member_Button(0, 0, 50, 30, 'New Member?',
True)
        self.buttons = [self.close_button, self.submit_login_button,
self.new_member_button]
        con = db.create_connection('file.db')
        con.close()


## New member screen 1
class new_member_pass(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.memberID = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 25, 100, 30,
'Enter your memberID', True)
        self.submit_button = New_Member_Button_submit((SCR_W / 2) - 50, (SCR_H / 2)
+ 50, 50, 30, 'SUBMIT', True)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.submit_button, self.back_button]
        self.input_boxes = [self.memberID]


## New member screen 2
class new_member_pass2(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.password1 = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 50, 100, 30,
'enter new password', True, False)
        self.password2 = InputBox((SCR_W / 2) - 100, (SCR_H / 2), 100, 30, 're-
enter password  ', True, False)
        self.submit_button = New_Member_Button_submit2((SCR_W / 2) - 50, (SCR_H /
2) + 50, 50, 30, 'SUBMIT', True)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.submit_button, self.back_button]
        self.input_boxes = [self.password1, self.password2]
        self.font = pg.font.Font('freesansbold.ttf', 16)
        self.pass_text_help = self.font.render(
            'Your password must be at least 8 characters long, have at least 1
capital letter, have at least 1 special character and have at least 1 digit ',
            True, WHITE)
        self.pass_text_help_to_screen = [self.pass_text_help,
                                         ((SCR_W * (1 / 2)) -
(self.pass_text_help.get_width() / 2), SCR_H * (1 / 8))]
        self.text = [self.pass_text_help_to_screen]


## Class for admin page
class admin_base(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.create_member = add_member(SCR_W * 1 / 5 - 50, SCR_H * 1 / 3 - 50,
100, 100, ' create member ', True)
        self.create_class = add_class_but(SCR_W * 2 / 5 - 50, SCR_H * 1 / 3 - 50,
100, 100, '  Create class  ', True)
        self.add_to_class = add_to_class_but(SCR_W * 3 / 5 - 50, SCR_H * 1 / 3 -
50, 100, 100, '  Add to class  ', True)
        self.badges = badges_but(SCR_W * 4 / 5 - 50, SCR_H * 1 / 3 - 50, 100, 100,
' Add badge ', True)
        self.classes = classes_but(SCR_W * 1 / 5 - 50, SCR_H * 2 / 3 - 50, 100,
100, ' Take Register ', True)
        self.pay = pay_but(SCR_W * 2 / 5 - 50, SCR_H * 2 / 3 - 50, 100, 100, 'Set
employee pay', True)
        self.edit = edit_but(SCR_W * 3 / 5 - 50, SCR_H * 2 / 3 - 50, 100, 100, '
Edit      ', True)
        self.view_reg = view_register(SCR_W * 4 / 5 - 50, SCR_H * 2 / 3 - 50, 100,
100, ' View Register ', True)

        self.buttons = [self.close_button, self.back_button, self.create_member,
self.create_class, self.add_to_class,
                        self.classes, self.pay, self.edit, self.view_reg,
self.badges]
        self.text = []


class create_member(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit = add_member_submit((SCR_W / 2) - 50, (SCR_H * (11 / 12)), 50,
30, 'Submit', False)
        self.next = add_member_next((SCR_W / 2) - 50, (SCR_H * (11 / 12)), 50, 30,
'Next', True)
        self.buttons = [self.close_button, self.back_button, self.submit,
self.next]
        self.firstname = InputBox((SCR_W / 2) - 100, (SCR_H * (1 / 12)), 100, 30,
'enter firstname', True)
        self.lastname = InputBox((SCR_W / 2) - 100, (SCR_H * (3 / 12)), 100, 30,
'enter lastname', True)
        self.type = InputBox((SCR_W / 2) - 100, (SCR_H * (5 / 12)), 100, 30, 'enter
member type', True)
        self.DOB = InputBox((SCR_W / 2) - 100, (SCR_H * (7 / 12)), 100, 30, 'enter
DOB', True)
        self.date_joined = InputBox((SCR_W / 2) - 100, (SCR_H * (9 / 12)), 100, 30,
'enter date_joined', True)
        self.mobile_number = InputBox((SCR_W / 2) - 100, (SCR_H * (1 / 12)), 100,
30, 'enter mobile number', False)
        self.medical = InputBox((SCR_W / 2) - 100, (SCR_H * (3 / 12)), 100, 30,
'enter medical information', False)
        self.postcode = InputBox((SCR_W / 2) - 100, (SCR_H * (5 / 12)), 100, 30,
'enter postcode', False)
        self.gender = InputBox((SCR_W / 2) - 100, (SCR_H * (7 / 12)), 100, 30,
'enter gender', False)
        self.input_boxes = [self.firstname, self.lastname, self.type, self.DOB,
self.date_joined, self.mobile_number,
                            self.medical, self.postcode, self.gender]


class create_class(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit = add_class_submit((SCR_W / 2) - 50, (SCR_H * (11 / 12)), 50,
30, 'Submit', False)
        self.next = add_class_next((SCR_W / 2) - 50, (SCR_H * (11 / 12)), 50, 30,
'Next', True)
        self.buttons = [self.close_button, self.back_button, self.submit,
self.next]
        self.name = InputBox((SCR_W / 2) - 100, (SCR_H * (1 / 12)), 100, 30, 'enter
name of class', True)
        self.no_days = InputBox((SCR_W / 2) - 100, (SCR_H * (5 / 12)), 100, 30,
                                'enter number of days of class per week', True)
        self.kidsinClass = InputBox((SCR_W / 2) - 100, (SCR_H * (9 / 12)), 100, 30,
'enter max kids in class', True)
        self.input_boxes = [self.name, self.no_days, self.kidsinClass]


class add_to_class(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.next = add_to_class_id_next((SCR_W / 2) - 50, (SCR_H / 2) + 50, 50,
30, 'NEXT', True)
        self.buttons = [self.close_button, self.back_button, self.next]
        self.id_search = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 25, 100, 30,
'Enter memberID to add', True)
        self.input_boxes = [self.id_search]
        self.class_id = []
        self.classes = []

# set pay screen
class pay(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit = pay_submit((SCR_W / 2) - 50, (SCR_H / 2) + 75, 50, 30,
'NEXT', True)
        self.buttons = [self.close_button, self.back_button, self.submit]
        self.id = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 25, 100, 30, 'Enter
memberID to set pay for', True)
        self.pay_input = InputBox((SCR_W / 2) - 100, (SCR_H / 2) + 25, 100, 30,
'Enter rate of pay for employee', True)
        self.input_boxes = [self.id, self.pay_input]

# add badges screen
class badges(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit = badges_submit((SCR_W / 2) - 50, (SCR_H / 2) + 75, 50, 30,
'SUBMIT', True)
        self.buttons = [self.close_button, self.back_button, self.submit]
        self.id = InputBox((SCR_W / 2) - 100, (SCR_H / 2) - 25, 100, 30, 'Enter
memberID to add badge to', True)
        self.badges_input = InputBox((SCR_W / 2) - 100, (SCR_H / 2) + 25, 100, 30,
'Enter name of badge', True)
        self.input_boxes = [self.id, self.badges_input]

# screen to show all classes
class classes(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.input_boxes = []
        self.classes = []
```

```python
        self.next_screen = []


class Register_main(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.input_boxes = []


class Regster_class(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit_button = register_submit((SCR_W * 6 / 8), SCR_H - 100, 50, 30,
'SUBMIT', True)
        self.buttons = [self.close_button, self.back_button, self.submit_button]
        self.input_boxes = []
        self.classid = ''
        self.text = []
        self.links = []
        self.day = []
        self.class_name = ''


class reg_view_1(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.input_boxes = []
        self.classid = ''
        self.classes = []


class reg_view_2(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.input_boxes = []
        self.day = ''
        self.classid = ''
        self.classes = []
        self.text = []


class edit_first(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.classes = []


class edit_second(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.button1 = class_dets_but(SCR_W * 1 / 3, SCR_H * 1 / 2, 50, 30, 'Edit
class details', True)
```

```
        self.button2 = edit_class_but(SCR_W * 2 / 3, SCR_H * 1 / 2, 50, 30, 'Edit
class', True)
        self.buttons = [self.close_button, self.back_button, self.button1,
self.button2]
        self.name = ''
        self.classid = ''


class edit_class_dets(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.next = edit_class_dets_next((SCR_W / 2) - 50, (SCR_H * (11 / 12)), 50,
30, 'next', True)
        self.submit = ed_class_dets_submit((SCR_W / 2) - 50, (SCR_H * (11 / 12)),
50, 30, 'Submit', False)
        self.buttons = [self.close_button, self.back_button, self.submit,
self.next]
        self.name = InputBox((SCR_W / 2) - 100, (SCR_H * (1 / 12)), 100, 30, 'enter
name of class', True)
        self.no_days = InputBox((SCR_W / 2) - 100, (SCR_H * (5 / 12)), 100, 30,
                                'enter number of days of class per week', True)
        self.kidsinClass = InputBox((SCR_W / 2) - 100, (SCR_H * (9 / 12)), 100, 30,
'enter max kids in class', True)
        self.input_boxes = [self.name, self.no_days, self.kidsinClass]


class edit_class(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.submit = remove_submit((SCR_W * 6 / 8), SCR_H - 100, 50, 30, 'SUBMIT',
True)
        self.buttons = [self.close_button, self.back_button, self.submit]
        self.text = []
        self.links = []


class chat(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        text = font.render('Chatroom in console', True, WHITE)
        text = [text, (SCR_W / 2, SCR_H / 2)]
        self.text = [text]


class gymnast_base(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.details = member_details_but(SCR_W * 1 / 3, SCR_H * 1 / 4, 100, 100,
'member details', True)
        self.general = gymnast_general_but(SCR_W * 2 / 3, SCR_H * 1 / 4, 100, 100,
' General ', True)
        self.game_but = game_but(SCR_W * 1 / 2, SCR_H * 3 / 4, 100, 100, '    Game
', True)
        self.buttons = [self.close_button, self.back_button, self.details,
self.general, self.game_but]
        self.input_boxes = []
        self.memberid = ''
```

```
class member_details(screen):

    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.input_boxes = []
        self.text = []


class gymnast_general(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.console_chat = console_chat_but(SCR_W * 4 / 5, SCR_H * 7 / 8, 100,
100, 'Console chat', True)
        self.buttons = [self.close_button, self.back_button, self.console_chat]
        self.text = []
        self.days = []


class coaches_base(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.register = classes_but(SCR_W * 2 / 5 - 50, SCR_H * 7 / 8 - 50, 100,
100, 'Take Register', True)
        self.timesheet = timesheet_but(SCR_W * 1 / 5 - 50, SCR_H * 7 / 8 - 50, 100,
100, 'Timesheet', True)
        self.console = console_chat_but(SCR_W * 3 / 5 - 50, SCR_H * 7 / 8 - 50,
100, 100, 'Console chat', True)
        self.view_reg = view_register(SCR_W * 4 / 5 - 50, SCR_H * 7 / 8 - 50, 100,
100, 'View Register', True)
        self.buttons = [self.close_button, self.back_button, self.register,
self.timesheet, self.console, self.view_reg]
        self.text = []
        self.memberid = ''
        self.days = []
        calender_title = 'Class times:'
        calender_title = font.render(calender_title, True, WHITE)
        calender_title = [calender_title, (SCR_W * (1 / 8), (SCR_H * 1 / 8))]
        self.text.append(calender_title)


class timesheet(screen):
    def __init__(self, active):
        super().__init__(active)
        self.close_button = Close_Button(SCR_W - 65, 0, 50, 30, 'QUIT', True, RED)
        self.back_button = Back_Button(0, 0, 50, 30, 'Back', True)
        self.buttons = [self.close_button, self.back_button]
        self.text = []

# class to work out timetable given an id number
def calender(id):
    con = db.create_connection('file.db')
    data = db.selectvalue(con, 'classid', 'class', 'memberid', id)
    times = []
    try:
        for i in data:
            new_data = i[0]
            times_add = db.selectvalue(con, 'days_times', 'class_details',
'classid', new_data)
            for i in times_add:
                times.append(i)
    except:
        pass
    con.close()
    return times




# set up all screen

login_page = login(True)
admin_base_page = admin_base(False)
gymnast = gymnast_base(False)
new_password = new_member_pass(False)
new_password1 = new_member_pass2(False)
create_member_screen = create_member(False)
add_to_class_screen = add_to_class(False)
create_class_screen = create_class(False)
classes_screen = classes(False)
register_screen = Register_main(False)
register_show_screen = Regster_class(False)
member_details_screen = member_details(False)
gymnast_general_screen = gymnast_general(False)
coach_base_page = coaches_base(False)
timesheet_screen = timesheet(False)
pay_screen = pay(False)
edit_first_screen = edit_first(False)
edit_second_screen = edit_second(False)
edit_class_dets_screen = edit_class_dets(False)
edit_class_screen = edit_class(False)
chat_screen = chat(False)
reg_view_1_screen = reg_view_1(False)
reg_view_2_screen = reg_view_2(False)
badge_screen = badges(False)

#set up back button relationships

back_relationships = {gymnast: login_page,
                      member_details_screen: gymnast,
                      new_password: login_page,
                      new_password1: new_password,
                      coach_base_page: login_page,
                      admin_base_page: login_page,
                      create_member_screen: admin_base_page,
                      add_to_class_screen: admin_base_page,
                      pay_screen: admin_base_page,
                      badge_screen: admin_base_page,
                      create_class_screen: admin_base_page,
                      classes_screen: admin_base_page,
                      register_screen: classes_screen,
                      register_show_screen: register_screen,
                      gymnast_general_screen: gymnast,
                      timesheet_screen: coach_base_page,
                      edit_first_screen: admin_base_page,
                      edit_second_screen: edit_first_screen,
                      edit_class_dets_screen: edit_second_screen,
                      edit_class_screen: edit_second_screen,
                      chat_screen: gymnast_general_screen,
                      reg_view_1_screen: admin_base_page,
                      reg_view_2_screen: reg_view_1_screen}

# main loop


while True:
    while login_page.active:
        login_page.screen_run()
    while new_password.active:
        new_password.screen_run()
    while new_password1.active:
        new_password1.screen_run()
    while gymnast.active:
        gymnast.screen_run()
    while member_details_screen.active:
        member_details_screen.screen_run()
    while gymnast_general_screen.active:
        gymnast_general_screen.screen_run()
    while admin_base_page.active:
        admin_base_page.screen_run()
    while create_member_screen.active:
        create_member_screen.screen_run()
    while create_class_screen.active:
        create_class_screen.screen_run()
    while add_to_class_screen.active:
        add_to_class_screen.screen_run()
    while classes_screen.active:
        classes_screen.screen_run()
    while register_screen.active:
        register_screen.screen_run()
    while register_show_screen.active:
        register_show_screen.screen_run()
    while coach_base_page.active:
        coach_base_page.screen_run()
    while timesheet_screen.active:
        timesheet_screen.screen_run()
    while pay_screen.active:
        pay_screen.screen_run()
    while edit_first_screen.active:
        edit_first_screen.screen_run()
    while edit_second_screen.active:
        edit_second_screen.screen_run()
    while edit_class_dets_screen.active:
        edit_class_dets_screen.screen_run()
    while edit_class_screen.active:
        edit_class_screen.screen_run()
    while chat_screen.active:
        chat_screen.screen_run()
    while reg_view_1_screen.active:
        reg_view_1_screen.screen_run()
    while reg_view_2_screen.active:
        reg_view_2_screen.screen_run()
    while badge_screen.active:
        badge_screen.screen_run()

    pg.display.update()
```

game.py

```
# import necessary libraries
import pygame as pg
import sys
import random
from pygame.locals import *
import time

# initialise pygame
pg.init()
# set up variables
scr_w, scr_h = pg.display.Info().current_w,
pg.display.Info().current_h

screen = pg.display.set_mode((scr_w, scr_h))

b = pg.image.load('test_back.png')
pic = pg.transform.scale(b, (scr_w, scr_h))
BASICFONT = pg.font.Font('freesansbold.ttf', 30)
SECONDFONT = pg.font.Font('freesansbold.ttf', 24)
clock = pg.time.Clock()
WHITE = (255, 255, 255)
spike_size = 50
player_size = 75
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLACK = (0, 0, 0)
font = pg.font.Font('freesansbold.ttf', 32)
spike = pg.image.load('spike.png')
spike = pg.transform.scale(spike, (spike_size, spike_size))
spike_double = pg.image.load('spike_double.png')
spike_double = pg.transform.scale(spike_double, (spike_size * 2,
spike_size))
char_run_0 = pg.image.load('char_run_0_2.png')
player_img_run = pg.transform.scale(char_run_0, (player_size,
player_size))
char_run_1 = pg.image.load('char_run_1.png')
player_img_run_1 = pg.transform.scale(char_run_1, (player_size,
player_size))
char_jump_1 = pg.image.load('adventurer-jump-00.png')
char_jump_1 = pg.transform.scale(char_jump_1, (player_size,
player_size))
char_jump_2 = pg.image.load('adventurer-jump-01.png')
char_jump_2 = pg.transform.scale(char_jump_2, (player_size,
player_size))
char_jump_3 = pg.image.load('adventurer-jump-02.png')
char_jump_3 = pg.transform.scale(char_jump_3, (player_size,
player_size))
char_jump_4 = pg.image.load('adventurer-jump-03.png')
char_jump_4 = pg.transform.scale(char_jump_4, (player_size,
player_size))
char_jump_5 = pg.image.load('adventurer-smrslt-00.png')
char_jump_5 = pg.transform.scale(char_jump_5, (player_size,
player_size))
char_jump_6 = pg.image.load('adventurer-smrslt-01.png')
char_jump_6 = pg.transform.scale(char_jump_6, (player_size,
player_size))
char_jump_7 = pg.image.load('adventurer-smrslt-02.png')
char_jump_7 = pg.transform.scale(char_jump_7, (player_size,
player_size))
char_jump_8 = pg.image.load('adventurer-smrslt-03.png')
char_jump_8 = pg.transform.scale(char_jump_8, (player_size,
player_size))
jump_char = [char_jump_1, char_jump_2, char_jump_3, char_jump_4,
char_jump_5, char_jump_6, char_jump_7, char_jump_8]
play = False
menu = True
highscore = False

bottom = (3 * (scr_h / 4))
speed = [0, 0]
gravity = 0.2


# main block class
class block:

    def __init__(self, xpos, ypos, size, visibility, image):
        self.xpos = xpos
        self.ypos = ypos
        self.size = size
        self.visibility = visibility
        self.image = image
        self.rect = image.get_rect()
        self.rect.x = xpos
        self.rect.y = ypos

    # draws block to screen
    def draw(self, screen):
        if self.visibility == True:
            screen.blit(self.image, (self.rect.x, self.rect.y - 50))
        else:
            pass


# player class which is a subclass of block
class player(block):
    def __init__(self, xpos, ypos, size, visibility, image, dead,
jump):
        super().__init__(xpos, ypos, size, visibility, image)
        self.dead = dead
        self.jump = jump

    def draw(self, screen):
        if self.visibility == True:
            screen.blit(self.image, (self.rect.x, self.rect.y))
        else:
            pass


# Platform class
class platform():
    def __init__(self, x, y, w, h):
        self.rect = pg.Rect(x, y, w, h)

    def draw(self, screen):
        pg.draw.rect(screen, GREEN, self.rect)


# this function is where the main section of the game runs
def main():
    game = True
    speed = [0, 0]
    block_speed = 10
    gravity = 0.5
    jump_index = 0
    run = 0
    count = 0
    score = 0

    plats = []
    bad_blocks = []
    s = block(scr_w, bottom, 25, True, spike)
    # instantiate player
    p = player(200, bottom - player_size, player_size, True,
player_img_run, False, False)
    spawn_pos = scr_w + random.randint(0, scr_w)

    base = platform(0, bottom, scr_w, (scr_h - bottom))
    plats.append(base.rect)
    # set up obstacles
    for i in range(4):
        bad_blocks.append(block(spike_size * -1, bottom, spike_size,
True, spike))
    distance = int(scr_w / 4)
    double = block(spike_size * -1, bottom, spike_size * 2, True,
spike_double)
    bad_blocks.append(double)
    # main while loop
    while game:
        if count >= 100:
            score_add = block_speed * 5
            score += score_add
            score = int(score)
            block_speed *= 1.05

            count = 0
        for event in pg.event.get():
            if event.type == QUIT:
                pg.quit()
                sys.exit()
        # checks if player is dead
        if p.dead == True:
            phrase = 'You scored ' + str(score)
            text = BASICFONT.render(phrase, 1, BLACK)
            screen.blit(text, (scr_w / 5, scr_h / 2))
            pg.display.update()
            time.sleep(2)
            file = open('scores.txt', 'r')
            for i in file.readlines():
                if int(i) < score:
                    file.close()
                    file = open('scores.txt', 'w')
                    file.write(str(score))
            file.close()
            game = False
        # checks if player on platform
        if p.rect.collidelist(plats) != -1:
            speed = [0, 0]
            gravity = 0
            p.jump = False
        else:
            gravity = 0.5
            # checks if player has hit an obstacle
        if p.rect.collidelist(bad_blocks) != -1:
            p.dead = True

        screen.fill(WHITE)
        s.draw(screen)
        for i in bad_blocks:
            i.draw(screen)
        p.draw(screen)
        base.draw(screen)
        keys = pg.key.get_pressed()
        for key in keys:
            if keys[pg.K_SPACE] and p.jump == False:
                speed[1] -= 6
                p.jump = True
            if keys[pg.K_RSHIFT] and p.jump == False:
                p.rect = p.rect.move([0, 1])

        p.rect = p.rect.move(speed)
        speed[1] += gravity
        for i in bad_blocks:
            if i.rect.x <= 0:
                i.rect.x = spawn_pos + random.randint(0, scr_w / 2)
                num = 0
                for j in range(len(bad_blocks)):
                    if i.rect.x in range(bad_blocks[j].rect.x - 
distance, bad_blocks[j].rect.x + distance):
                        num += 1
                if num >= 2:
                    i.rect.x = spike_size * -1

            i.rect.x -= block_speed

        text = font.render(str(score), True, RED)
        screen.blit(text, (100, 100))
        clock.tick(60)
        pg.display.update()
        count += 1
        if run < 10 and not p.jump:
            p.image = player_img_run_1
            run += 1
        elif run >= 10 and run < 20 and not p.jump:
            p.image = player_img_run
            run += 1
        # uses different images to animate character
        elif p.jump:

            if run < 5:
                jump_index = 0
            elif run <= 5 and run < 10:
                jump_index = 1
            elif run <= 10 and run < 15:
                jump_index = 2
            elif run <= 15 and run < 20:
                jump_index = 3
            elif run <= 20 and run < 25:
                jump_index = 4
            elif run <= 25 and run < 30:
                jump_index = 5
            elif run <= 30 and run < 35:
                jump_index = 6
            elif run <= 35 and run < 40:
                jump_index = 7
            else:
                run = 0
            p.image = jump_char[jump_index]
            run += 1

        else:

            run = 0
            jump_index = 0


# menu loop
while True:
    while menu:
        for event in pg.event.get():
            if event.type == QUIT:
                pg.quit()
                sys.exit()
            if event.type == KEYDOWN:
                if event.key == K_g:
                    play = True
                    menu = False
                if event.key == K_h:
                    highscore = True
                    menu = False
                if event.key == K_q:
                    pg.quit()
                    sys.exit()
        screen.blit(pic, (0, 0))
        phrase = '''Welcome:'''
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (scr_w * 1 / 2 - 75, scr_h * (1 / 8)))
        phrase = '''Press G to start'''
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (scr_w * 1 / 2 - 100, scr_h * (3 / 8)))
        phrase = '''Press H to see highscores'''
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (scr_w * 1 / 2 - 150, scr_h * (4 / 8)))
        phrase = '''Press Q to quit'''
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (scr_w * 1 / 2 - 100, scr_h * (5 / 8)))
        pg.display.update()

    while play:
        for event in pg.event.get():
            if event.type == QUIT:
                pg.quit()
                sys.exit()
        main()
        menu = True
        play = False

    while highscore:
        for event in pg.event.get():
            if event.type == QUIT:
                pg.quit()
                sys.exit()
            if event.type == KEYDOWN:
                if event.key == K_b:
                    menu = True
                    highscore = False

        screen.blit(pic, (0, 0))
        file = open('scores.txt', 'r')
        for i in file.readlines():
            phrase = '''The highscore is ''' + i.strip('\n')

        file.close()
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (scr_w * 1 / 2 - 150, scr_h * (4 / 8)))
        phrase = 'Press B to go back'
        text = SECONDFONT.render(phrase, 1, BLACK)
        screen.blit(text, (0, scr_h * (19 / 20)))

        pg.display.update()
```

server.py

```
import socket

from _thread import start_new_thread

import pickle


server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)


server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

server.bind(('', 12345))
#binds the server to an entered IP address and at the specified port
number. The client must be aware of these parameters
server.listen(100)
#listens for 100 active connections. This number can be increased as
per convenience
list_of_clients=[]

def clientthread(conn, addr,name):
    conn.send(pickle.dumps("Welcome to this chatroom! Type exit to
leave"))
    #sends a message to the client whose user object is conn
    while True:
            try:
                message = pickle.loads(conn.recv(2048))
                if message:
                    print("<" + name + "> " + message)
                    message_to_send = "<" + name + "> " + message
                    broadcast(message_to_send,conn)
                    #prints the message and address of the user who
just sent the message on the server terminal
                else:
                    remove(conn)
            except:
                continue

def broadcast(message,connection):
    #sends out sent messages to every other connected client
    for clients in list_of_clients:
        if clients != connection:
            try:

                clients.send(pickle.dumps(message))
            except:
                clients.close()
                remove(clients)

def remove(connection):
    if connection in list_of_clients:
        list_of_clients.remove(connection)

while True:
    conn, addr = server.accept()
    list_of_clients.append(conn)
    message = pickle.loads(conn.recv(2048))
    print('{} connected'.format(message))
    print(addr[0] + " connected")
    #maintains a list of clients for ease of broadcasting a message
to all available people in the chatroom
    #Prints the address of the person who just connected
    start_new_thread(clientthread,(conn,addr,message))
    #threading.Thread.(clientthread,(conn,addr))
    #creates and individual thread for every user that connects

conn.close()
server.close()
```

Testing

Test Number	Requirement	Test Description	Test Data	Expected Output	Actual Output	Result
1	L1	Login screen is shown with area to enter id and password	Run main.py	Presented with login screen	Presented with login screen	Pass
2	L2	New member button is visible on start of program	Run main.py	Button is visible	Button is visible	Pass
3	N1	When new member button is pressed it takes you to a page to enter id	Press new member button on login page	Go to page where you can input id	Go to page where you can input id	Pass
4	N2	Incorrect value is entered	Nothing is entered and submit button is pressed	Nothing happens	Nothing happens	Pass
5	N2	Incorrect value is entered	String is entered	Nothing happens	Nothing happens	Pass

6	N2	Correct value is entered	0 is entered	Takes you to the next screen	Takes you to the next screen	Pass
7	N3	Next screen shows two input boxes and a submit button with text showing explaining what a valid password is	Submit button on previous page is pressed with valid data	Shows screen	Shows screen	Pass
8	N3	Invalid password is entered	Orange Orange	Nothing happens	Nothing happens	Pass
9	N3	Passwords do not match	Asdf12@ AsDF32&	Nothing happens	Nothing happens	Pass
10	N3	Valid password entered	Finlay1234@ Finlay1234@	Taken back to login screen	Taken back to login screen	Pass
11	L3	Invalid id and password are entered	Hello 1234sa	Text telling the user they have not entered the details correctly will show	Text telling the user they have not entered the details correctly will show	Pass
12	L3 + A1	Valid id and password are entered for admin account	0 Finlay1234@ (admin account example)	Take the user to the admin base page screen	Take the user to the admin base page screen	Pass
13	L3 + C1	Valid id and password are entered for coach account	1 Finlay1234@ (coach account example)	Take the user to the coach base page	Take the user to the coach base page	Pass
14	L3 + G1	Valid id and password are entered for gymnast account	2 Finlay1234@ (gymnast account example)	Take the user to the gymnast base page	Take the to the gymnast base page	Pass
15	L4	Quit button appears on the login page	Run main.py	Quit button visible	Quit button visible	Pass

16	L4	Quit button quits the program	Press the quit button	Program quits	Program quits	Pass
17	G2	Three buttons should show on the screen	Login as a gymnast	All buttons show	All buttons show	Pass
18	G3	Member details button pressed takes you to a page in which shows member details	Press member details button	Takes the user to the member details screen with the details on the screen	Takes the user to the member details screen with the details on the screen	Pass
19	G4	Taken to a page where they can view their class name, class time, Badges, coaches and the console chat button is visible	Press General button	Taken to page with all details and button showing	Taken to page with all details and button showing	Pass
20	G5	Console chat function runs in console	Press console chat function	Console chat runs	Console chat runs	Pass
21	G6+ G6.1	Game button Starts game	Press game button	Game starts	Game starts	Pass
22	G6.2	Game menu shows with all its text showing	Press game button	Game menu shows with all text showing	Game menu shows with all text showing	Pass
23	G6.4	User presses H for high score screen	‘H’	High score menu shows which has the current high score on show	High score menu shows which has the current high score on show	Pass
24	G6.5	Back button pressed to	‘B’	Takes user back to	Takes user back to	Pass

		get back to menu		game menu screen	game menu screen	
25	G6.6	When ‘G’ is pressed It takes you to the game screen and starts the game	‘G’	Game screen shows and game starts to run	Game screen shows and game starts to run	Pass
26	G6.7	When space is pressed character jumps	‘Space’ is pressed	Character jumps	Character jumps	Pass
27	G6.8	Score increased as game goes along		Score increases	Scores increases	Pass
28	G6.9	High score updates if score is greater than current high score when player hits bad block	Player hits bad block with a new high score	User taken back to main menu and high score updated	User taken back to main menu and high score updated	Pass
29	G6.9	High score not updated if score is not greater than current high score when player hit bad block	Player hits bad block with a lower score than the high score	User taken back to main menu	User taken back to main menu	Pass
30	G6.10	User taken back to membership system if ‘Q’ is pressed	‘Q’	User taken back to general page	User taken back to general page	Pass
31	C1+C2	After logging on to a coach account user presented with timetable, register button, timesheet	Logged on as a coach	User taken to page which shows their timetable, a register button, a timesheet button and a console	User taken to page which shows their timetable, a register button, a timesheet button and a console	Pass

		button and console chat button		chat button	chat button	
32	C3 + A8	After clicking register button they should be taken to a page of all the classes they coach	Register button pressed	Taken to a page in which they can view all classes they can take a register for as a button	Taken to a page in which they can view all classes they can take a register for as a button	Pass
33	C3.1 + A8.a + A9	After clicking on a class, they will be shown they times that class runs as buttons	One of the classes buttons is pressed	Taken to a screen which shows what days that specific class runs per week as buttons	Taken to a screen which shows what days that specific class runs per week as buttons	Pass
34	C3.2 + A8.b	After clicking on this time, the register screen will show	Class time button pressed	All gymnasts in the class will show with 2 buttons next to them, ‘present’ and ‘absent’, as well as a submit button at the bottom	All gymnasts in the class will show with 2 buttons next to them, ‘present’ and ‘absent’, as well as a submit button at the bottom	Pass
35	C3.2 + A8.b	When one of the present or absent button is pressed it turns green	Press an absent button	Absent button turns green	Absent button turns green	Pass
36	C3.2 + A8.b	When one of the present or absent buttons is green the other	Press the Present button after pressing the absent button	Present button turns green and absent button turns back to normal	Present button turns green and absent button	Pass

		button on the same line cannot be green			turns back normal	
37	C3.2 + A8.b	Submit button pressed submit register into database and takes user back to coach base page or admin page	Press the submit button	Register is submitted into database and user is taken back to coach base page or admin page	Register is submitted into database and user is taken back to coach base page or admin page	Pass
38	C4	Taken to timesheet screen which shows the coaches rate of pay, pay per week and pay per month	Press the timesheet button	Taken to screen in which has the coaches rate of pay, pay per week and pay per month	Taken to screen in which has the coaches rate of pay, pay per week and pay per month	Pass
39	C5 + G5	Console chat runs when button pressed	Press the console chat button	Console chat runs in console	Console chat runs in console	Pass
40	C5 + G5	When you type a message into chat it is sent to server	Enter text such as, 'hello'	In server.py hello + name of user shows	In server.py hello + name of user shows	Pass
41	C5 + G5	When another user types a message into their chat it appears on current user's screen	A different user import text such as 'hi there'	'hi there' shows in users console	'hi there' shows in users console	Pass
42	C5 + G5	When user types exit console chat ends	'exit'	Console chat ends and user is taken back	Console chat ends and user is taken back	Pass

				to previous screen	to previous screen	
43	A1	When user logs on as admin his is shown 8 buttons	Logged on as admin	All buttons shown	All buttons shown	Pass
44	A2	When user presses edit button, they are taken to a page that shows all classes	Pressed edit button	User taken to a screen with all classes showing as buttons	User taken to a screen with all classes showing as buttons	Pass
45	A2.a	User is taken to a page in which they can edit either class details or class members	Pressed class button in previous test	User taken to screen with 2 buttons on show. Class details and class	User taken to screen with 2 buttons on show. Class details and class	Pass
46	A2.b	If user presses edit details taken to same page as in create class (A6)	Pressed class_details button	User taken to same screen as in create class function (A6)	User taken to same screen as in create class function (A6)	Pass
47	A2.c	If user presses class button, they will be taken to a page in which they can view and remove any member in class	Pressed class button	User taken to page in which all members in class are visible with a remove button next to each one and a submit button at the bottom	User taken to page in which all members in class are visible with a remove button next to each one and a submit button at the bottom	Pass
48	A2.c	When user presses remove button next to name it turns red	Press remove button	Turns remove button red	Turns remove button red	Pass

49	A2.c	When submit button is pressed any members labelled to be removed are removed and user taken back to admin page	Submit button pressed	Any members that are labelled to be removed are removed and user taken back to admin page	Any members that are labelled to be removed are removed and user taken back to admin page	Pass
50	A3	When user presses add member button taken to a page in which they can enter details	Pressed add member button	Shows screen with 5 input boxes saying allowing you to input data with a next button at the bottom	Shows screen with 5 input boxes saying allowing you to input data with a next button at the bottom	Pass
51	A3	When user presses next button taken to a page where more details can be entered	Press next button	Shows screen with 4 input boxes showing to fill out more details and a submit button at the bottom	Shows screen with 4 input boxes showing to fill out more details and a submit button at the bottom	Pass
52	A3	When user presses submit button, the data is inserted into members table and taken back to admin page	Press submit button	Data inserted into table and screen turns back to admin page	Data inserted into table and screen turns back to admin page	Pass
53	A4	Press badges	Press badge button	User is taken to a	User is taken to a	Pass

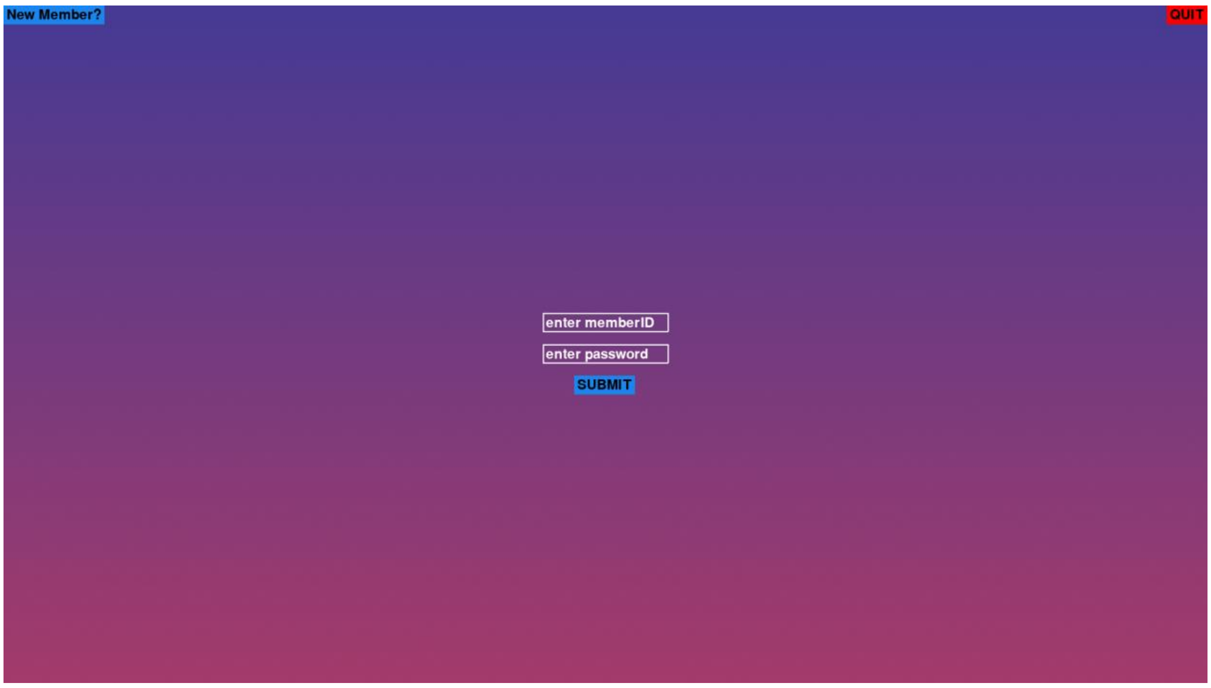
		button then they are taken to a screen in which they can input member id and badge to add		screen in which they can input member id and badge to add as well as a submit button	screen in which they can input member id and badge to add as well as a submit button	
54	A4	When submit button is pressed the badge is added and user taken back to admin page	Press submit button	Badge is added to database and user is taken back to admin page	Badge is added to database and user is taken back to admin page	Pass
55	A5	If user presses pay button, then they are taken to a screen in which they can input member id and pay rate	Press pay button	User is taken to a screen in which they can input member id and pay rate to add as well as a submit button	User is taken to a screen in which they can input member id and pay rate to add as well as a submit button	Pass
56	A5	When submit button is pressed the pay is added into the database and user taken back to admin page	Press submit button	Pay rate is added to database and user is taken back to admin page	Pay rate is added to database and user is taken back to admin page	Pass
57	A6	When create class button is pressed user is taken to screen with 3 input boxes to enter class name, days of class and	Press create class button	User is taken to page with all 3 input boxes showing and a next button	User is taken to page with all 3 input boxes showing and a next button	

		max kids in class				
58	A6.a	Takes the number of days inputted and put the needed number of input boxes on the screen	2 in number of days input box and press next button	4 input boxes shown, 2 day and 2 time	4 input boxes shown, 2 day and 2 time	Pass
58	A6.a	Takes the number of days inputted and put the needed number of input boxes on the screen	4 in number of days input box and press next button	8 input boxes shown, 4 day and 4 time	8 input boxes shown, 4 day and 4 time	Pass
59	A6.b	When submit button is pressed inputs class into database	Submit button pressed	Class inputted into database	Class inputted into database	Pass
60	A7	When add to class button is pressed the user will be prompted to enter a member id number and a next button will show	Add to class button pressed	User prompted to enter a member id number and a next button shown	User prompted to enter a member id number and a next button shown	Pass
61	A7	Classes that member can be added to shown as buttons	Id 1 entered and next button pressed	User shown all classes that member can join as button	User shown all classes that member can join as button	Pass
62	A7.a	When the specific class button	Press a class button.	Member is added to class.	Member is added to class.	Pass

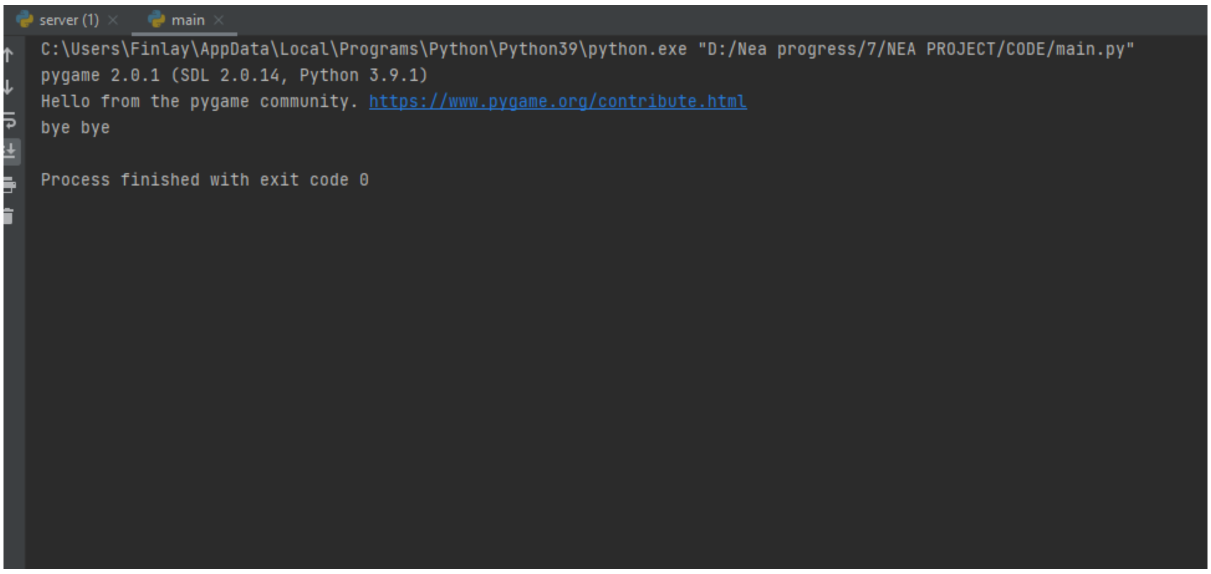
		is pressed it will add that member to the class and user taken back admin page	(Elite example)	(member 1 is added to elite) User taken back to admin page	(member 1 is added to elite) User taken back to admin page	
63	A9 + C6	User taken to screen where names shown with either a present or absent next to it	Press class 'elite' time 'Monday 4-6' after taking register with Finlay Gray present and Alice Alice absent	Screen showing Finlay Gray present and Alice Alice absent	Screen showing Finlay Gray present and Alice Alice absent	Pass

Screenshots for testing

1+2+15

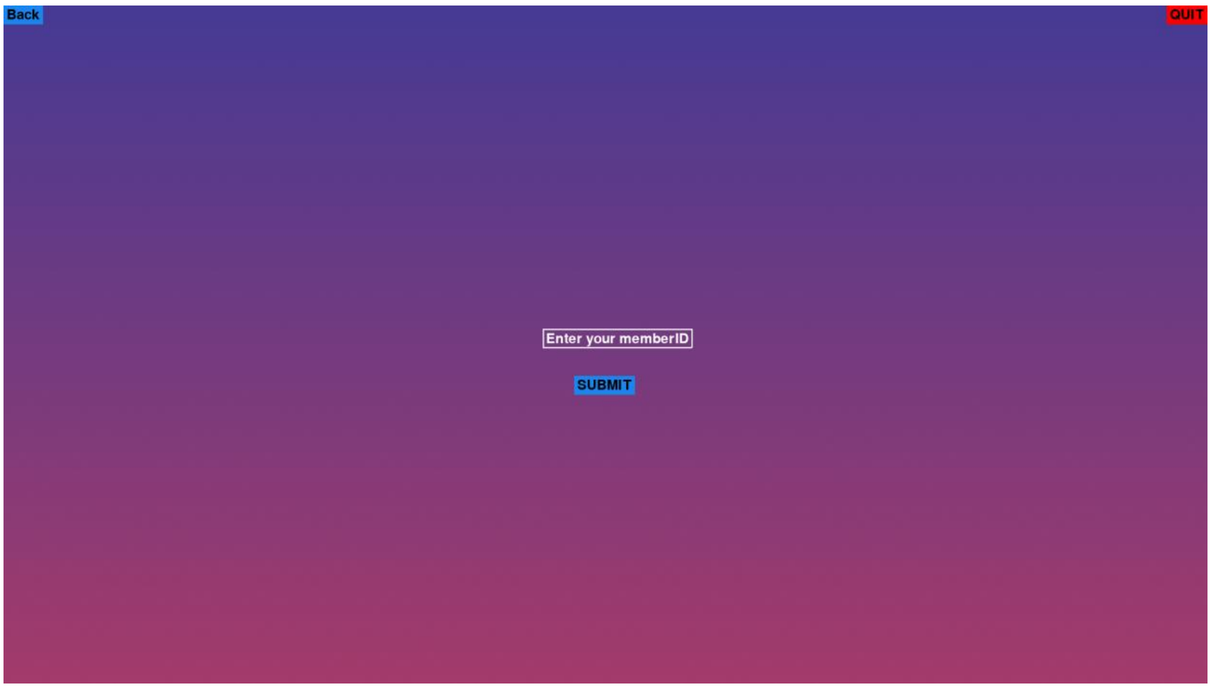


16

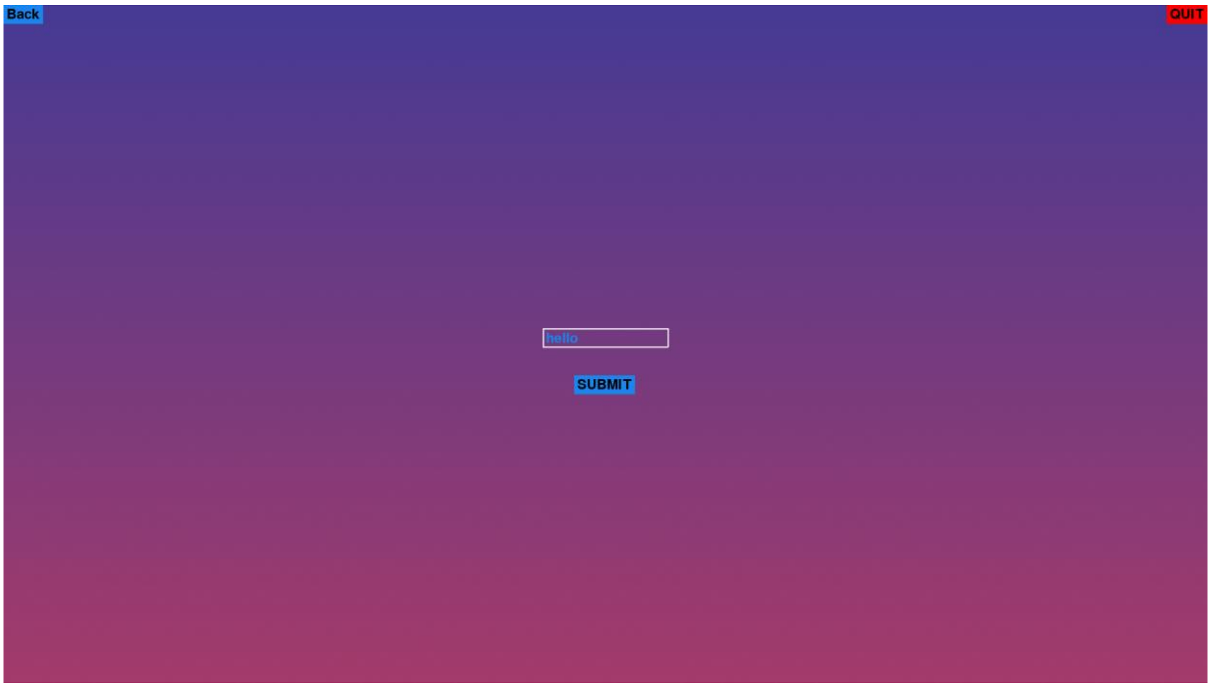


```
C:\Users\Finlay\AppData\Local\Programs\Python\Python39\python.exe "D:/Nea progress/7/NEA PROJECT/CODE/main.py"
pygame 2.0.1 (SDL 2.0.14, Python 3.9.1)
Hello from the pygame community. https://www.pygame.org/contribute.html
bye bye

Process finished with exit code 0
```

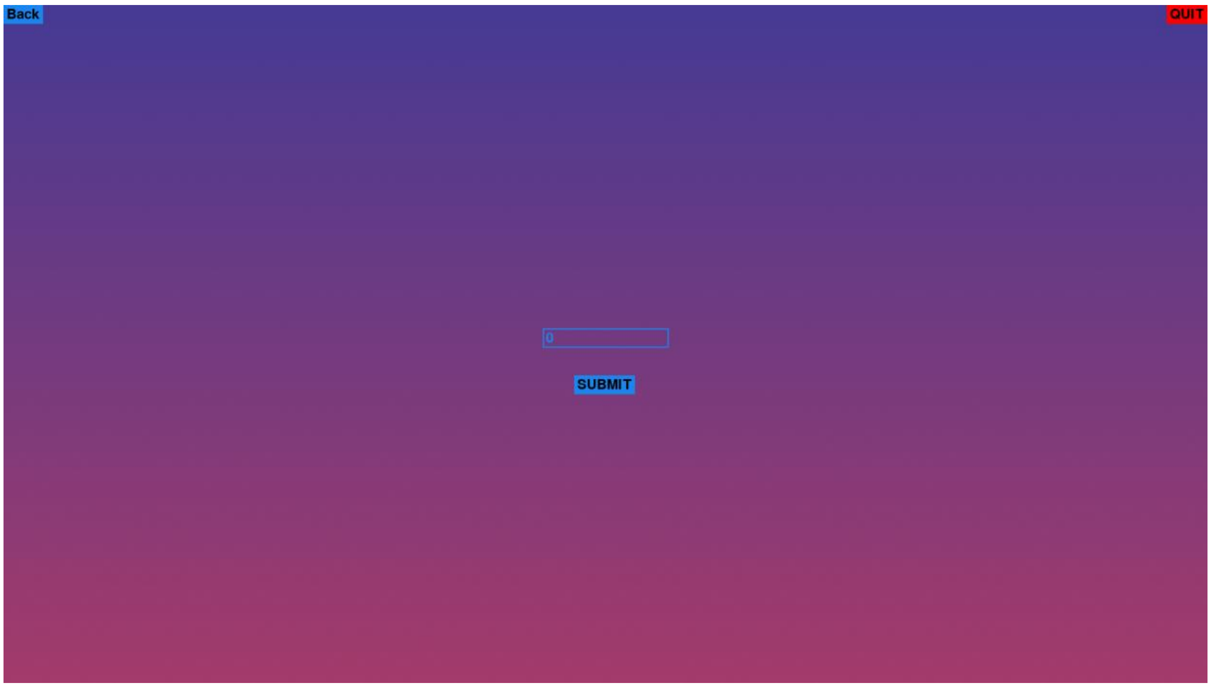
3 + 4



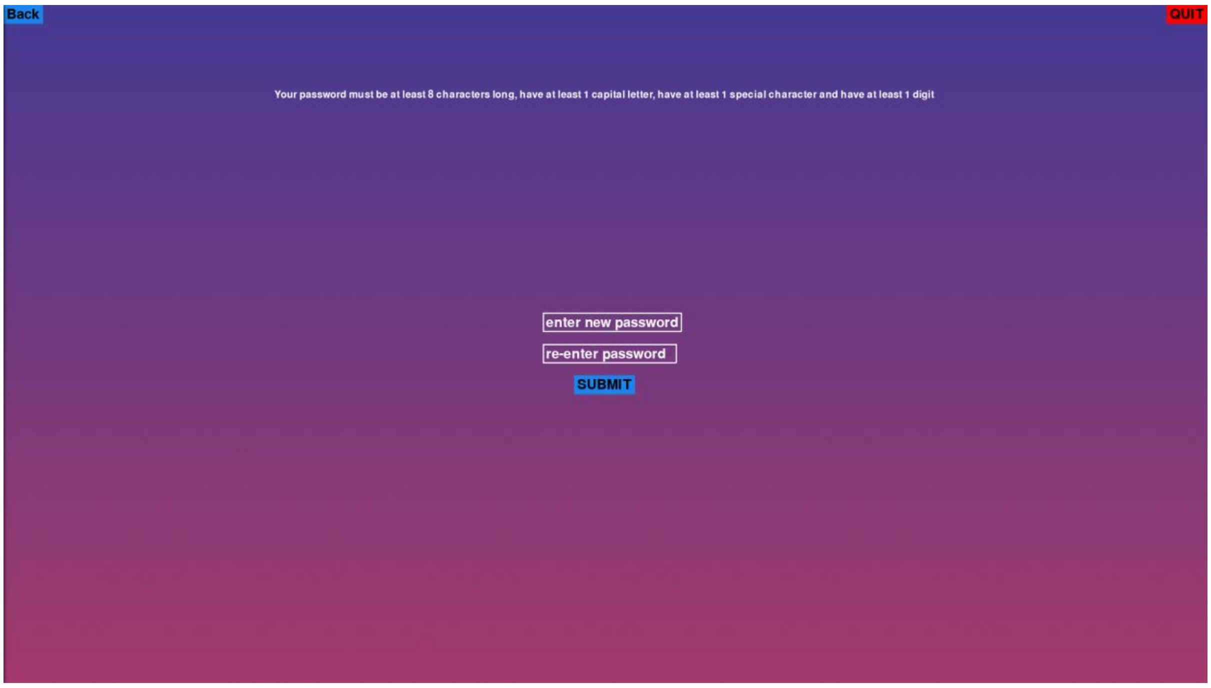
5



6

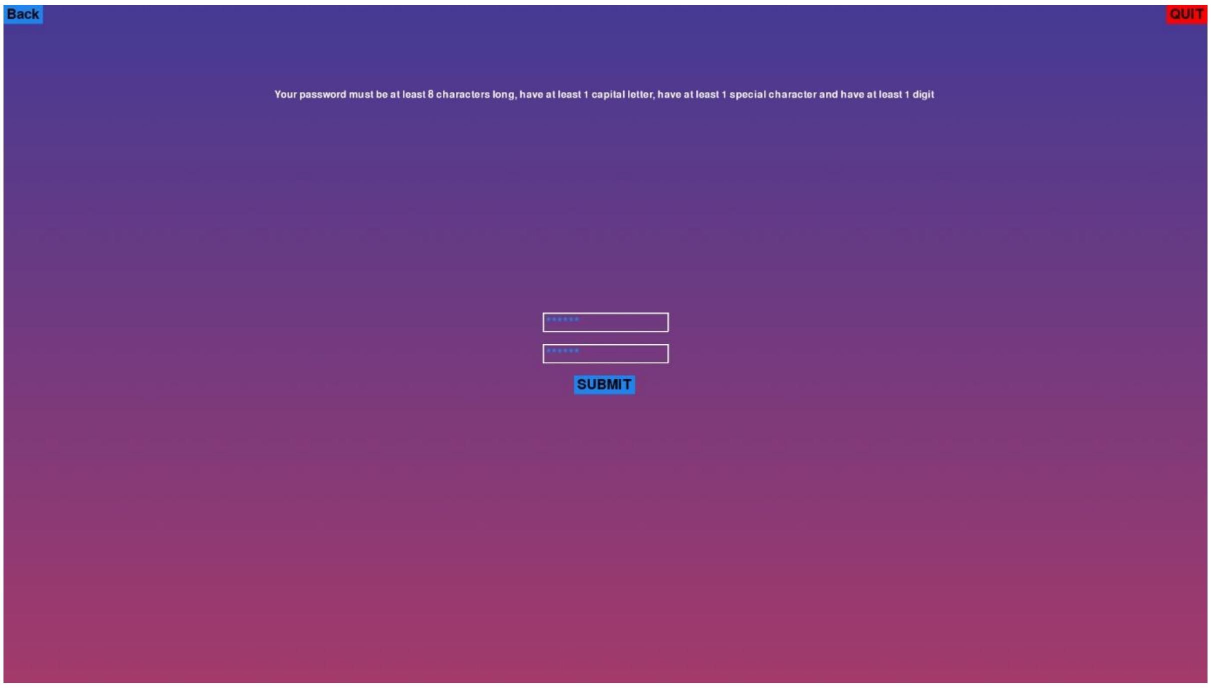


7

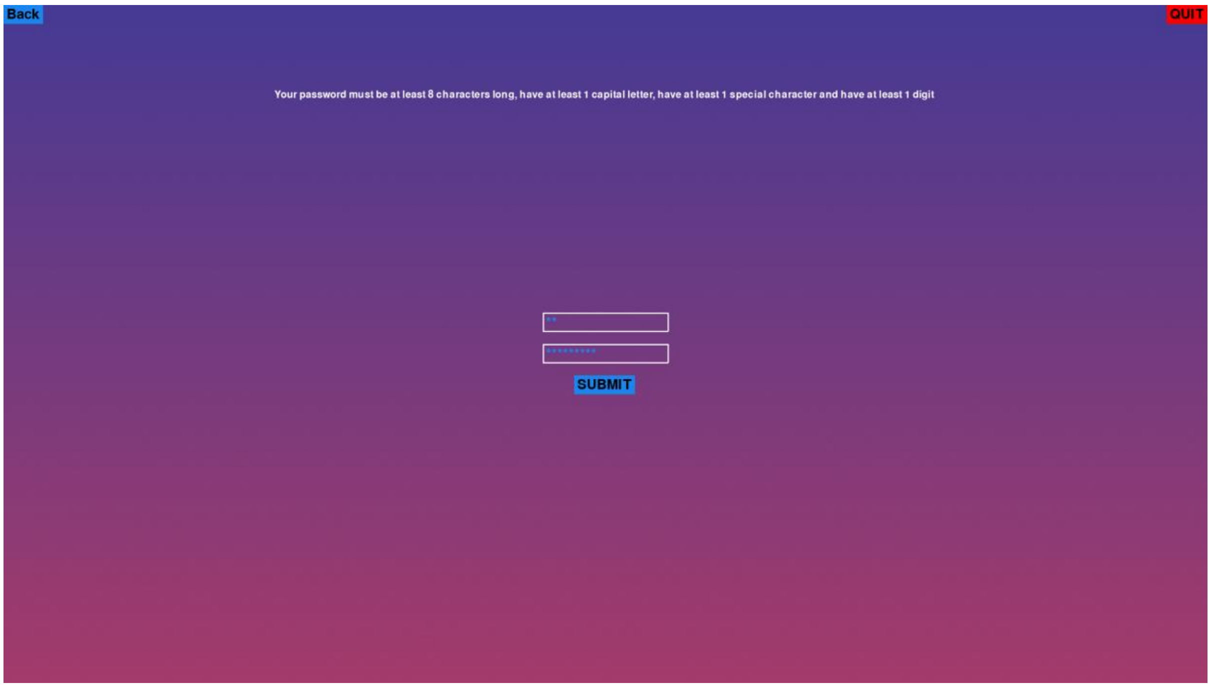


Your password must be at least 8 characters long, have at least 1 capital letter, have at least 1 special character and have at least 1 digit

8

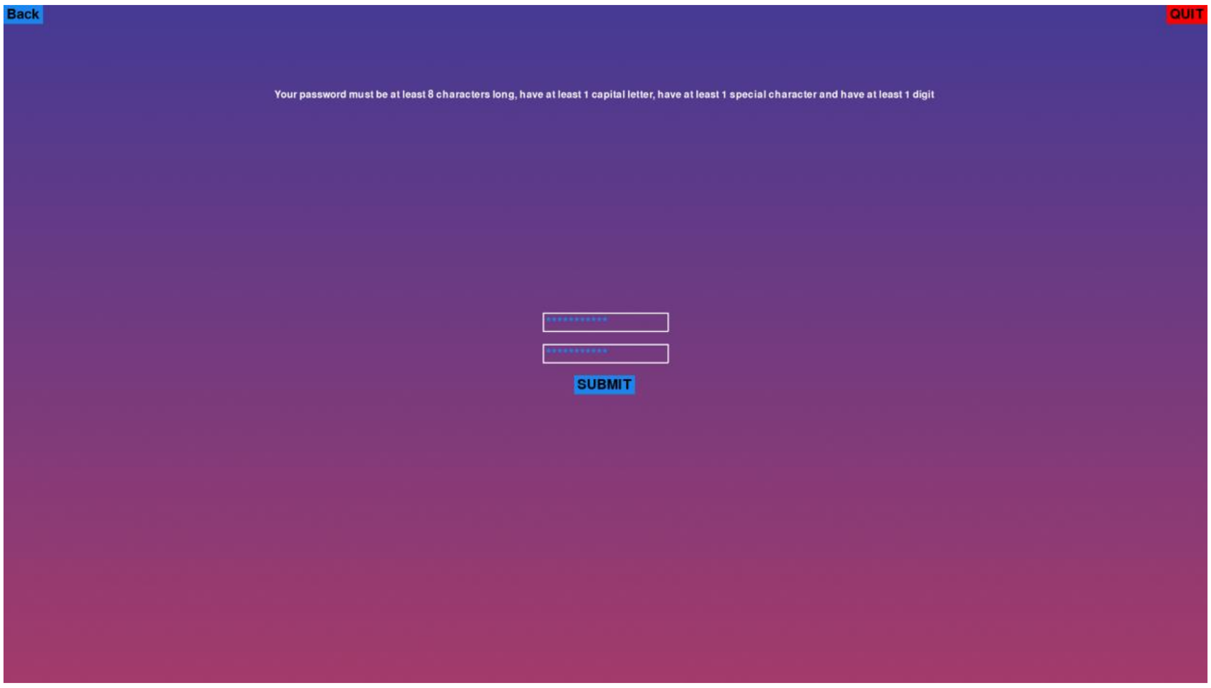


Your password must be at least 8 characters long, have at least 1 capital letter, have at least 1 special character and have at least 1 digit

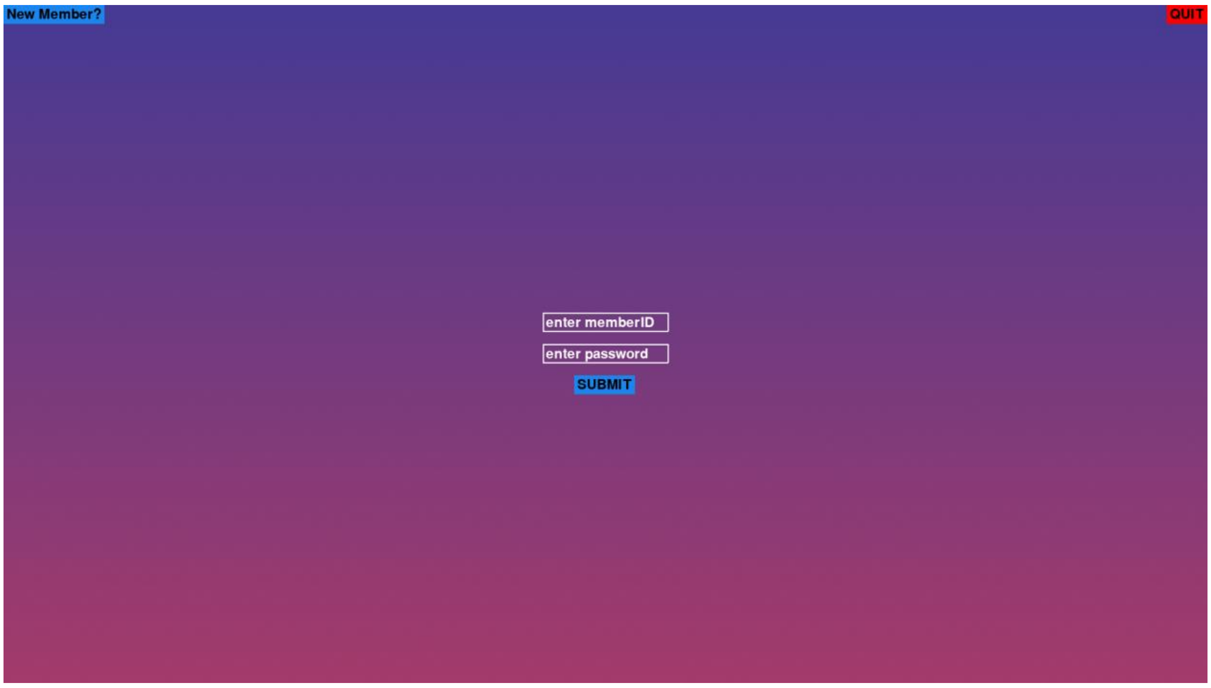
9


Your password must be at least 8 characters long, have at least 1 capital letter, have at least 1 special character and have at least 1 digit

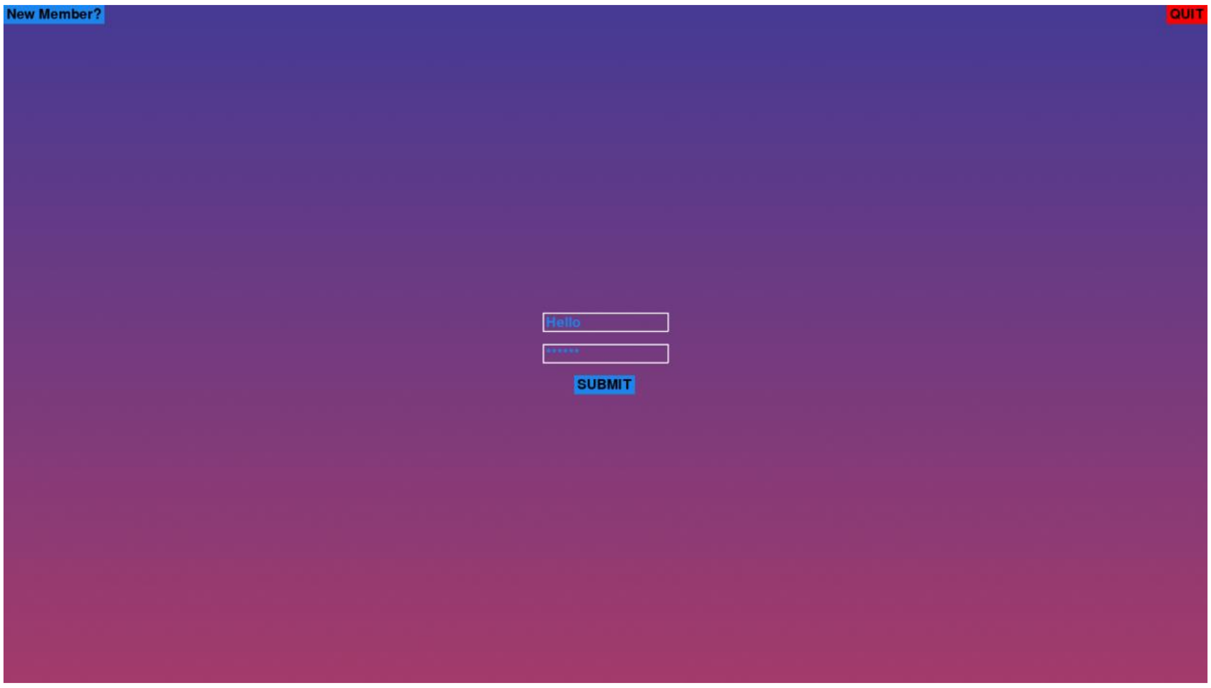
10

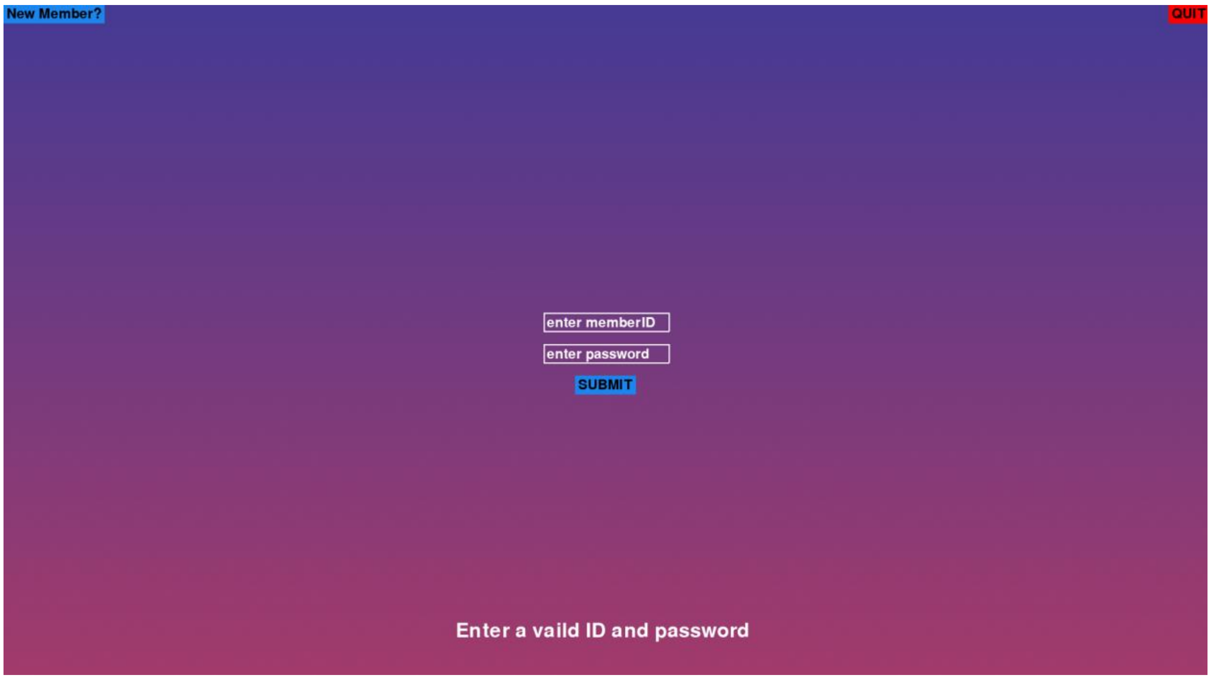


Your password must be at least 8 characters long, have at least 1 capital letter, have at least 1 special character and have at least 1 digit



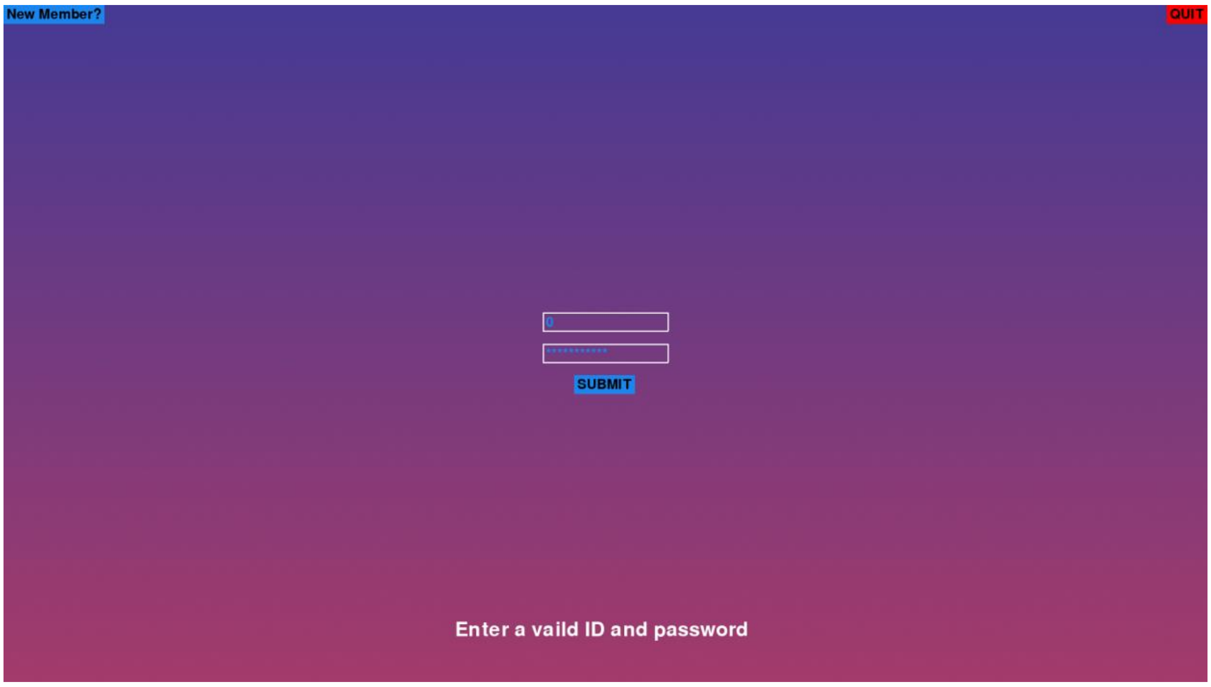
11

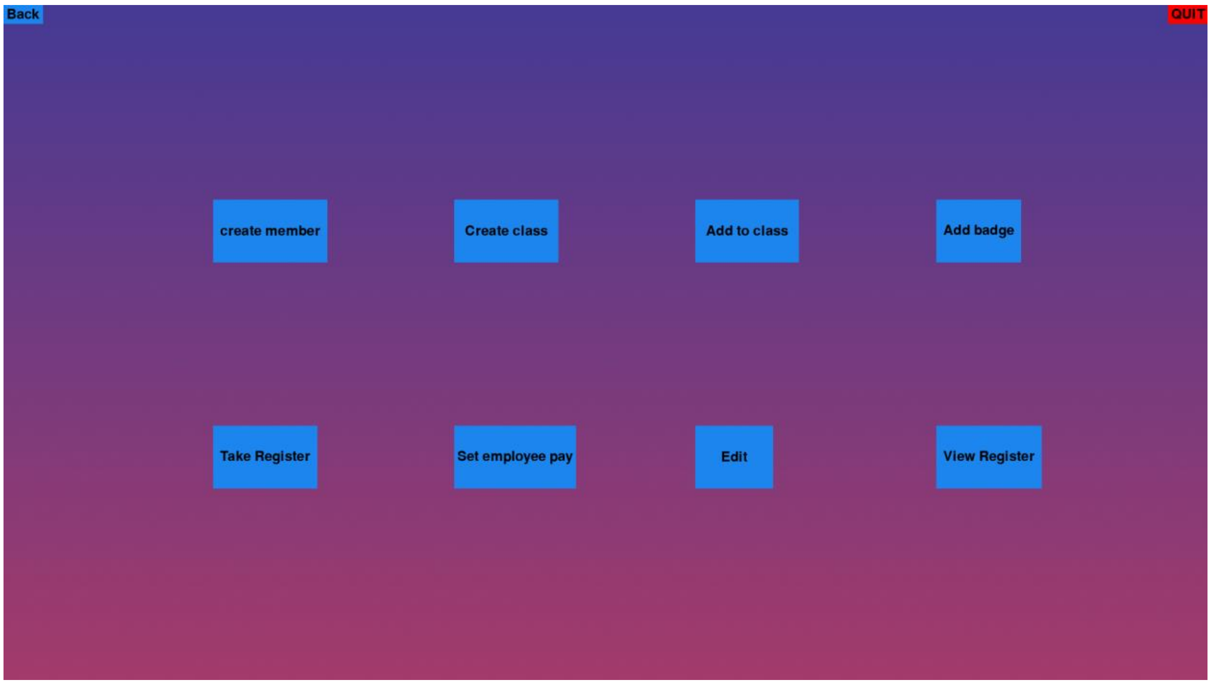




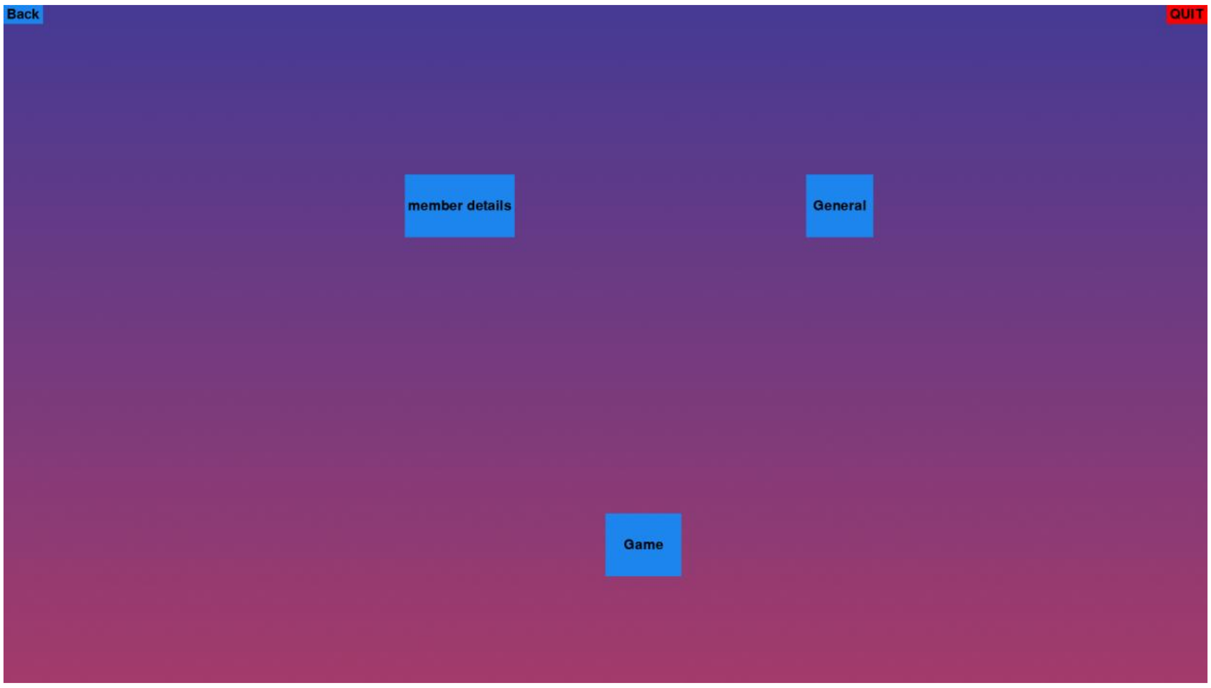
Enter a vaild ID and password

12 + 43

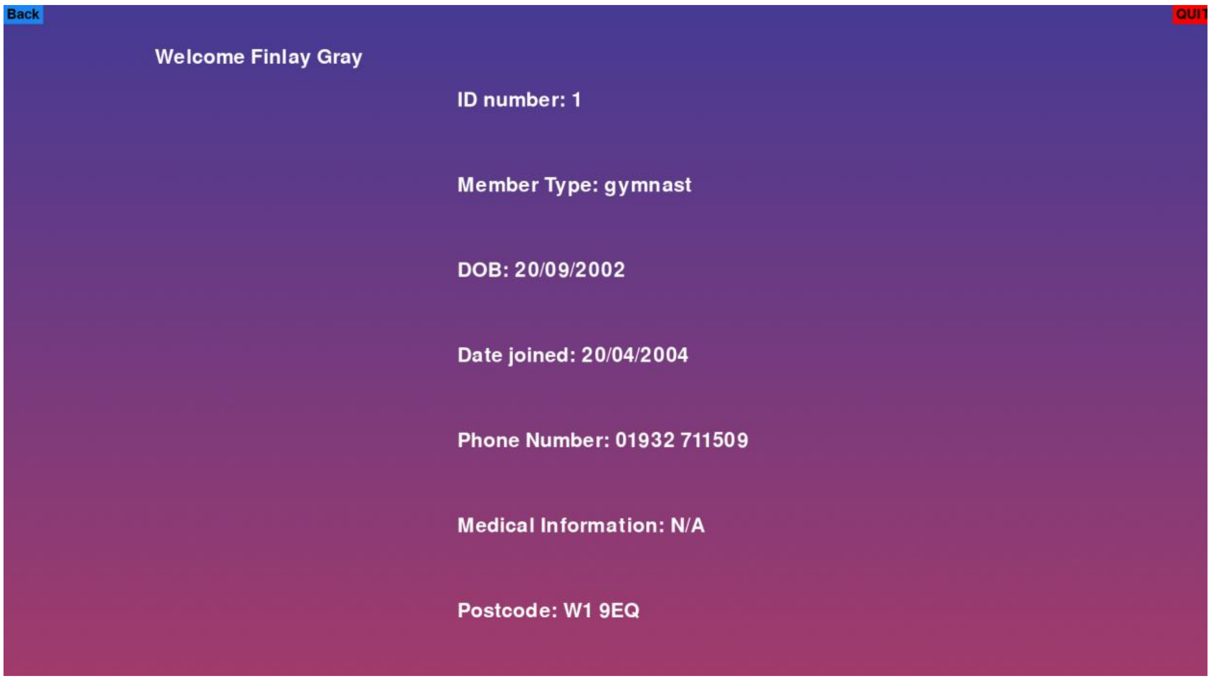

New Member?
QUIT
0
SUBMIT
Enter a vaild ID and password


Back
QUIT
create member
Create class
Add to class
Add badge
Take Register
Set employee pay
Edit
View Register

14 + 17



18



Welcome Finlay Gray

ID number: 1

Member Type: gymnast

DOB: 20/09/2002

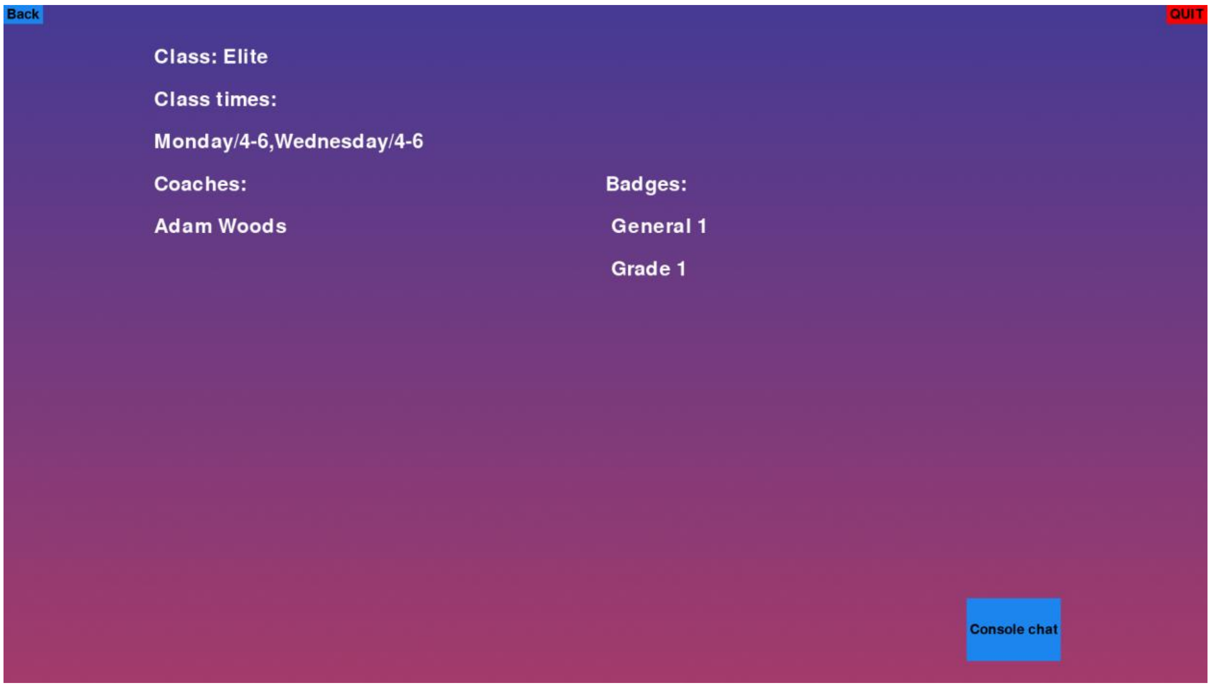
Date joined: 20/04/2004

Phone Number: 01932 711509

Medical Information: N/A

Postcode: W1 9EQ

19



Class: Elite

Class times:

Monday/4-6,Wednesday/4-6

Coaches:

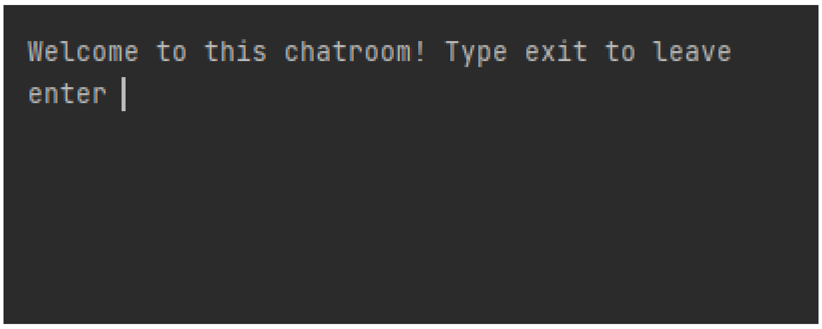
Adam Woods

Badges:

General 1

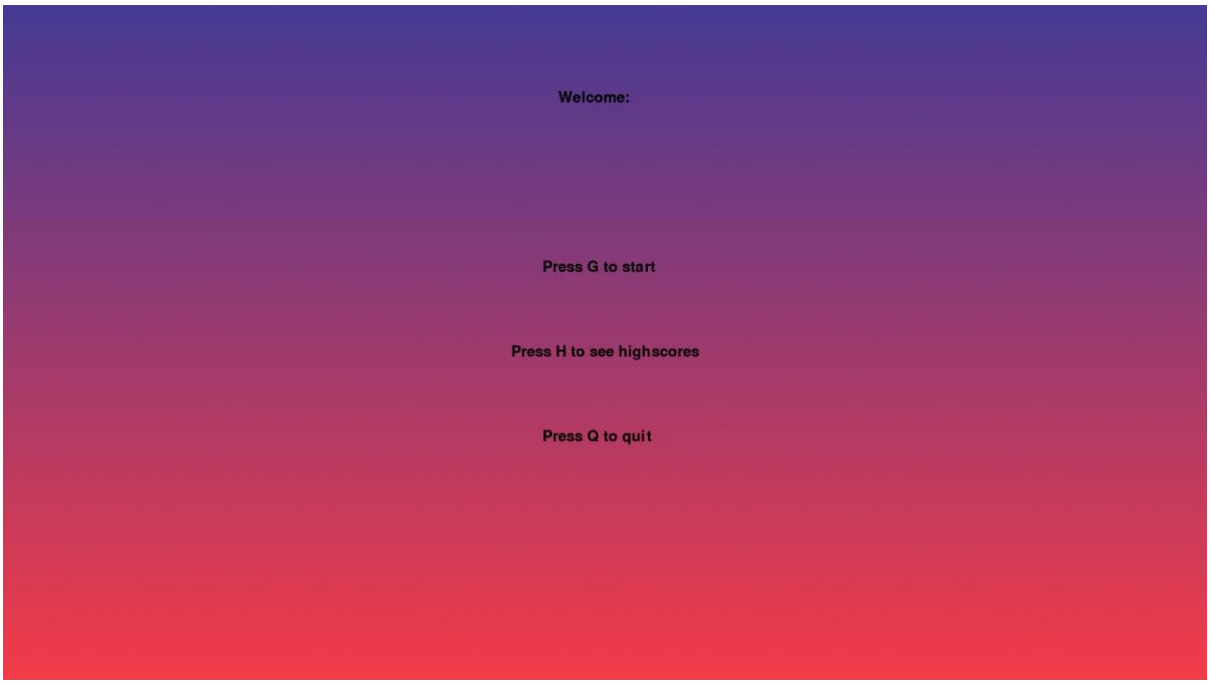
Grade 1

20



```
Welcome to this chatroom! Type exit to leave
enter 
```

21 + 22

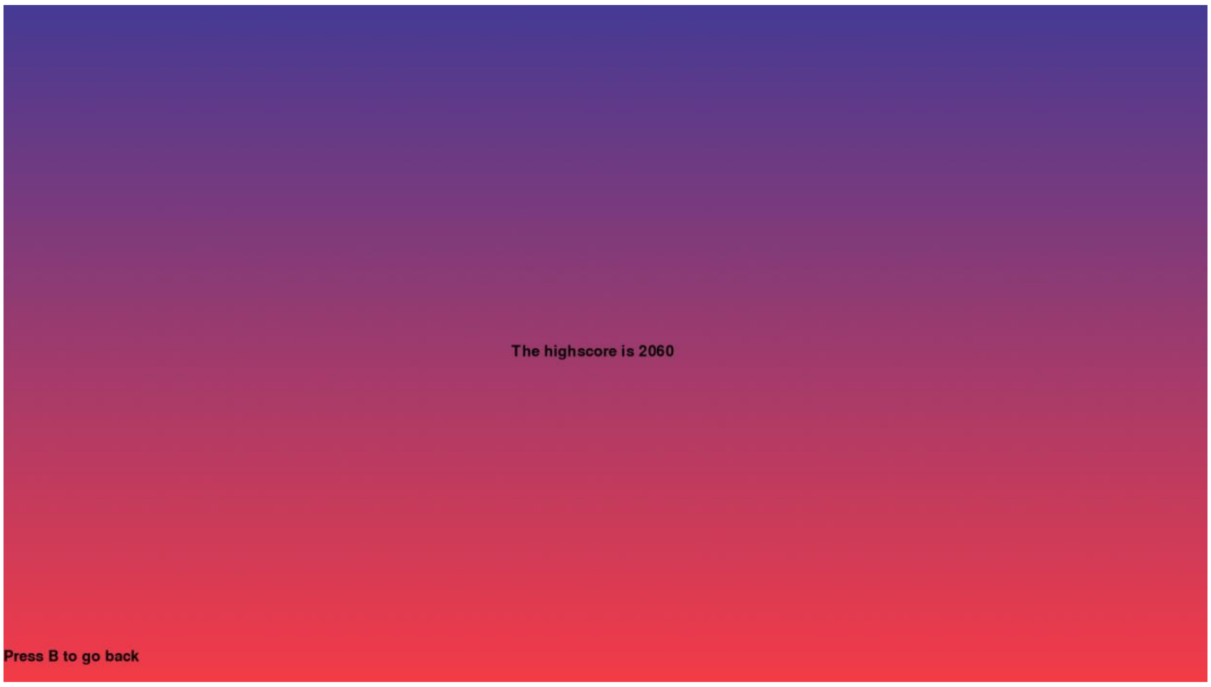

Welcome:

Press G to start

Press H to see highscores

Press Q to quit

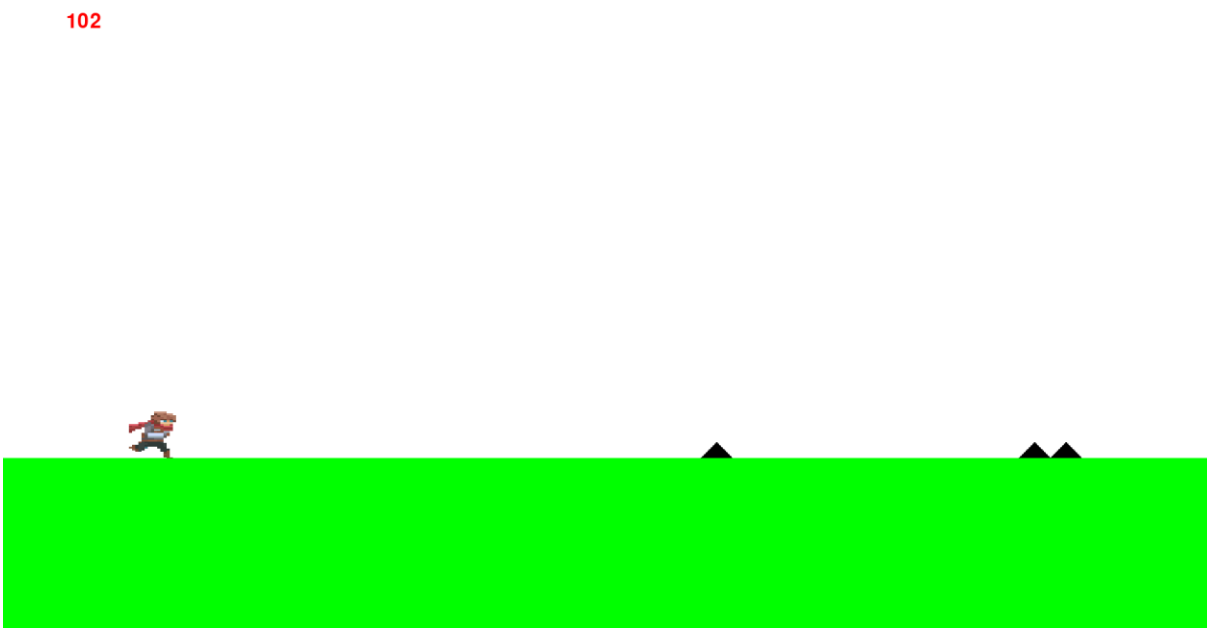
23 + 24



The highscore is 2060

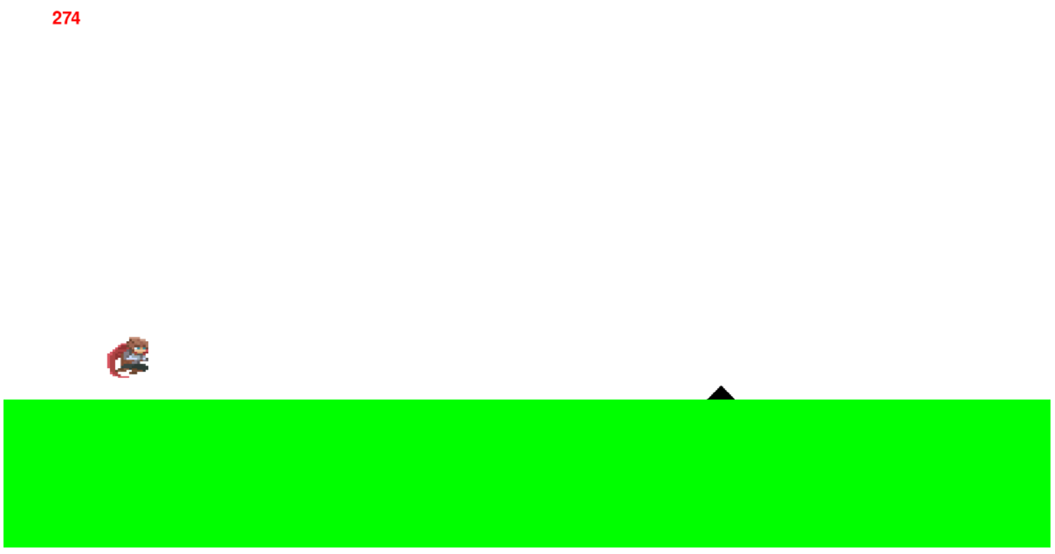
Press B to go back

25

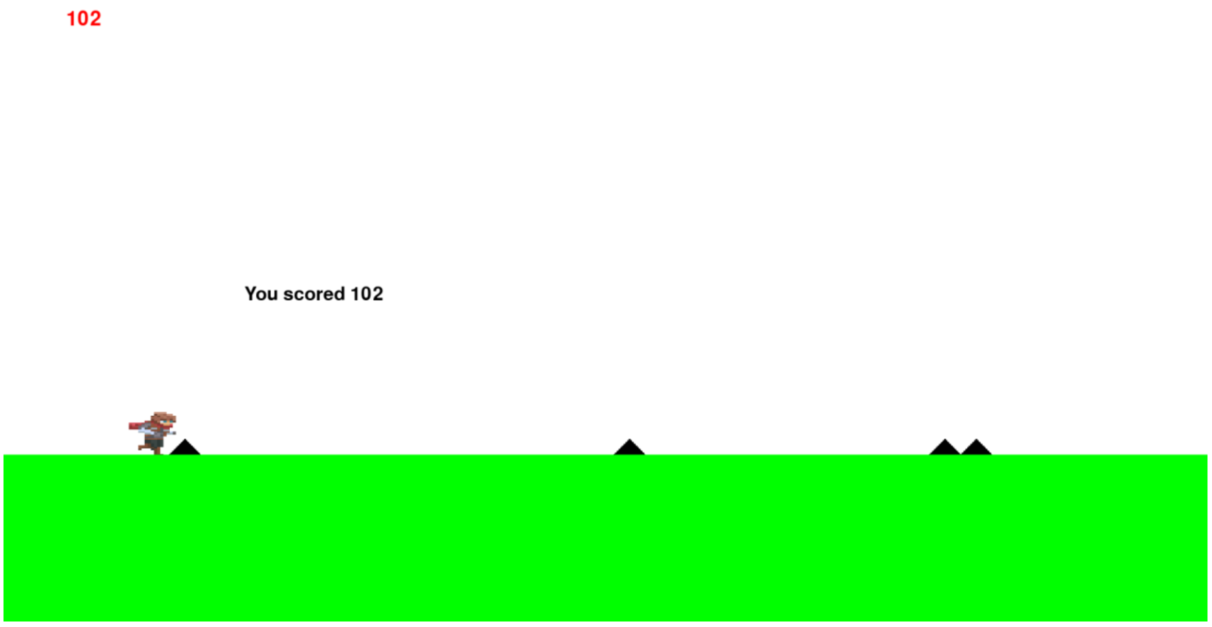


102

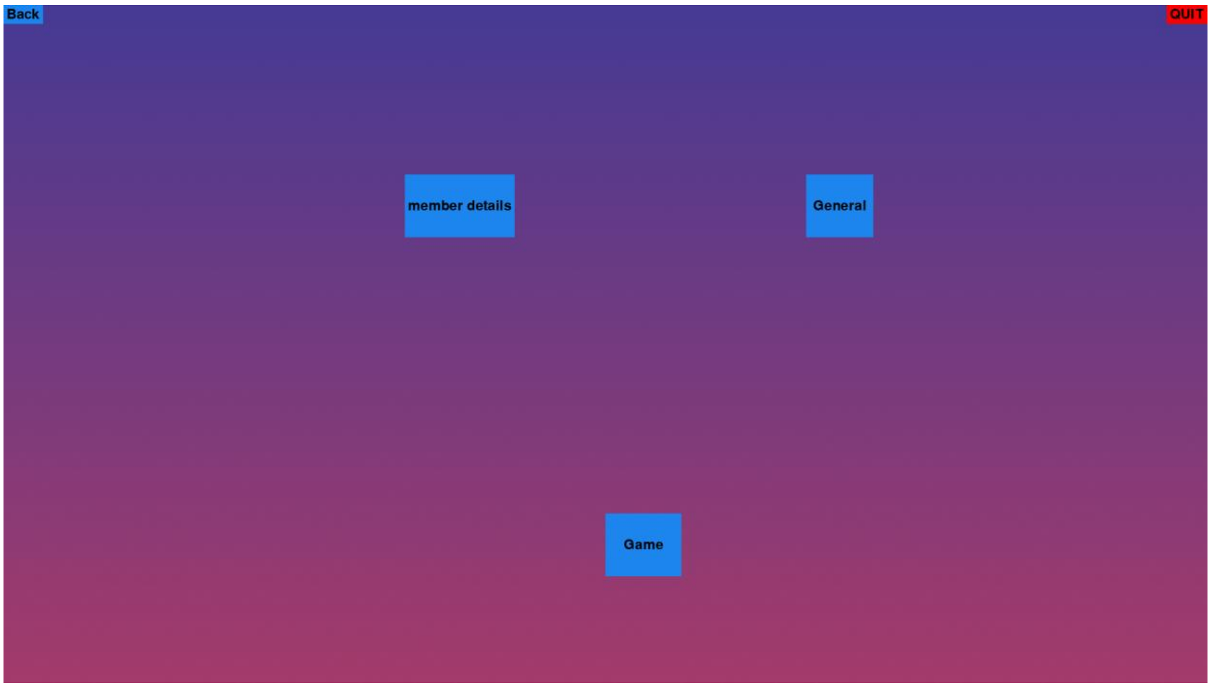
26 + 27



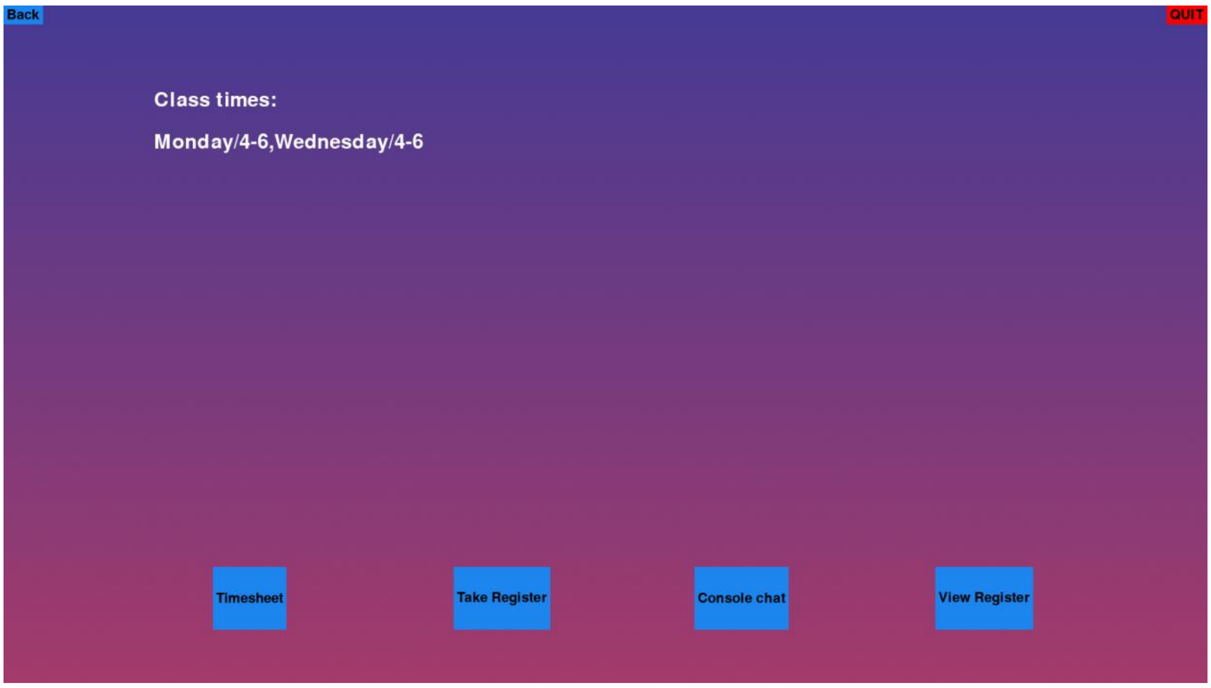
28



30



31



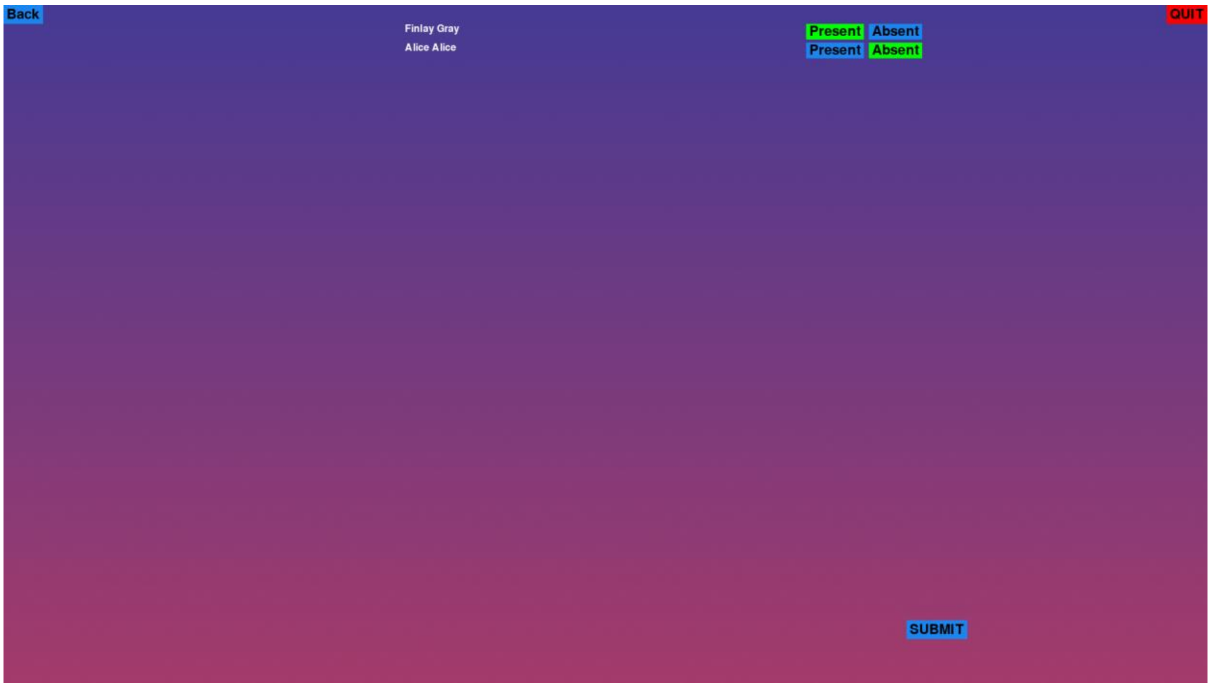
Class times:

Monday/4-6,Wednesday/4-6

32

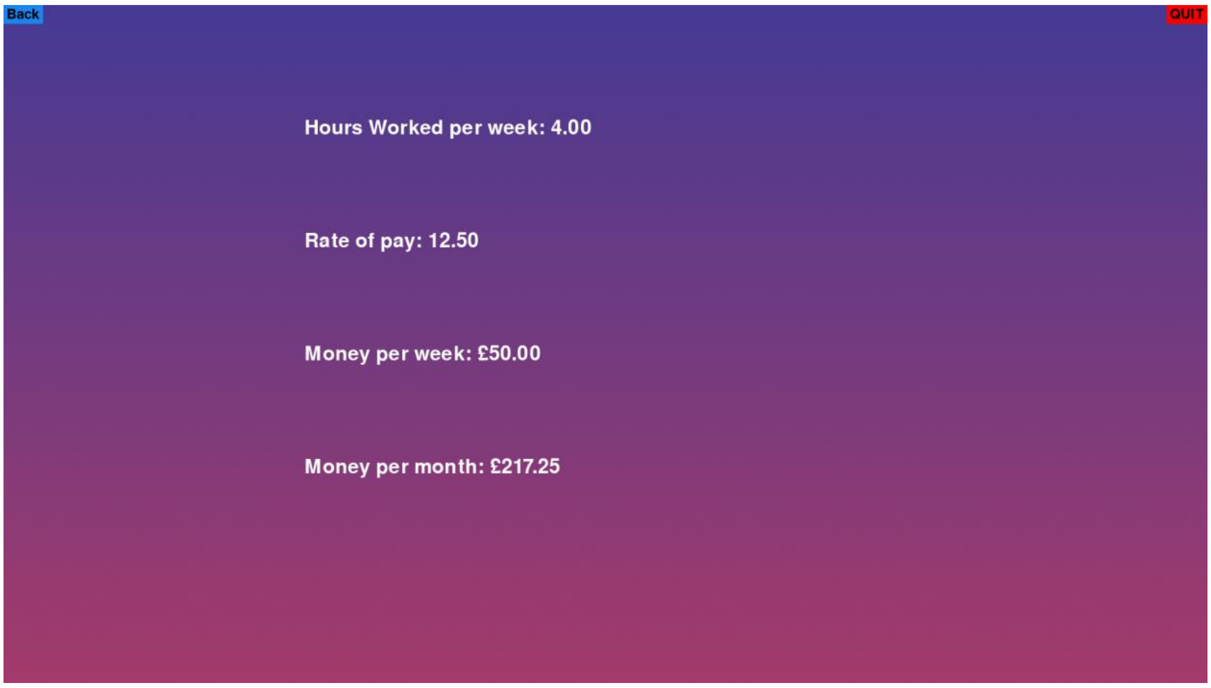
33

34-36



37

38



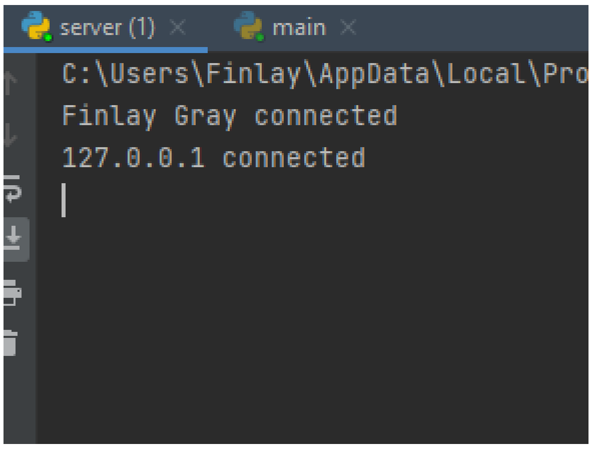
Hours Worked per week: 4.00

Rate of pay: 12.50

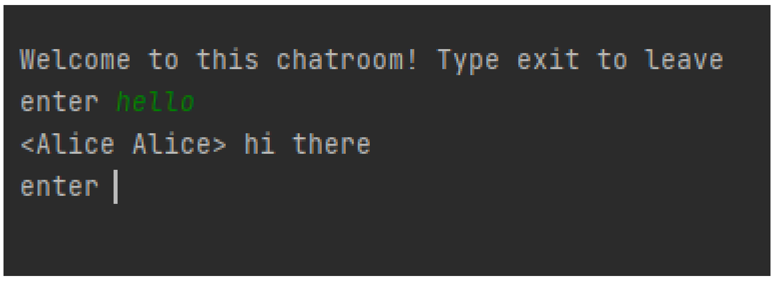
Money per week: £50.00

Money per month: £217.25

39 - 42



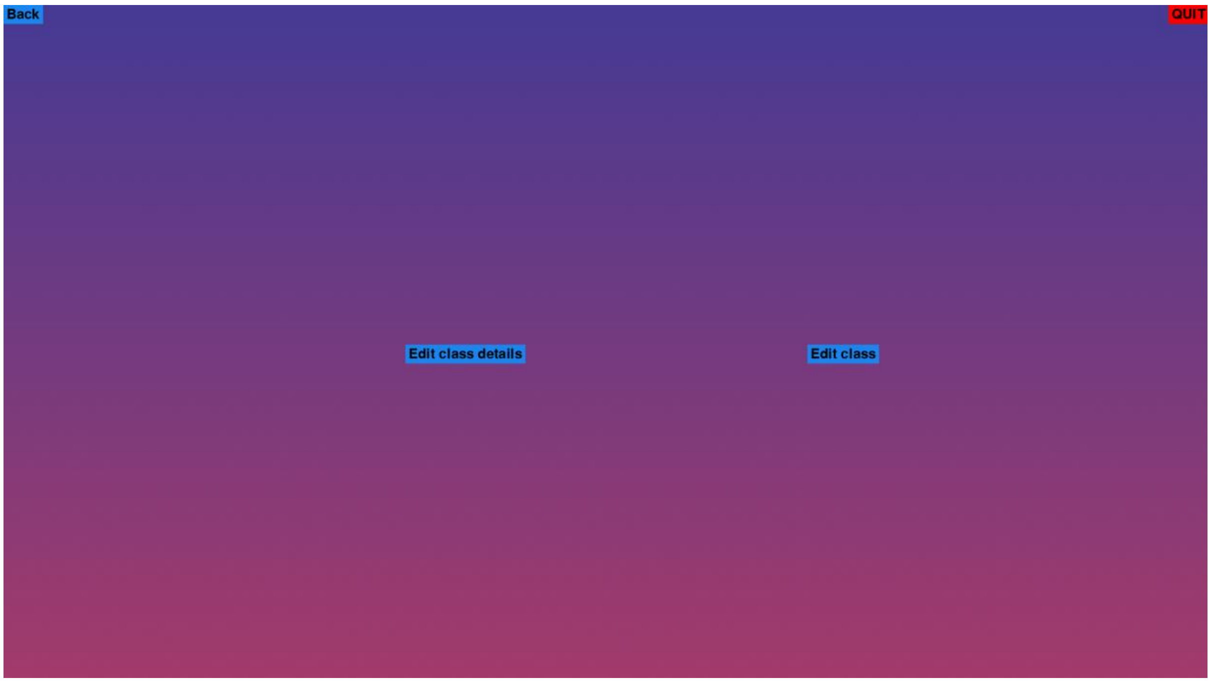
```
C:\Users\Finlay\AppData\Local\Pro
Finlay Gray connected
127.0.0.1 connected
```



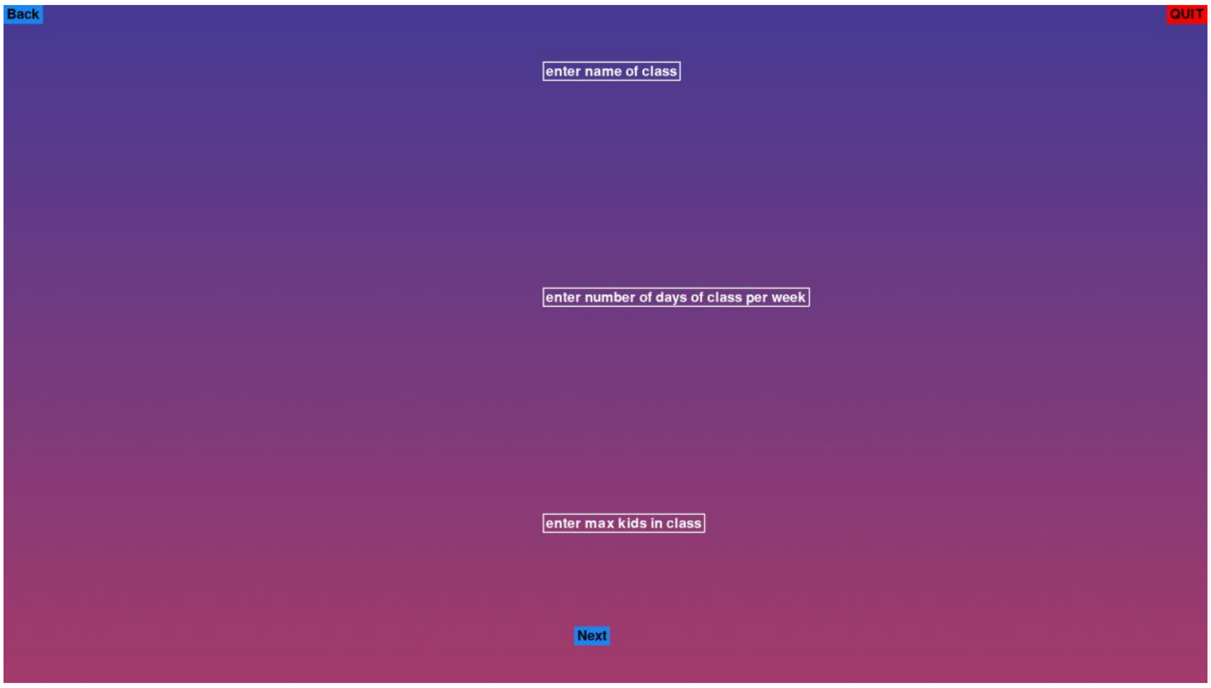
```
Welcome to this chatroom! Type exit to leave
enter hello
<Alice Alice> hi there
enter
```

44

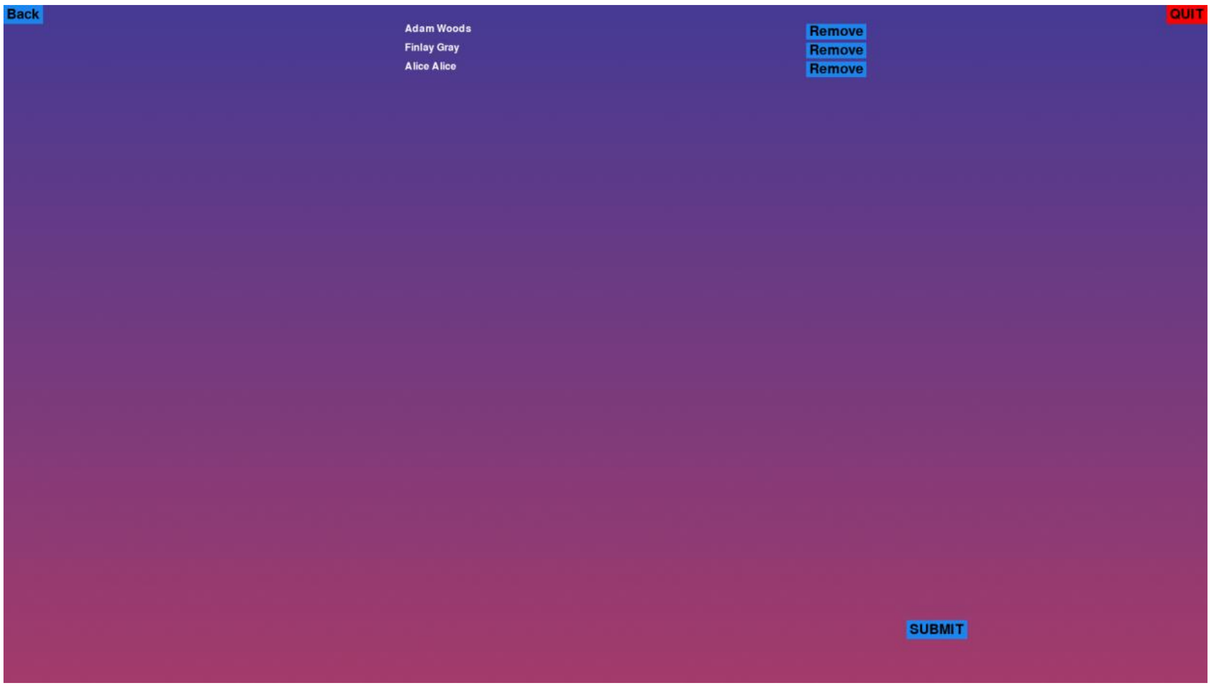
45


Back
QUIT
Edit class details
Edit class

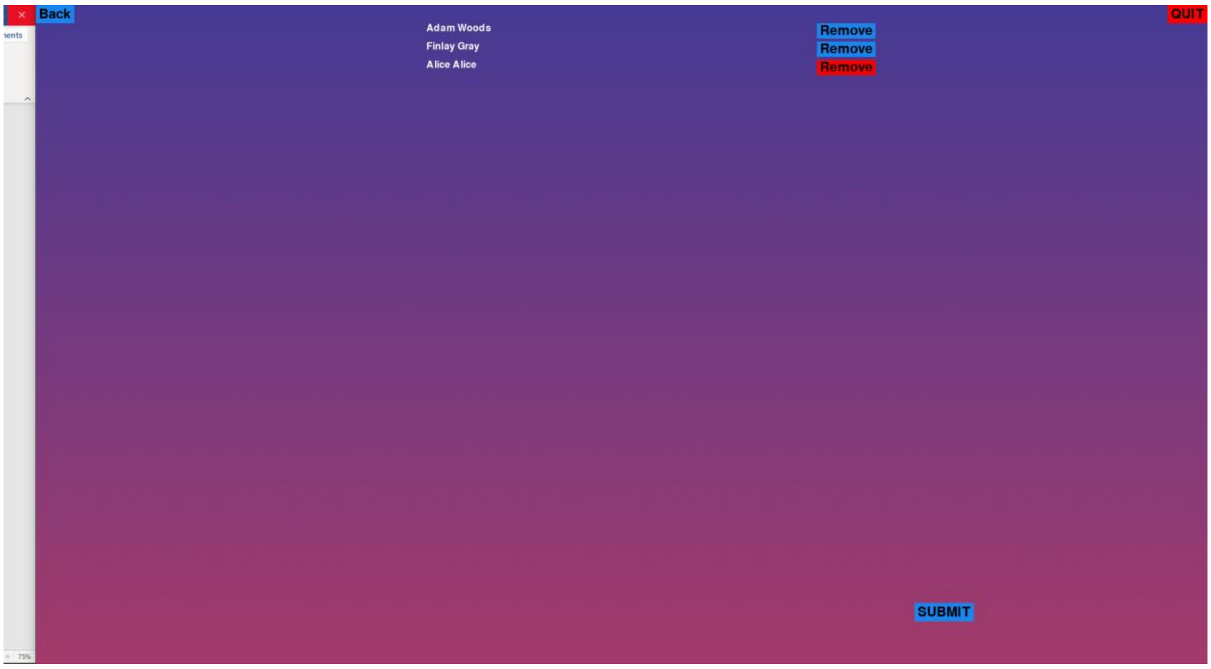
46


Back
QUIT
enter name of class
enter number of days of class per week
enter max kids in class
Next

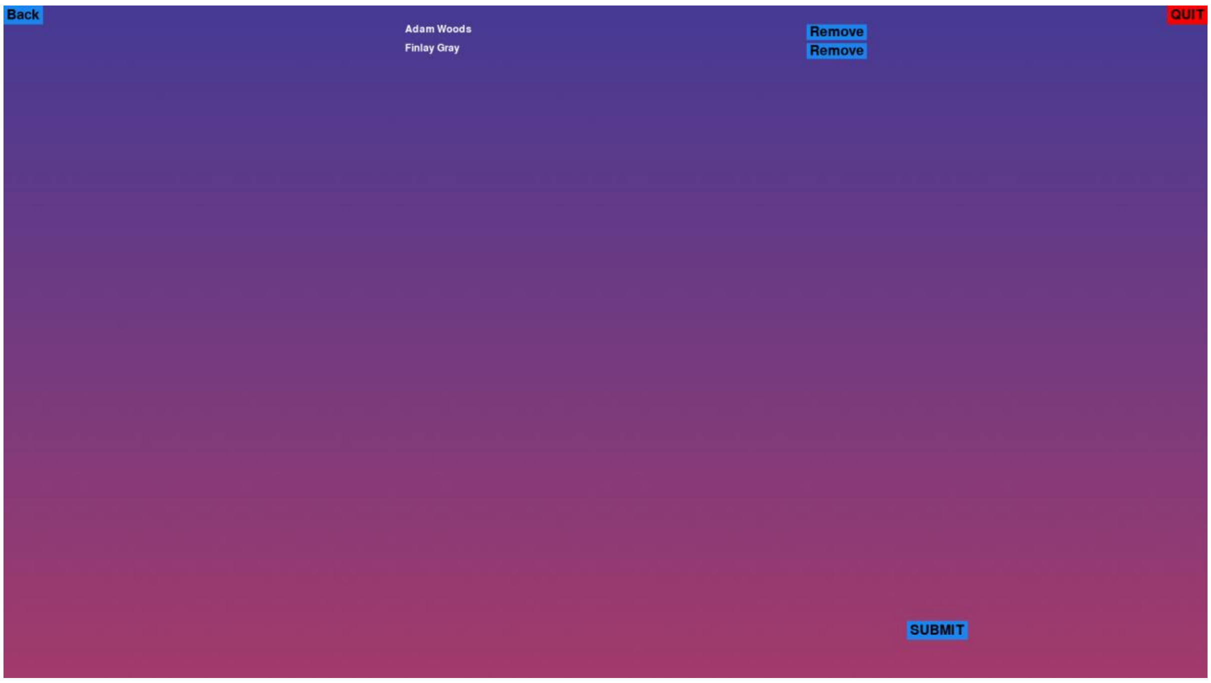
47


Back
Adam Woods
Finlay Gray
Alice Alice
Remove
Remove
Remove
QUIT
SUBMIT

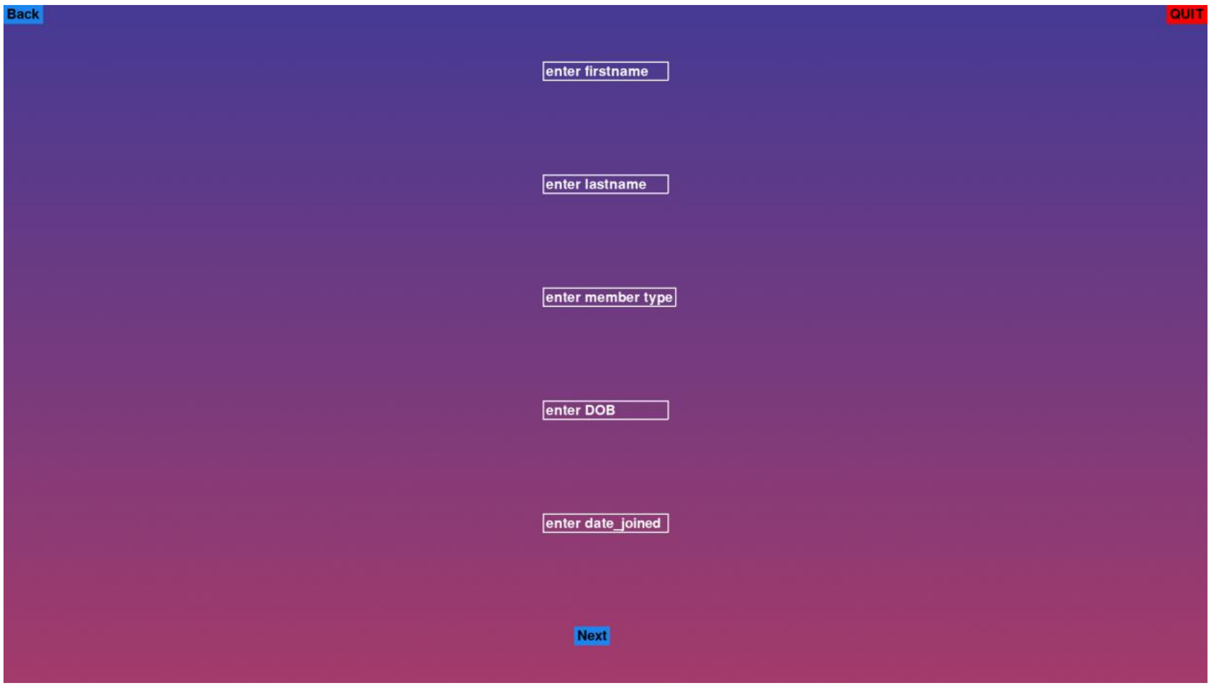
48

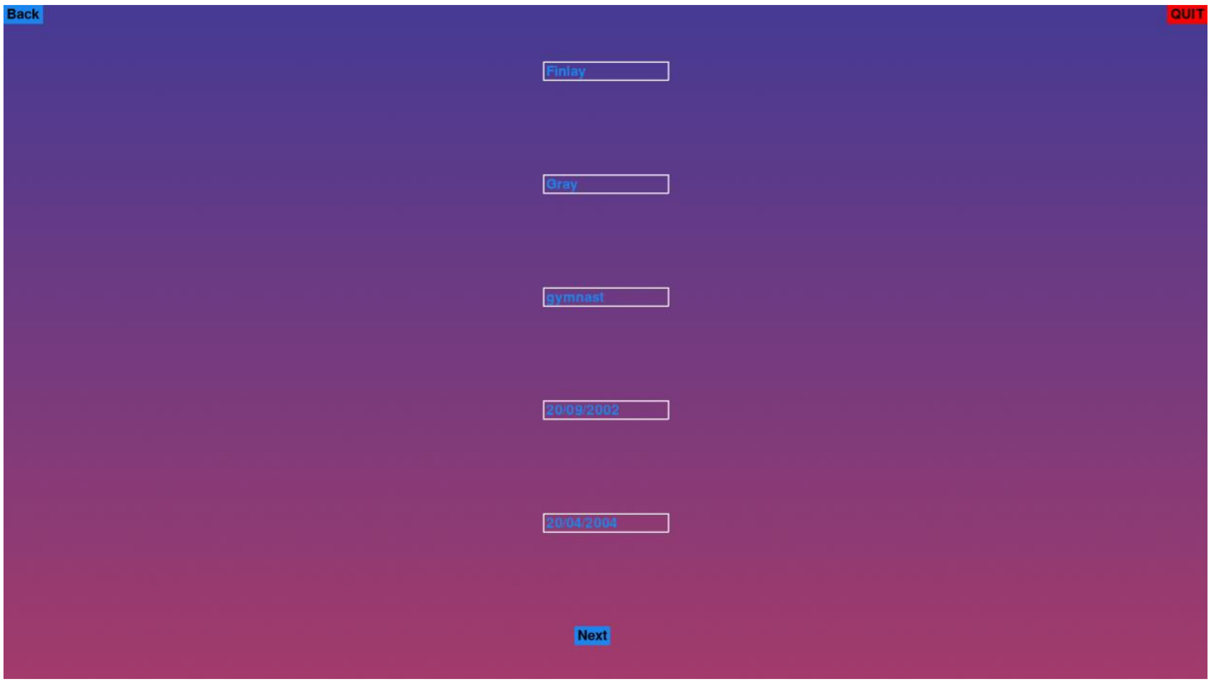

Back
Adam Woods
Finlay Gray
Alice Alice
Remove
Remove
Remove
QUIT
SUBMIT

49


Back
Adam Woods
Finlay Gray
Remove
Remove
QUIT
SUBMIT

50


Back
QUIT
enter firstname
enter lastname
enter member type
enter DOB
enter date_joined
Next



Finlay

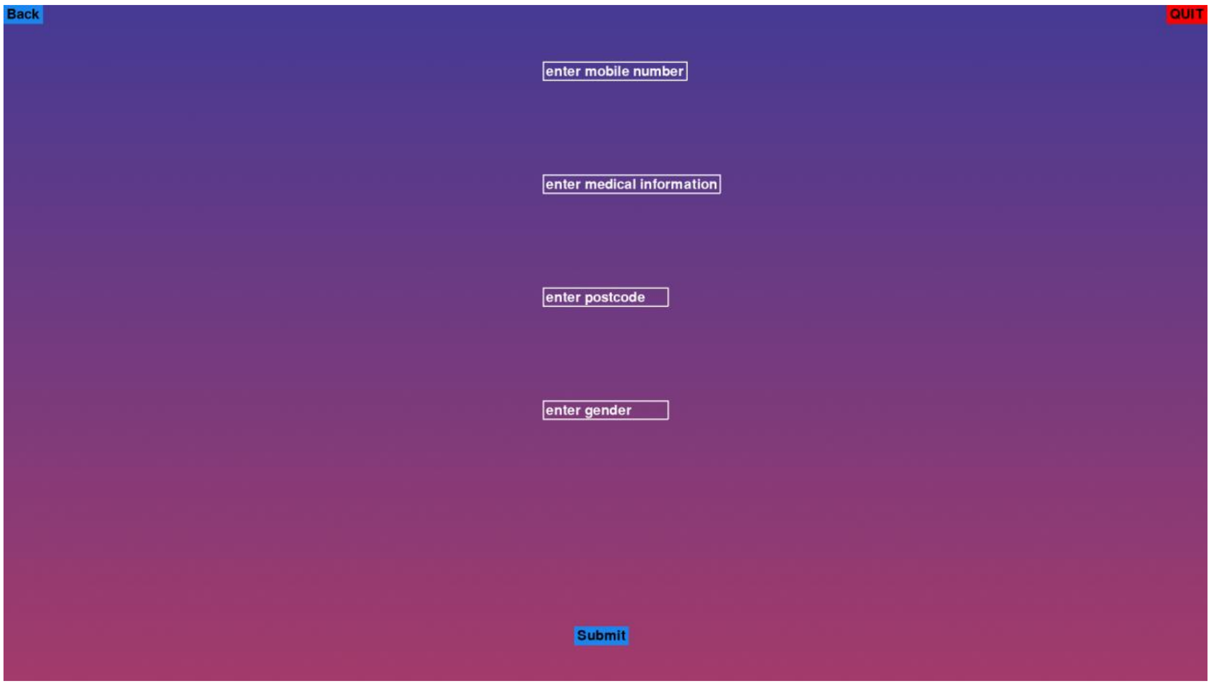
Gray

gymnast

20/09/2002

20/04/2004

51

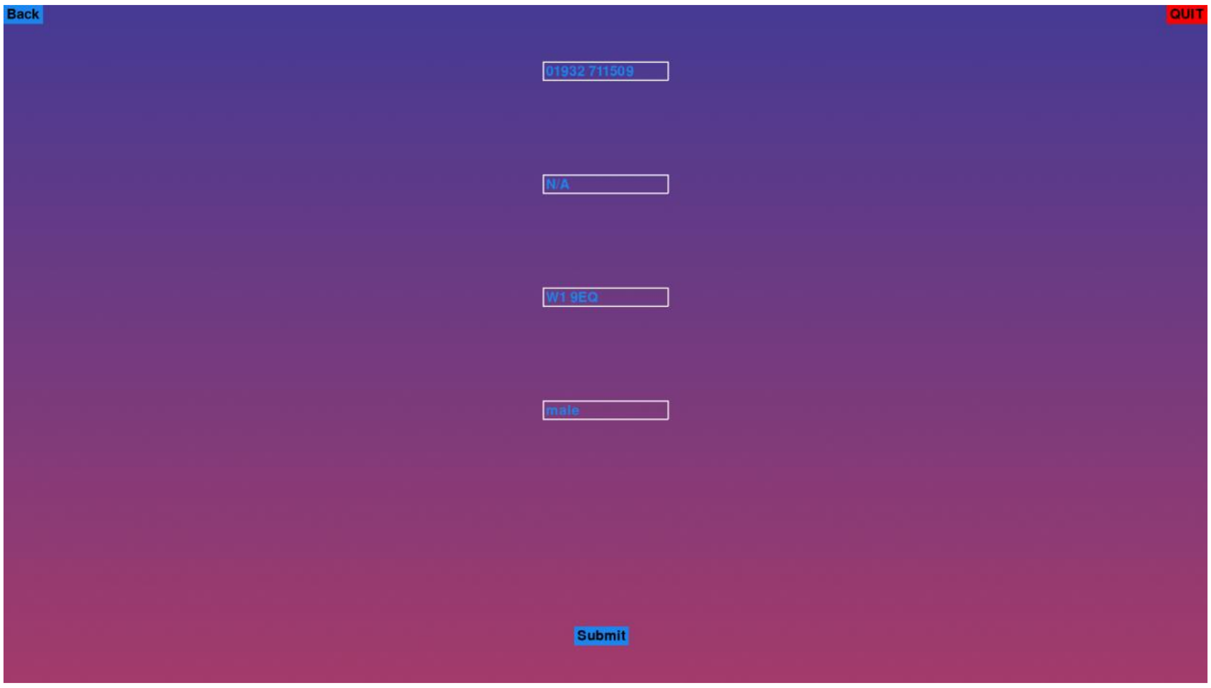


enter mobile number

enter medical information

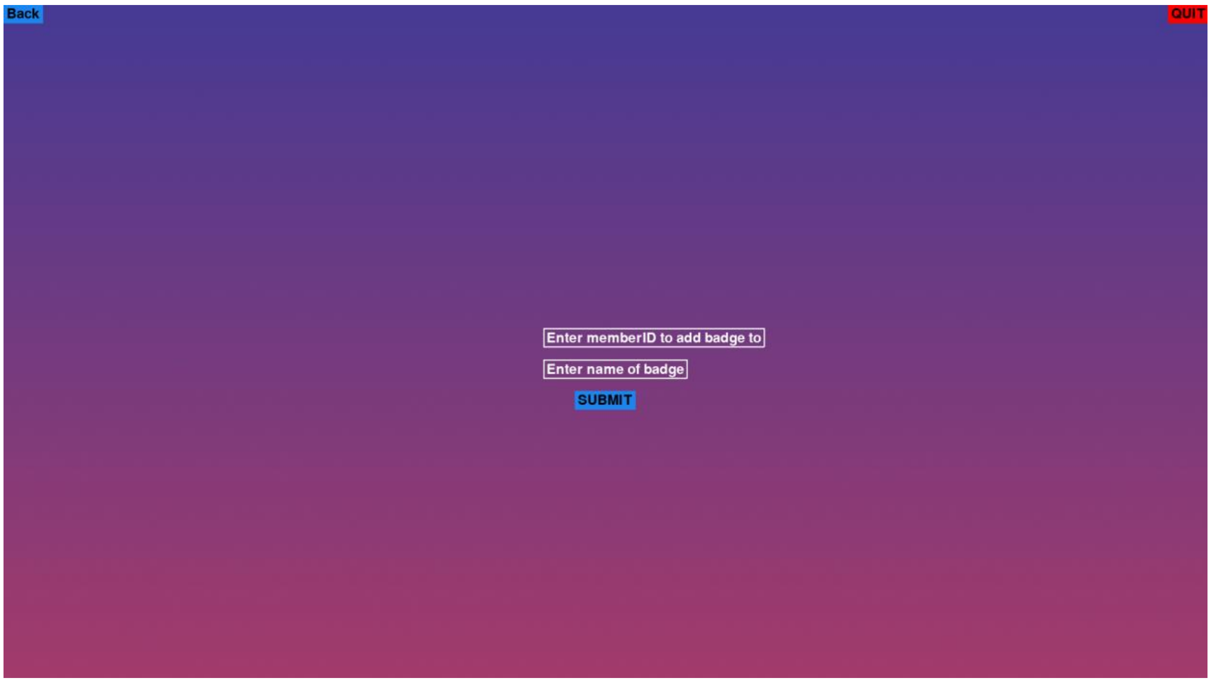
enter postcode

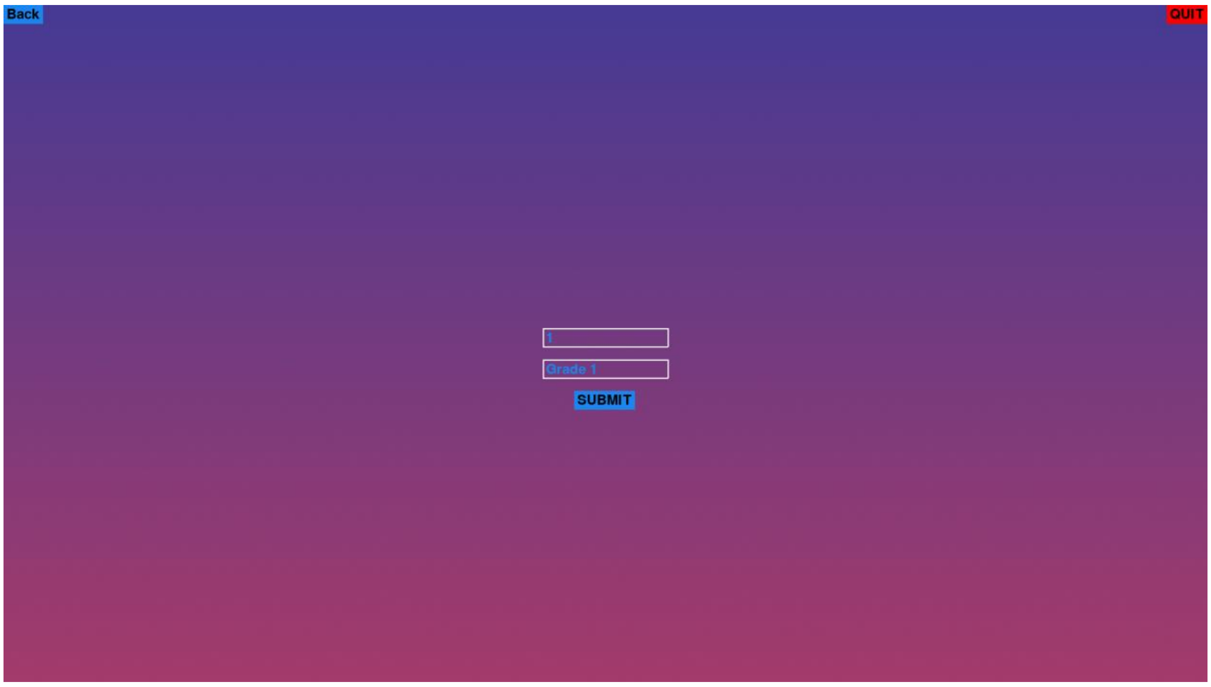
enter gender



52
As seen in earlier screenshot of user Finlay Gray

53





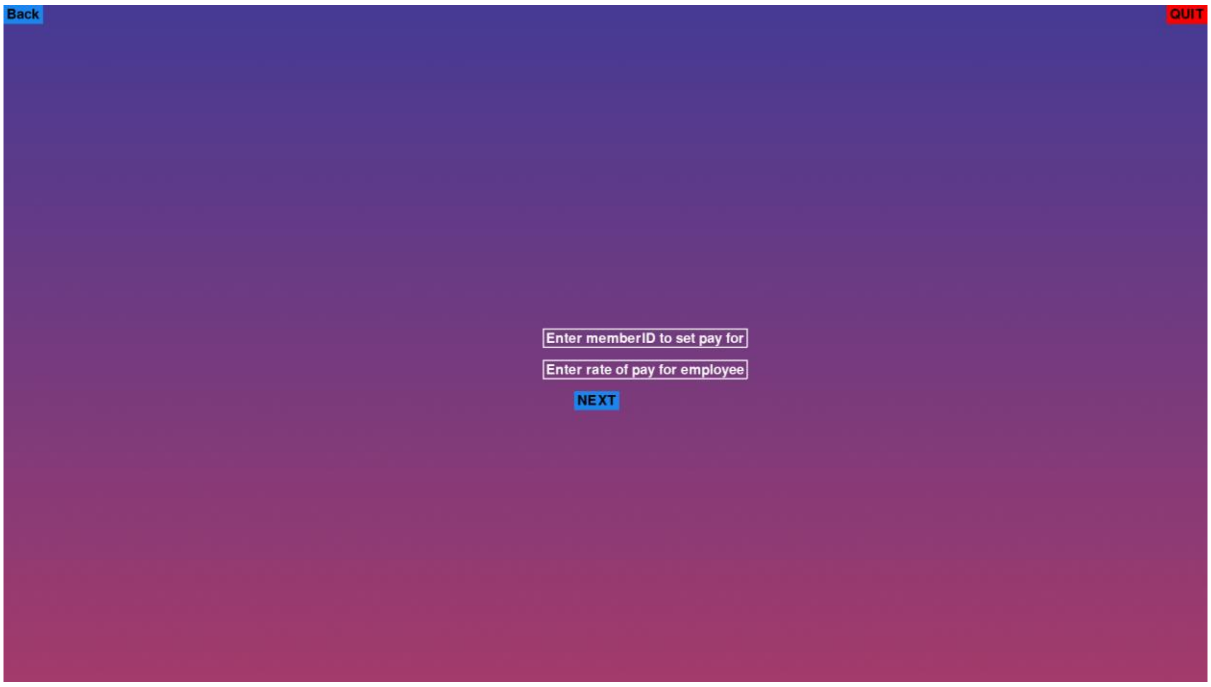
1

Grade 1

54

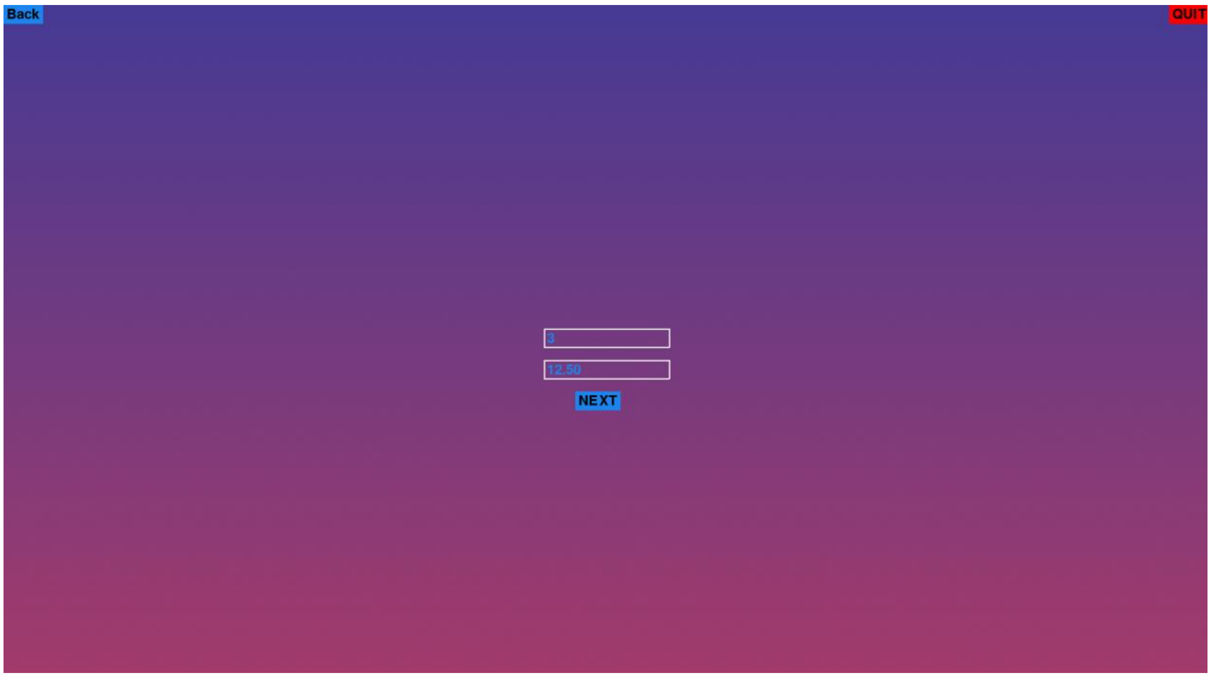
As seen in earlier screenshots of badges being displayed

55



Enter memberID to set pay for

Enter rate of pay for employee



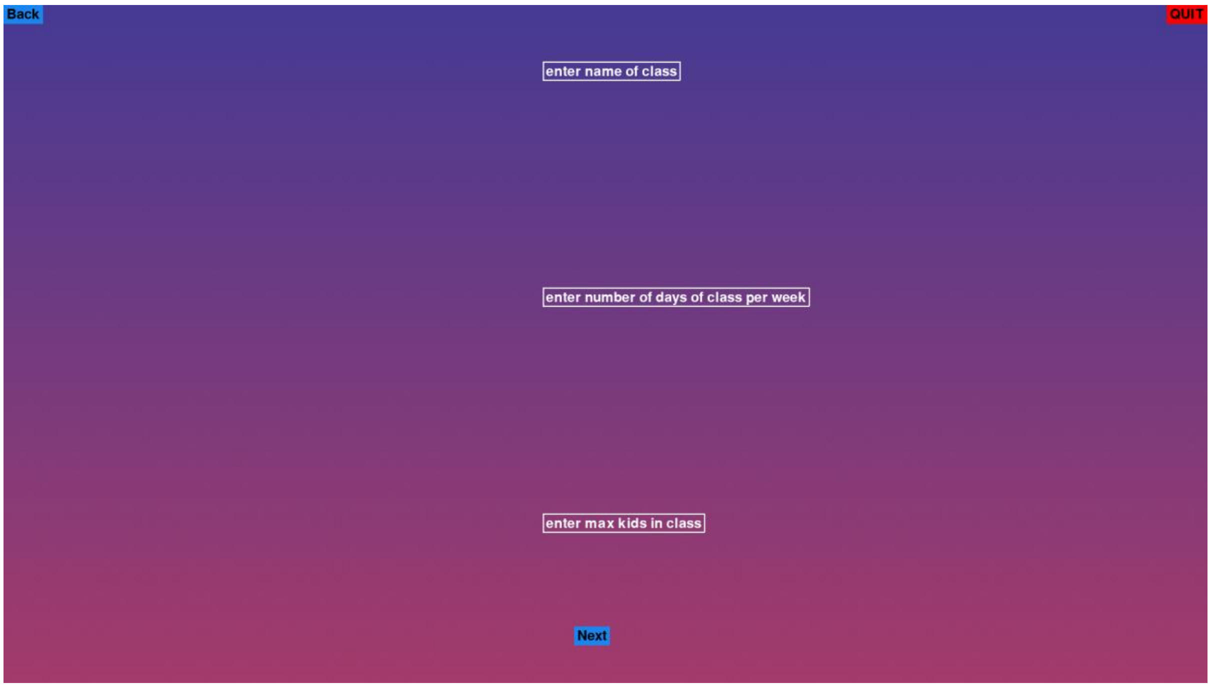
3

12.50

56

As seen in earlier screenshots of coaches ‘timesheet button’

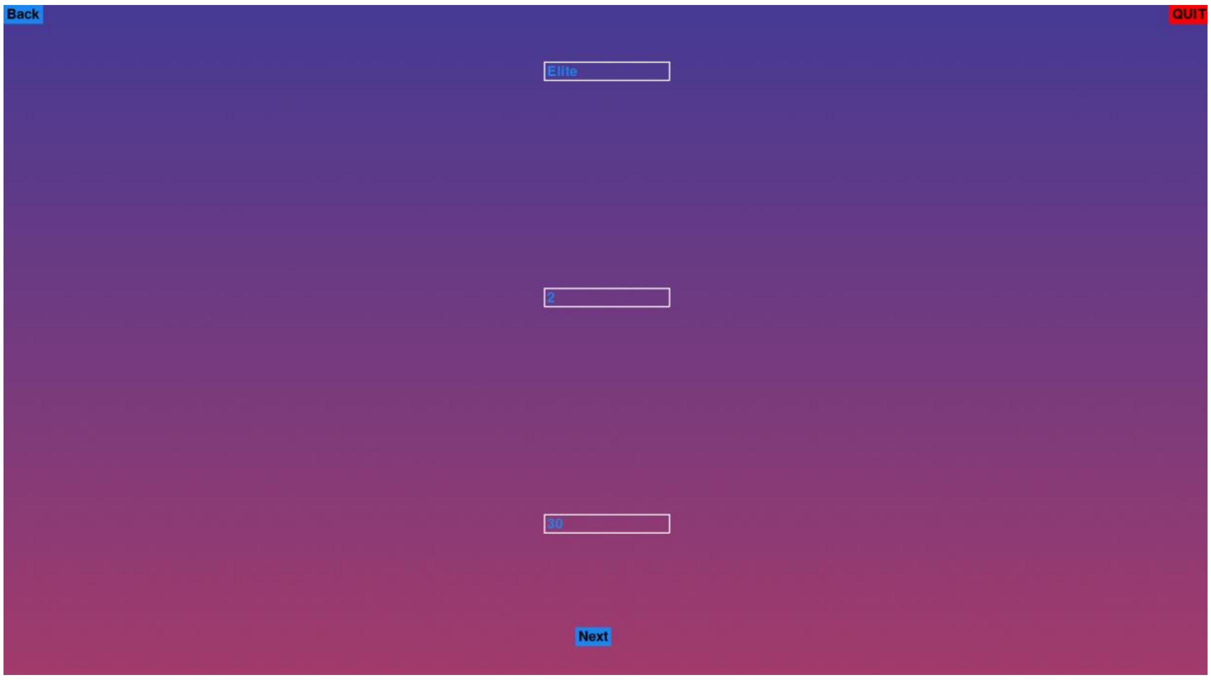
57

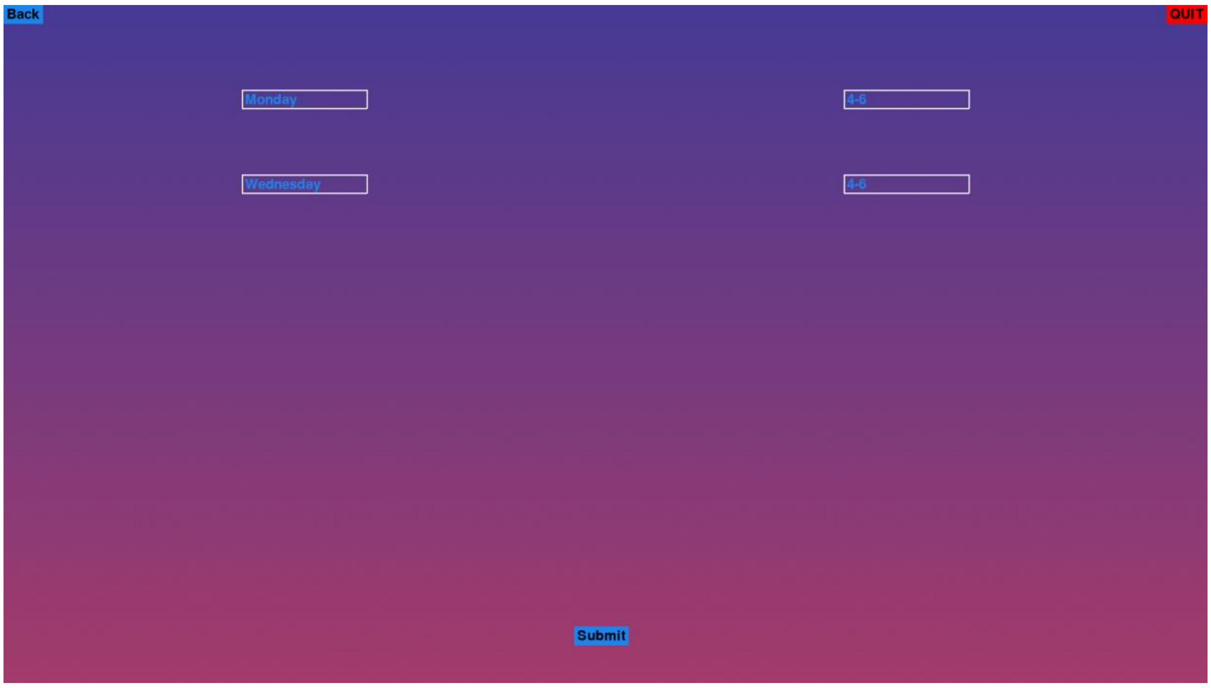


enter name of class

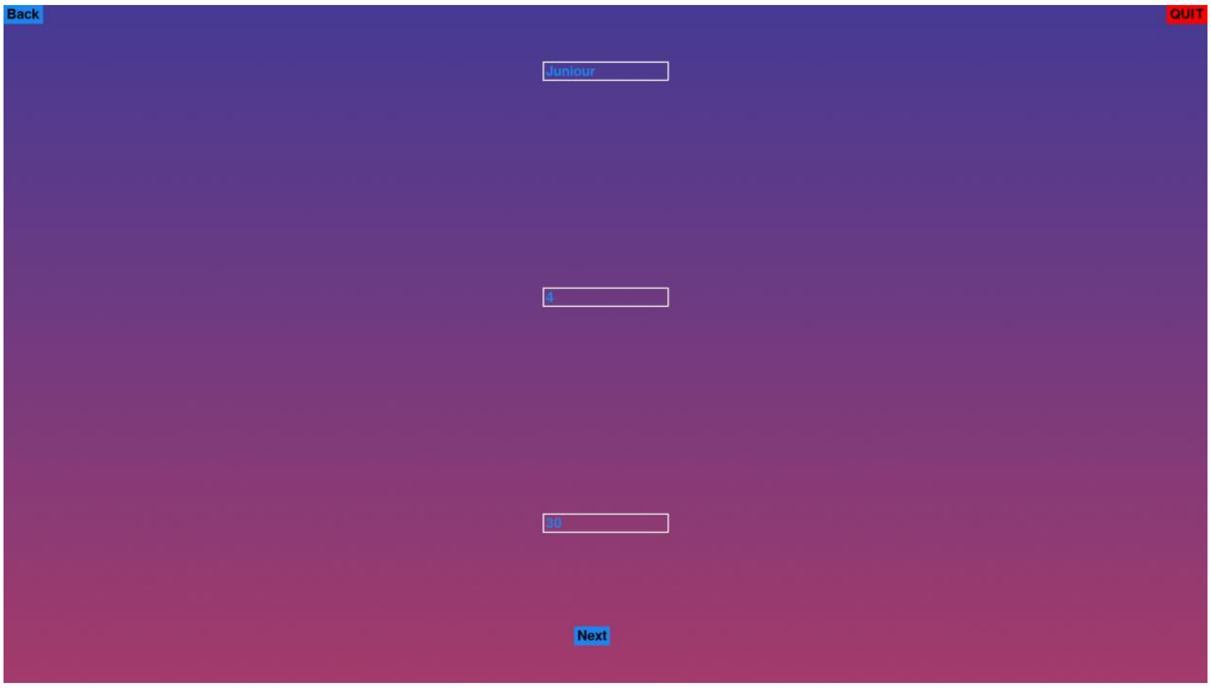
enter number of days of class per week

enter max kids in class


Back
QUIT
Elite
2
30
Next

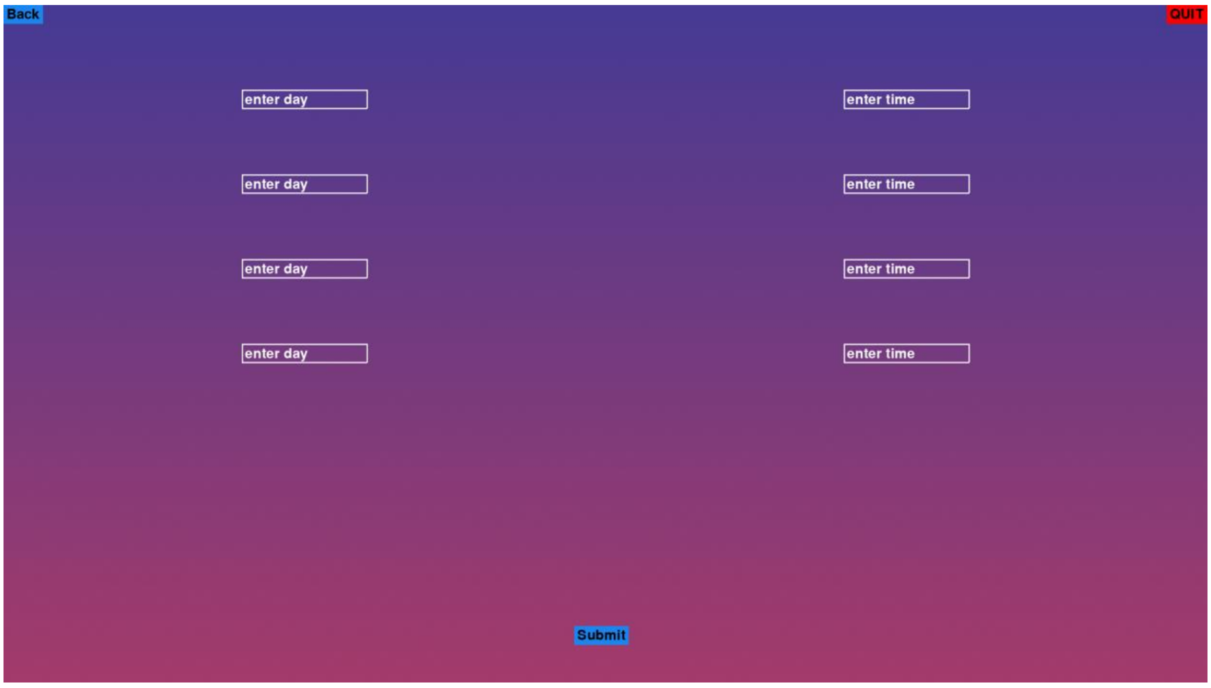

Back
QUIT
Monday
4-6
Wednesday
4-6
Submit

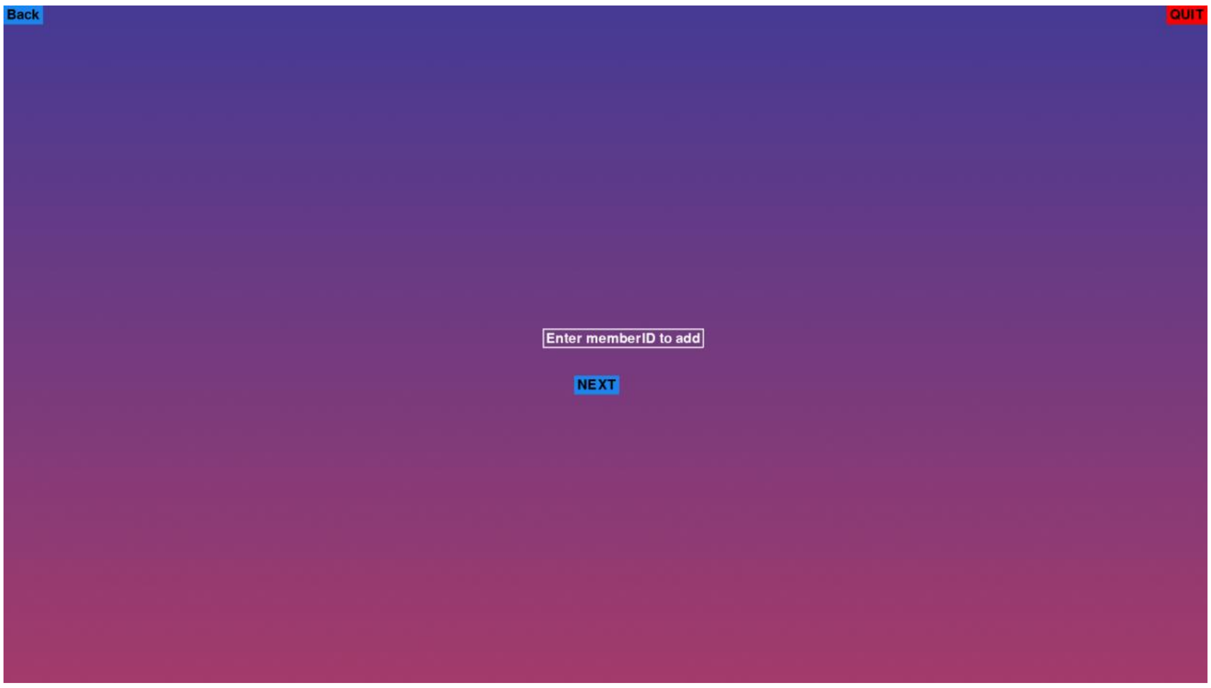
59


Juniour

4

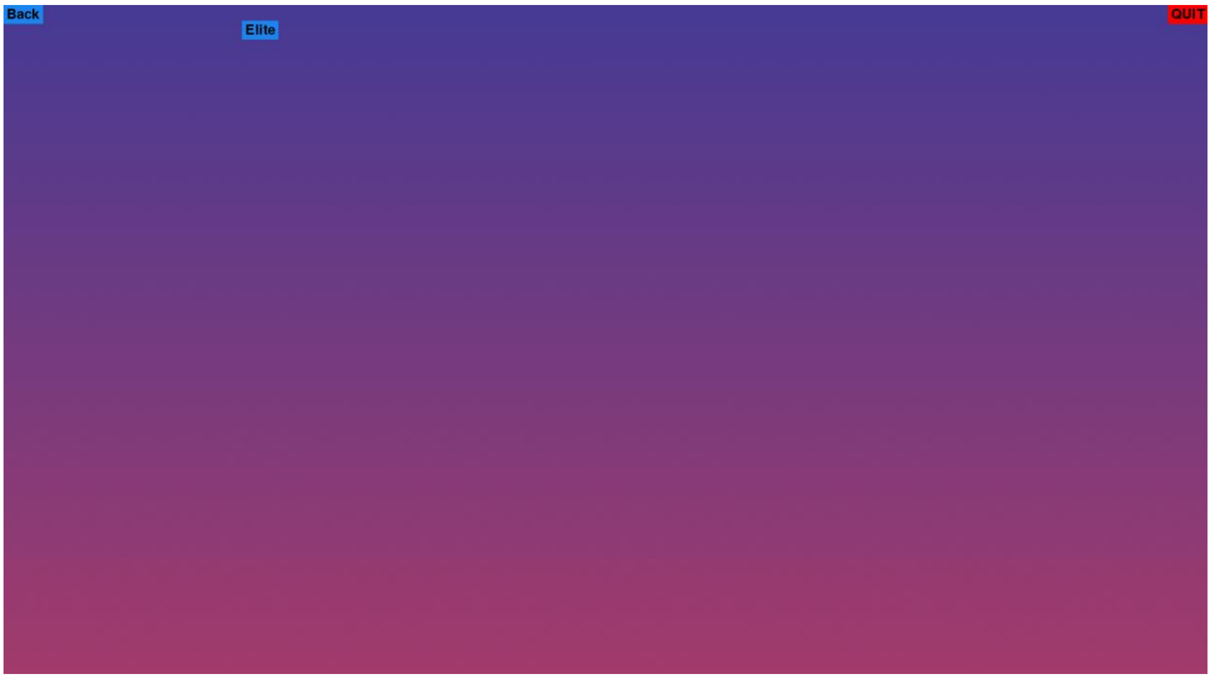
30




Back
QUIT
Enter memberID to add
NEXT

Back
QUIT
NEXT

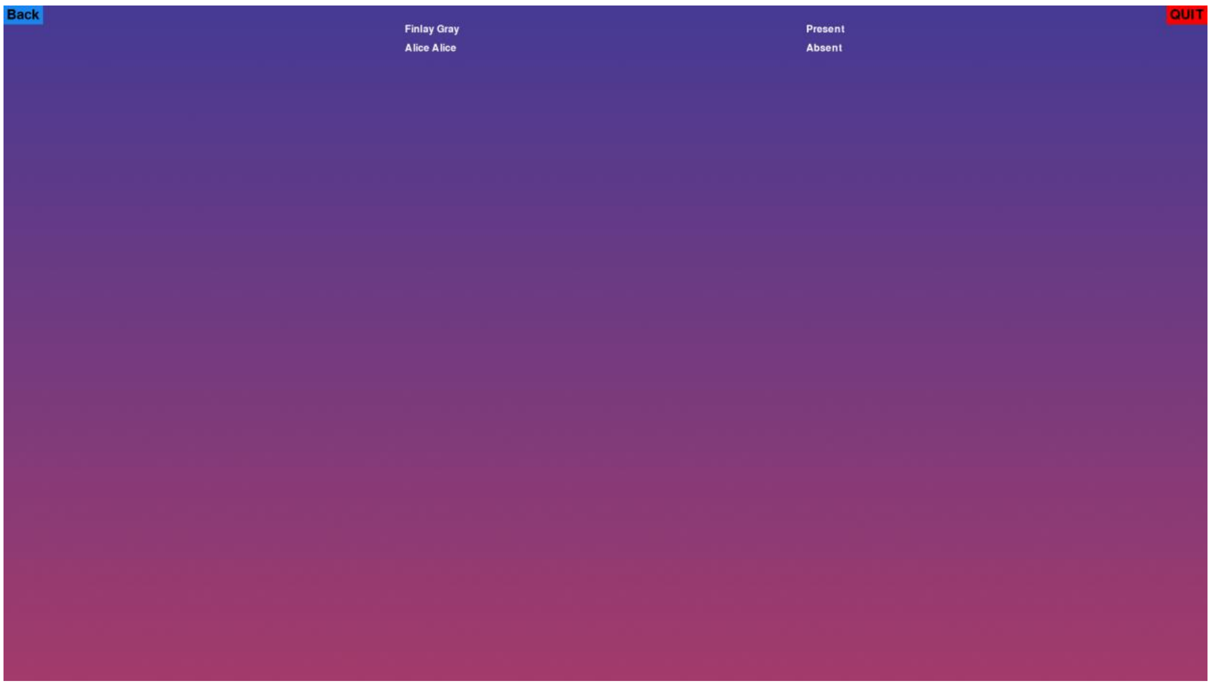
61



62

As seen in earlier screenshots of user Finlay Gray in elite class

63



Finlay Gray	Present
Alice Alice	Absent

Evaluation

In order to evaluate my project, I will go through the high level requirement to see what I achieved.

Must haves

- Each gymnast/coach/admin must have a log in which gives them access to the system.

I was able to achieve this by using a membership type data row in my table which allowed me to differentiate between gymnasts, coaches and admin.

- When creating an account there must be a way to fill in details about the person that need to be stored.
- Admins must be able to create the accounts.
    - Must be a way for them to easily fill in all required details.

I was able to achieve this by using input boxes to enter such details when creating a user in the admin account only. This means no one else but the admin can make these accounts.

- There must be a timetable displayed to gymnasts and coaches when they log on showing their classes.

I was able to achieve this for both gymnasts and coaches allowing them to view what times their classes are running.

- There must be a way coaches and admin can view and take registers.

I was able to achieve this for both admin and coaches having buttons for both on their own base page screen. They can take and view different registers depending on what class they want to view or register, and what time of that class they want to view and register.

- There must be a way admin can edit classes by adding and removing both gymnasts and coaches to and from classes.

I was able to achieve this by, for adding to class having a separate 'add to class' button allowing the admin to enter the member id of the user they want to add and then pick a button from the list of classes that pop up. For the removing of gymnasts there is an edit button where you can view all the members in the class and remove any you wish.

- There must be a way admin can change class times and requirements for classes such as max number of gymnasts.

I have achieved this by allowing the admin to re-create the details of the class in order to update its details.

- It must be user friendly.
    - Suitable for primary school children to adults

I have achieved this by making the UI very clean, simple and easy to navigate, however I do not believe it looks childish and so it is suitable for both adults and children. I have also put a minigame in the gymnast section to allow the system to be even more catered for kids.

- There must be a way that gymnasts can view all their details as well as what class they are in and their coaches.

I have achieved this by splitting it into two sections. There is a member details button where the gymnasts can view all their details and a general button where they can view their coaches.

- There must be a way that users can create their password when first logging on given their already known membership id.

I have achieved this with a new member button which shows up on the log in screen to allow the user to set up a password.

Should haves

- There should be an automatic system which works out how many hours a coach has worked and how much money they will get at their specific rate.

I have achieved this in the ‘timesheet’ section of the coaches’ page in which I have also worked out their weekly and monthly rate.

- Admin should be able to change the pay rate for all staff.

I have achieved this with the 'pay' button on the admin page which allows the admin to enter a member id and then set the hourly rate for that member.

- There should be a way admin can give out badges.

I have achieved this with the 'pay' button on the admin page which allows the admin to enter a member id and then add a badge that they can name.

- There should be a way the gymnasts can view their badges they have earned.

I have achieved this by allowing the gymnast to view their badges in the general section of their page.

Could Haves

- There could be small minigame available to the gymnasts, such as a simple 2d run and jump game with the character doing a flip or something gymnastics related.

I achieved this by creating a simple jumping game where you must avoid spikes by jumping over them. Also, every time you jump the character does a flip which has a relation to gymnastics.

- There could be a way gymnast can speak to their coaches and other gymnasts in a forum type chat and vice versa.

I achieved this by creating a separate server program to allow the users to speak with one another in the console.

Final

In order to finally evaluate my project, I took it back to the general manager of my gym, who I interviewed initially, and my brother who would be a typical user for the gymnast side of the system.

The general managers response was that the system in terms of the main display looked simple and effective which was what she wanted.

For the admin side the response was that it performed well, and all the buttons were of great use, effective and easy to use and navigate. A bit of feedback that she gave me was that she found the buttons to view and take the register were a bit long as they step through the same pages even though they are two different buttons on the main screen, so a possible future enhancement would be to make the choice to view or take registers after selecting the class and class time.

The general managers response for the coaches' page was good as she liked the simplicity of the timetable and the fact that the coaches could easily view their rate of pay as well as weekly and monthly breakdowns.

The response from my brother for the gymnast side of the system was that it was very simple to use as the buttons were big and visible. He found the game was a good l addition and the chat system worked, however it would be better if it did not run in the console for a future enhancement.