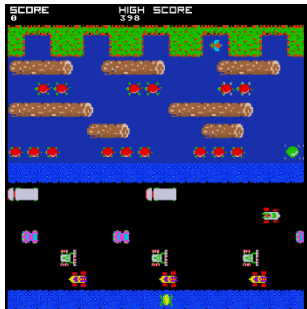


[illegible]

Development													
Testing													
Documentation													
Evaluation													

Existing Solutions



When I searched up frogger on the internet I found a game on the internet for free, it is a web-based version of frogger that can be played with a keyboard instead of the controls in an arcade machine with the joystick, I will use the keyboard controls of the frogger online because I do not have access to a joystick which can be coded into pygame. I also want to keep a similar graphics style to frogger but make it suit what the survey I did says in the results. I will make the game more elaborate than frogger as I believe that for a modern

audience the simplicity of the game will be too little and could get boring eventually. I also like the obstacles in frogger and will import the obstacle into my game. I do not like that there is not much customization in the game, and I will add skins to my game to make it more customisable.

The link is <https://froggerclassic.appspot.com/>



Another inspiration for my game is crossy road which is a more modern version of frogger which is endless and has the objective of getting the most points possible, the controls in the game are worse as you can only tap to go forwards and must swipe left right or backwards so I will be adding

keyboard WASD controls as they are better. I will be incorporating the skin system and coins which can be picked up and are randomly generated. They can be used at one hundred coins a skin to buy skins and then there are skins which can be achieved from getting new high score and completing achievements. Crossy Road also has more colourful graphics than frogger and what I am going for in my colour scheme. I will also not be making my game 3D like crossy road because of time constraints and my choice of library being pygame where it only uses an x coordinate and y coordinate meaning its 2D and my client not requiring so it will be different to Crossy Road. Crossy road is mobile only and I will not be making a mobile version to the app.

Questionnaire

Should the graphics be Ultra-Modern or more Old School in Between or Hyper Realistic.

- i. Ultra-Modern

- ii. Old School
- iii. In Between
- iv. Hyper Realistic
- v. Other

The results show that most people want a mix in in between hyper-realistic, old-school, and ultra-modern and do not want it to be only in one style. One person said it should be text based but most people said it should be in between.

Ultra Modern	2
Old School	3
In Between	8
Hyper Realistic	0
Other	1

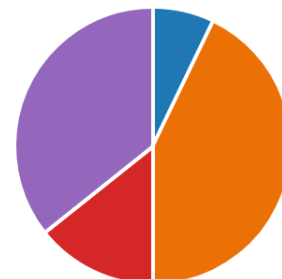


Which colour should the frog be?

- i. Red
- ii. Green
- iii. Blue
- iv. Orange
- v. Other

Majority of people said the frog should be green as this is how it is in the original game however this will only be the default skin as there will be multiple different skins that can be purchased and there can be distinctive styles. My client also wants the frog to be green as it is nostalgic for him so I will be making a green frog.

Red	1
Green	6
Blue	0
Orange	2
Other	5



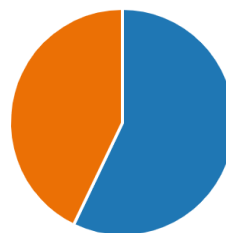
Should there be a login system.

- i. Yes
- ii. No

Most people believe that there should be a login system to save their details and high scores as well as their skins, my client Rucheer also believes this so I will include a login system into the game. A login system will mean that the game can be opened multiple times, I will have an exe file instead of the .py file and that will mean that the file will already be compiled and all you need to do is click on the exe file.

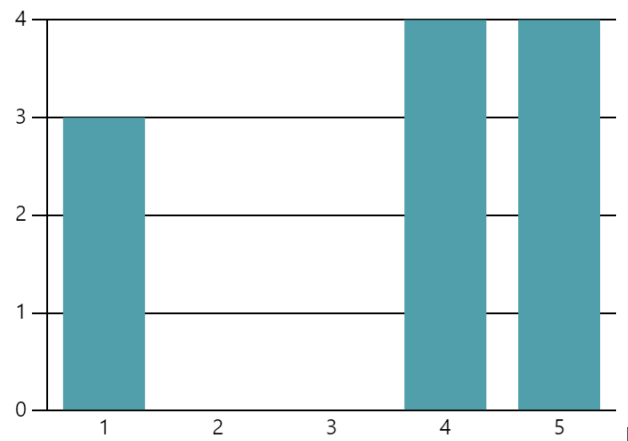
● Yes
● No

8
6



How good is this idea?

3.55
Average Rating



got an average rating of 3.55 most of the 1-star ratings are because it is not an original idea and it is a copy of frogger, so people did not like the idea of it. However more people did like it than did not like it. Frogger is a well-regarded game. My client gave the game a 5-star rating as he quite likes frogger and wants to play the game.

Any other ideas for the game?

13
Responses

Latest Responses

"i dont know what frogger is"

"inverted controls mode and a mode where the cars swap skins with the frog .."

"Super frog"

1 respondents (8%) answered **skins with the frog** for this question.

ldk cars skins is simple login system game password
no login skins with the frog enter name feature
viceSuper frog controls mode arcade game nice day
sound effects good end coins shop frogger

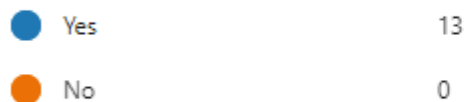
?

A lot of people gave the idea of making skins in the game which can be used to show progression, I will be incorporating this idea into my game as it seems like a good idea. One other idea was a shop which I will be adding as it links into the skins idea and the skins can be bought using in-game currency which are going to be coins. I will also be adding a login system into the game as that will mean progress can be saved in the account and you will not lose it if you close the game. There will be sound effects in the game for the different obstacles like cars and the water and the logs. I will not be adding

inverted controls and a mode where the cars swap skins with the frog as that will be too much work for something my client doesn't care about.

Should the games have skins?

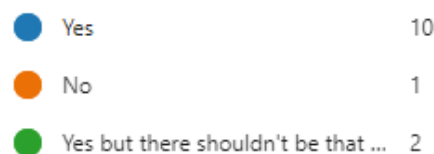
- i. Yes
- ii. No



Everybody chose to have skins into the game as it adds depth which is not in the original frogger. This makes the game more interactive and makes it more fun for a modern audience who have skins in other games that are more recent. A lot of the extra suggestions also said add skins into the game so that is quite a popular idea, and my client also wants this so I will be adding this.

Should there be difficulties e.g. speed of cars?

- i. Yes
- ii. No
- iii. Yes, but there shouldn't be that big of a jump between levels



Most people said that there should be difficulty levels and the way I will implement this into the game is that I will make it so that the number of points per tile moved forward is

proportional to the difficulty level e.g. you will get 5 points in the hardest level but only 1 in the easiest level. I will also be making it so that there are different high scores for each mode and difficulty and there will also be a main one for all the scores together and these will all be stored on a text file.

Coursework Interview

Question: Have you ever played Frogger or Crossy Road?	Answer: Yes, I used to play Frogger a lot in my youth when it first came out and was very good at it and I downloaded forger on my phone recently when one of my friends who knew that I loved Frogger told me to download it as it is similar to Frogger.
Question: Which parts of Frogger do you prefer and which parts of Crossy Road do you prefer?	Answer: I like the simplicity of frogger compared to modern games as the objective is very easy to understand and I can pick it up anytime I want to play whenever nut I like the skins in Crossy Road and I like the challenge of working towards them because it gives me a larger objective as sometimes Frogger can be a bit too simplistic and there isn't any progression once you have finished the game as it is finite
Question: Do you want skins in the game, Why/Why not?	Answer: Yes, I want skins in the game as they can give the player a sense of achievement and make it easier for me to play longer hours on the game and makes it easier and more enjoyable to grind.
Question: Do you want a login system?	Answer: Yes, I would like a login system so that each session that I play the game, I can save my high score and the skins I have, it will allow me to play for longer as losing progress each time the app is closed would be quite discouraging.
Question: Do you prefer the retro colour theme or the modern one	Answer: I would prefer the retro colour scheme with a mix of the modern one as I remember the retro style from when I was younger, and I have good memories of the old-style nut I also like the new bright colours in Crossy Road.
Question: What colour frog do you like?	Answer: I like the green colour frog as that is how it was in the original Frogger game and that is how I like the frog, but I also want skins so that there are multiple different characters that are unlockable so it will have different colour skins.
Question: What Hardware Specifications do you have	Answer: I have an old Intel core i5 5 th generation and a GTX 1030 form Nvidia with 8GB Ram and a 256 GB HDD.

Requirements

- The player should be able to use the keyboard to move around in the game since that is the most common form of input especially on PC/ Laptop
- The game should be opened through an exe file since that will stop the game from being cheated by modifying the source code.
- There should be a login window and system that saves the login details in a .db file for later use.
- The high scores should be saved in the same .db file as the username and password and each user should have their own high score since that is what my stakeholders said in the survey.
- There should be a menu that can be interacted with and buttons that can be clicked to go to different parts of the game since that will make the game more useable and easier to play.
- The game should have a mix of retro and modern theme since that is what my stakeholders said in the survey that is shown above.
- There should be cars as obstacle that will kill the player if hit since my game is inspired by crossy road which has that feature.
- There should be obstacles in the game like stones and rivers that the player can't get past normally.
- A GUI in the game to make it easier to understand and use the application.
- Multiple Levels to make the game more interactive.
- Each Level giving a different number of points since the difficulty is harder more points are given.

Success Criteria

Below are the success criteria of what I need in the game to make it feasible and playable as well as some limitations and the minimum specification requirements to play the game.

- I will use the checker function in pygame to move when the WASD buttons are clicked.
- I will use PyInstaller to compile the game into an exe file that can run.
- I will use PyQt to create a GUI that can interact with the user and allow a login page.
- I will create a .db file that will use SQL to save high scores which can be retrieved.
- I will use PyQt to create a menu window that has start game, exit, skins and settings in it.
- I will incorporate graphics from the original frogger game as well as create some newer art.
- I will use PyGame to add obstacles which will have a car skin and cause the game to be over if the player touches them.

- I will code in representations of obstacles and make the game not allow the player to go through.
- I will include a menu using PyQt that will have clearly labelled to help the player navigate.
- I will add a button in the menu that can help change the speed of the character and the camera to make it harder for the character to get forward.
- I will assign a variable called difficult and a 2D list that has columns for levels, speed of player and speed of camera.

Limitations

- Can't make the game multiplayer with 2 players controlling the characters since it will only be on one computer with one keyboard and mouse and most keyboards can only take in a few inputs at a time.
- Making the game 3D as PyGame doesn't support 3D within the low minimum requirements I gave.
- Having the game be accessible on any computer in the world since I don't want to create a server and do all the network configurations for port forwarding and setting up the server.
- Having the game run with great AAA graphics since i don't have the resources to create a AAA game.
- Can't run on Mac OS since I don't have access to a Mac.

Programming Language and Hardware

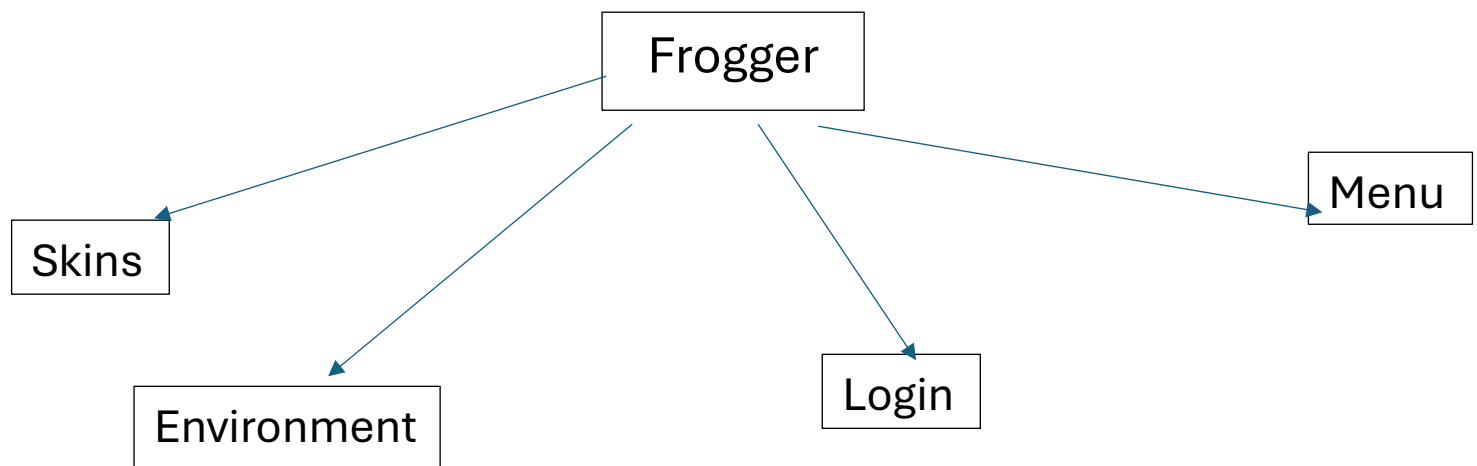
I will be using Python for this game and will be using the PyGame library for the graphical aspect, I could have used Ursica which is a 3D library for app development however, I would have to learn the library which I don't have time to do, I could have used C++ because it has libraries that are more efficient than python and easier to use for a variety of use cases but I don't know the language and don't have the time to learn it, as is with Java. I would have used JavaScript and made it a website however the frameworks are lacking for making embedded games in JavaScript and I would have had to host the website and buy the domain name which I don't have the funds to do.

My Hardware requirements would be:

- 256MB of RAM
- 20MB of storage space
- 100MHz clock speed
- Any Windows OS

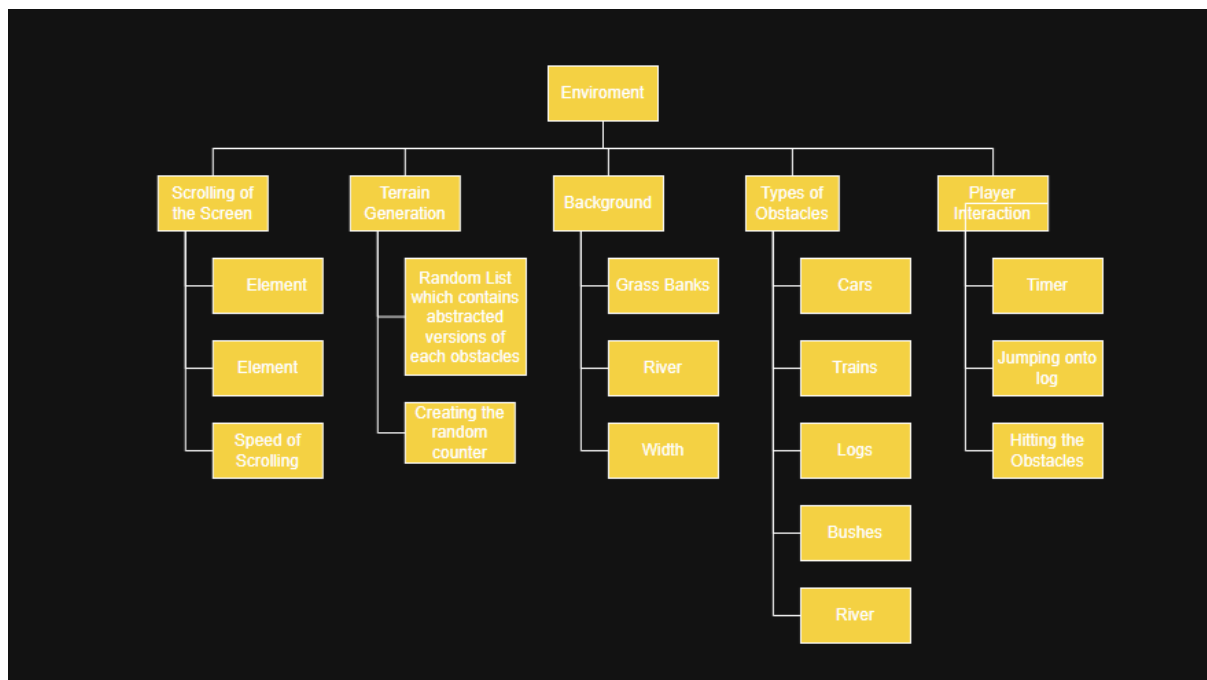
Design

In the design section I will be decomposing the problem and stating how it will be constructed. This will make it easier to understand how to make the program in an easier way, it will also make it easier for any reader to understand the program better. I will be breaking up the film into 5 parts, the shop, the environment, the login page, the frog character and the menus. I will be using a top-down methodology.



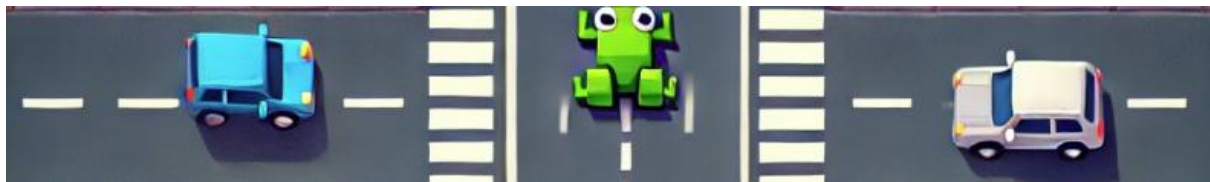
Environment

The environment of the game includes the background and the obstacles in the games, I have decided to decompose it into different types of obstacles and the elements of the background, since my game includes parts of frogger and crossy road, I will be using cars, logs, a river and bushes in my game, I will also be including a list to map to each object.



I will be using pseudo code to show the type of code required for this.

For the scrolling of the screen, I would use a while loop for while game is not over, keep scrolling at a constant rate which will be a variable that will change depending on the level. All the terrain will be preset and randomly generated out of some presets, and this will add randomness to the game, all previous ones would be deleted.



This is an example of how the roads in my game could look since the game is meant to be a mix of crossy road and frogger, I would need to determine the width of the grid which would be optimized to the correct width, each group of pixels corresponds to some colours which can be used.

River = ["W", "W", "L", "W", "W", "W", "L", "W", "L", "W", "W", "W", "W", "W", "L", "W", "W"]

Road = ["R", "R", "R", "R", "R", "C", "R", "R", "R", "T", "T", "R", "R", "R", "R", "R", "R", "R"]

Land = ["B", "G", "G", "B", "G", "G", "G", "G", "G", "B", "G", "G", "B", "G", "G", "G", "G", "B", "B"]

This is used to randomise which row is next to be crossed by the character and makes the game more exciting, All the strips are the same width and some of the elements move like the logs, cars and trucks, there would be one for the player that would move through each of the strips

I would also have a part of the program that is used to check the player collisions, this would check if the hitbox of player is on or touching any of the ones that can finish the game like Cars.

function checkCollision(playerX, playerY, playerWidth, playerHeight, vehicleX, vehicleY, vehicleWidth, vehicleHeight):

```
if (playerX < vehicleX + vehicleWidth) and (playerX + playerWidth > vehicleX) and (playerY < vehicleY + vehicleHeight) and (playerY + playerHeight > vehicleY):
```

```
    return true.
```

```
while (!gameOver):
```

```
    input playerInput()
```

```
    if checkCollision(playerX, playerY, player_width, player_height, vehicle.x, vehicle.y, vehicle.width, vehicle.height):
```

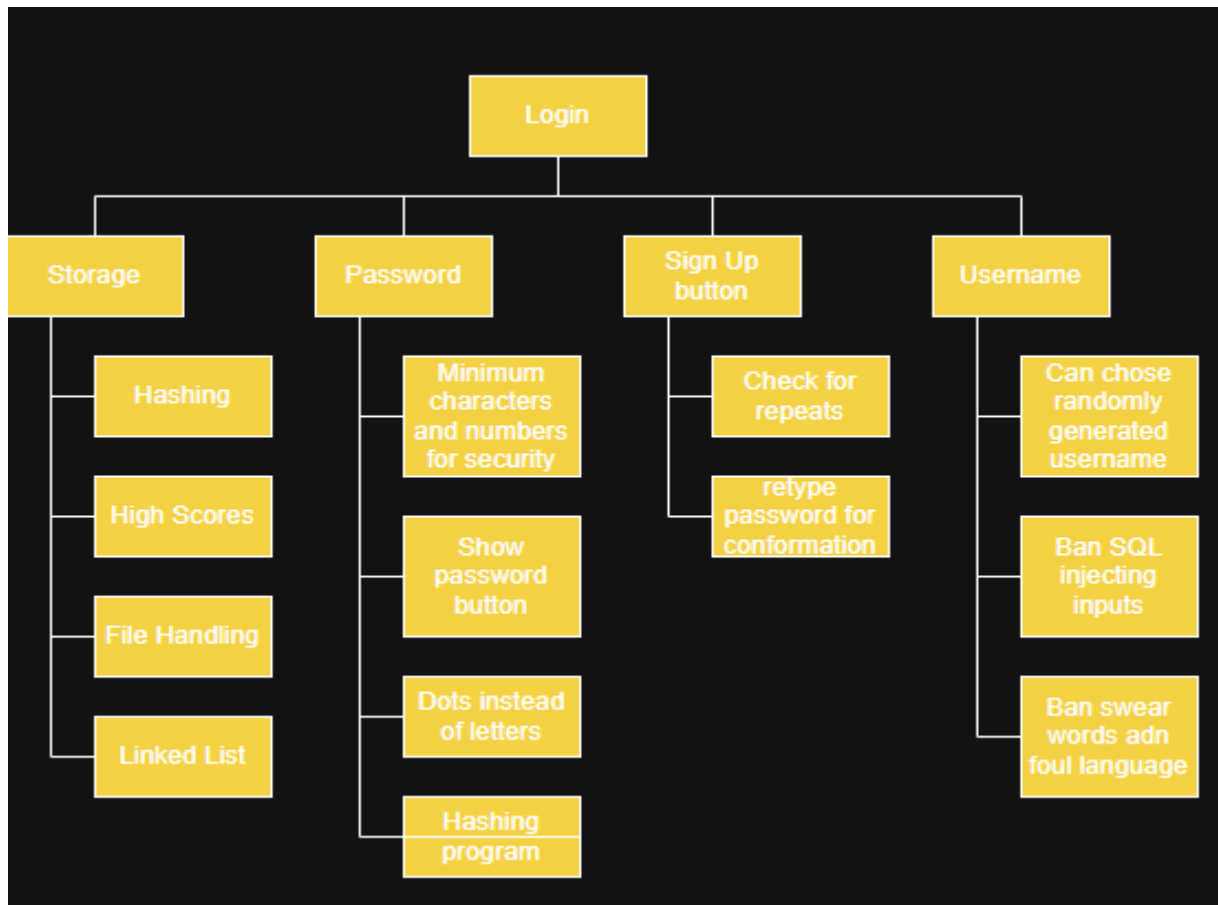
```
        ()gameOver = true
```

This is an example of the code that could be used to check for any collisions, it checks if the position of the player is the same as any of the obstacles and the game gets finished if it is.



These are some ideas for the skins in the game.

Login



For the login screen I will be implementing a sign-up button, hashing and a login screen.

The image shows a mockup of a login screen. It features a white card with a light blue background. The card has a title 'Login' in blue. Below the title are two input fields: 'Username' and 'Password'. At the bottom of the card are two buttons: a blue 'Login' button and a green 'Sign Up' button.

The image shows a 'Sign Up' form on a dark background. The form is centered and contains the following elements:

- Sign Up**: A title in a light blue font.
- Username**: A label in light blue font above a white input field.
- Password**: A label in light blue font above a white input field.
- Confirm Password**: A label in light blue font above a white input field.
- Sign Up**: A light blue button with white text.
- Login**: A light blue button with white text.

My hashing program will be simple in nature but will provide adequate security for the app.

Create empty list 'ulist'.

For each character 'i' in 'username':

Add 'i' to 'ulist'.

For each character 'i' in 'ulist':

Get ASCII value of 'i' Double the ASCII value.

Calculate the modulus (value % 65) and store it.

Create empty list 'plist'.

For each character 'i' in 'password':

Add 'i' to 'plist'.

Prompt the user to enter 'username' and 'password'.

Create empty list 'usernameList'.

For each character 'i' in the input 'username':

Add 'i' to 'usernameList'.

For each character 'i' in 'usernameList':

Get ASCII value of 'i'.

Double the ASCII value.

Calculate the modulus (value % 65) and store it.

If stored username hash equals entered username hash AND stored password hash equals entered password hash:

Print 'Welcome' + username Else:

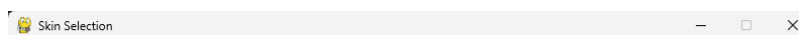
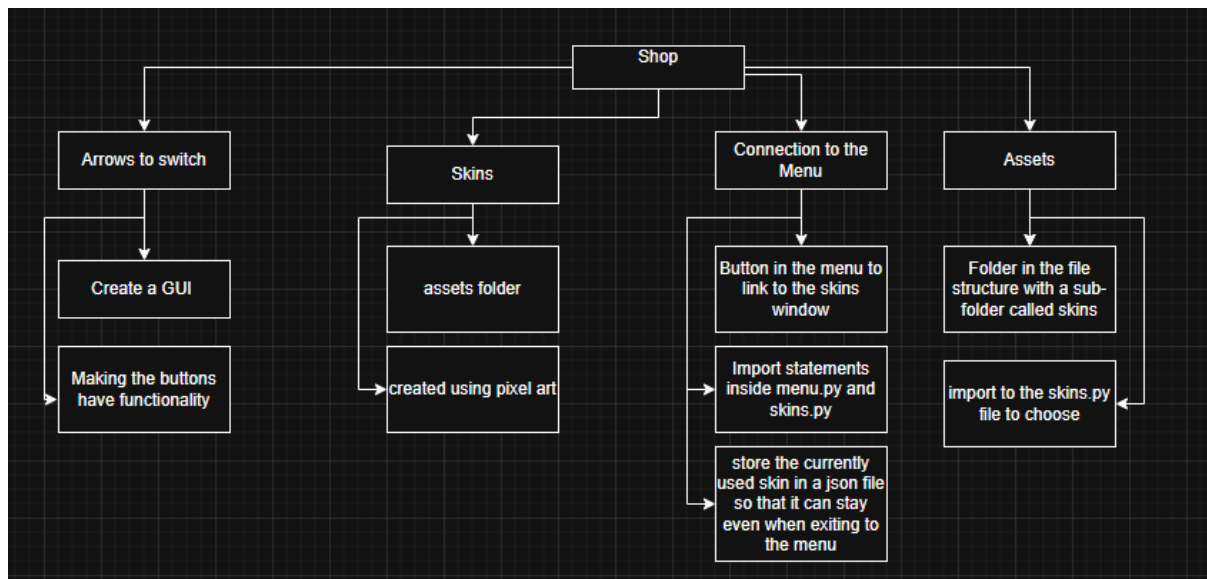
Print 'Access Denied, Wrong Username or Password'

I will also store this in a text file with linked lists.

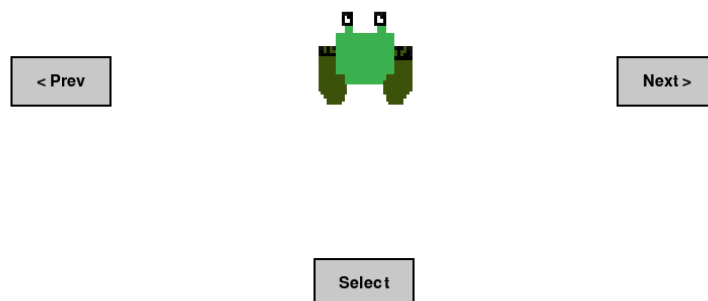
I will also store high scores associated with each account and then each account high scores ranked in order.



Skins



Frog



skins system

open skins menu

load user's current coin balance and unlocked skins.

display all skins:

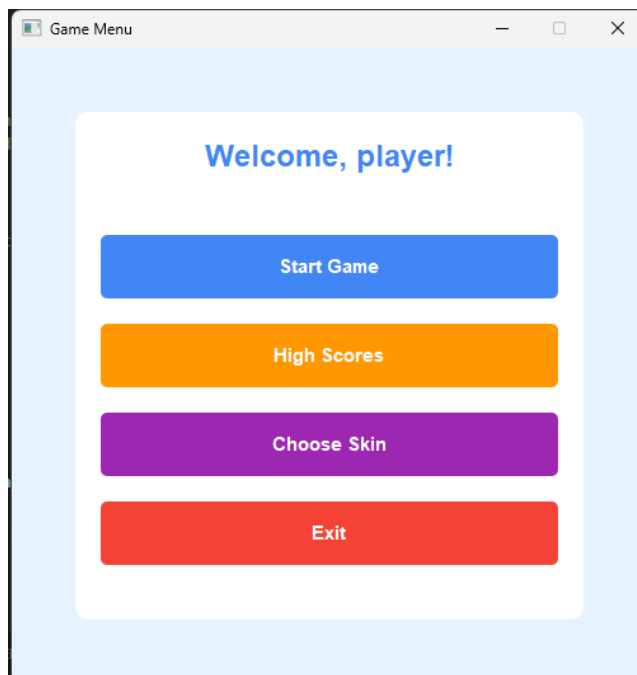
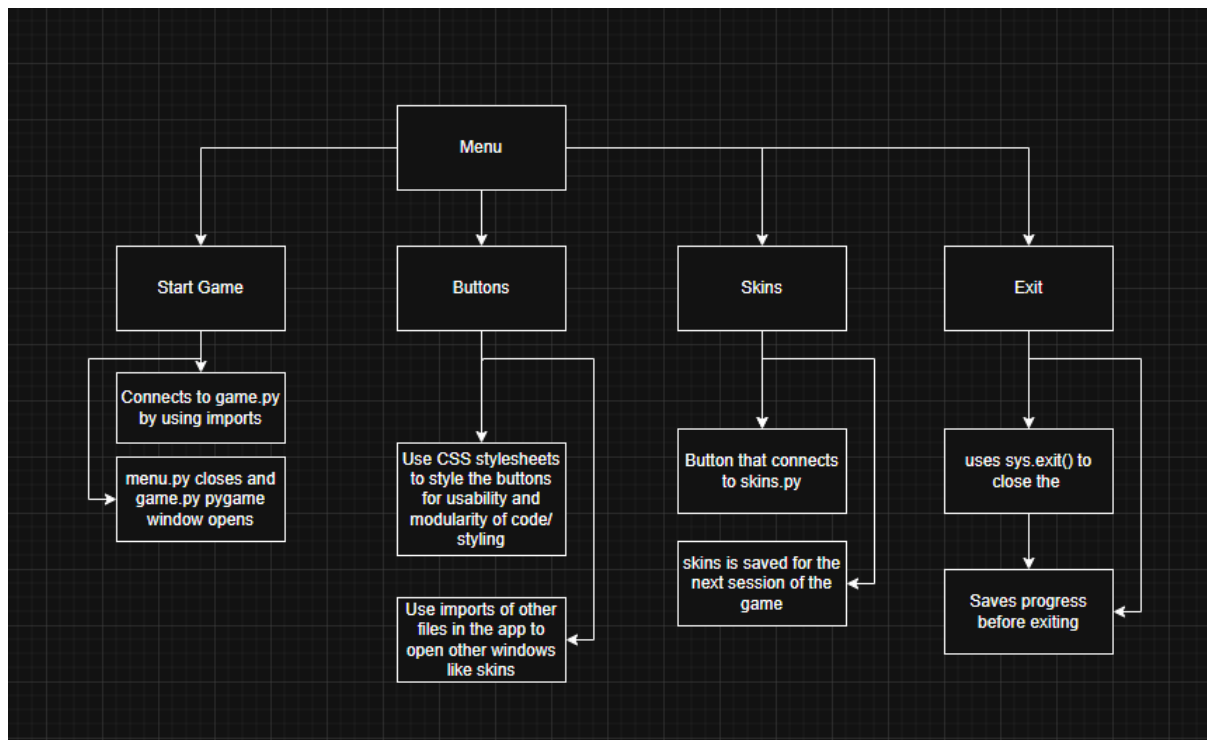
#Show unlocked skins as selectable

#Show locked skins as unavailable

wait for user to select a skin.

```
if selected skin is already unlocked then
    apply skin to player's profile.
    show confirmation message: "skin selected".
else if selected skin is locked then.
    if user has enough coins, then.
        subtract coins from user balance.
        unlock selected skin.
        apply skin to player's profile.
        save changes to database.
        show message: "skin unlocked and applied".
    else
        show error message: "not enough coins".
    end if
end if
return to main menu
```

Menu



main menu system

display menu window with four options:

- start game
- skins
- settings
- exit

wait for user to click a button.

if user clicks "start game" then

load saved user data (e.g., skin, high score)

launch the game window.

end if

if user clicks "skins" then

open the skins menu.

display available and locked skins.

if user selects a skin and it is unlocked then

save selected skin for future use.

else if user selects a locked skin, then.

show message: "not enough coins".

end if

end if

if user clicks "settings" then

open settings menu

allow user to change:

- difficulty level
- sound preferences

save settings.

end if

if user clicks "exit" then

save progress.

close the application.

end if

Variables

Variable Name	Variable Type	Function
Width	Integer	Width of the screen
Height	integer	Height of the screen
Screen	Pygame Surface Object	Is the surface on which the rest of the objects can be placed upon
Green	tuple	Colour stored in a tuple
Blue	tuple	Colour stored in a tuple
Grey	tuple	Colour stored in a tuple
Yellow	tuple	Colour stored in a tuple
camera_speed_y	integer	Speed of the camera
camera_y	Integer	Position of the camera
obstacle_types		
skin_path	string	Path of the skin relative to the main game
skin_name	string	Name of the skin
terrain_list	List	Where the next obstacles are stored
add_new_rows	Function	Adds the rows that are from the txt file
app		
load_level_data	Function	Loads the data that is stored in the txt file
add_initial_rows	Function	Adds in the initial rows for when the game starts
level_data	list	Stores the data from the text file about obstacles
handle_events	function	Contains a while loop to see if the x in the top right has been pressed and closes the window
Keys	Boolean	Checks if a key has been pressed for event handling
Left	integer	x-coordinate on the left-hand side of the window
Right	Integer	x-coordinate on the right-hand side of the window
Top	integer	y-coordinate at the top of the screen
Bottom	integer	Y-coordinate at the bottom of the screen
dash_length	integer	Length of each movement based on the direction of the key
log_image	surface	Loads the log image to render
clock	object	Controls the clock function that dictates the fps

Test No.	Test	How to test it	Expected Outcome
1	Only the back cause game over	Move the sprite to all four edges and see if any of the other edges except the back cause the game to end	Only touching the back causes the game to end
2	Choosing Skins	Go to the skin's sections and check if the player is able to choose any of the skins and hit select	The player can choose any available skin he wants
3	Logging In	Try login to the game from the login window and see if the player can successfully login	The player logs in and their username is displayed and used for any relevant messages
4	The hashing of passwords in the .db file	Open the main.db file and check if the passwords are in plaintext or hashed	The passwords are hashed, and you cannot see the actual password
5	The player dies when they hit the back of the screen	Let the camera move and keep the sprite still and see if the game ends when the sprite hits the back	Hitting the back cause the game to end
6	The sprite is visible with the correct selected skin	Go to skins, select a skin and start a game to see if that skin is visible	The player can see the correct chosen skin when they start the game
7	The sprite stops when it hits an obstacle	Move into an obstacle with the sprite and see if it stops	The sprite stops when it is about to move into an obstacle
8	The game ends when it hits a truck	Move onto the road and wait for a truck to move into the player and see if the game ends	The game ends when the sprite is hit with a truck
9	Player moves onto water without log	Move the sprite onto the water where there is not a log	The player dies and the game ends when they step on the water
10	Points increase with moving forward	Move forwards and have a counter in the top right for number of points	The number of points go up when the player moves forward
11	Invalid login attempt	Type in login details that you know are false	Gives a message saying the details are wrong
12	Save the score to the database	At the end of the round get a new high score and see if it gets saved to main.db	The new high score gets saved to main.db

13	Press multiple keys at the same time	Press multiple keys at the same time when playing the game	If the keys are opposing then they get cancelled out, otherwise the sprite moves diagonally in both their directions
14	Try an SQL injection attack	Type in an SQL injection command into the login window	The game gives an error saying that those keywords are not allowed
15	The game goes to menu after death	Die in the game and see if it goes to the menu	The game goes to the menu when the player dies
16	Data saved on exit	Exit the game after playing a round	The data is saved in the .db file and can be retrieved later
17	The game in an exe file	Double click the exe file	The game opens onto the login page

I wrote my code with pygame to draw a background, it uses the variable screen to act a surface that other sprites and objects can be laid over, this means that I can overlay my background or my obstacles over the screen making it easier since I don't need to draw everything separately, this, I also set the frame rate to 60 fps however it is better to set it using a variable so you just need to change the variable rather than going into the line of code. There is also an event handler that sets the default to true while the pygame window is running, it is always checking to see if the x in the top right corner is clicked, and it sets running to true and closes the window. The pygame.display.flip() refreshes the window at the required rate to make it work.

```
import pygame
fps = 60
#import the library whichc is specified earlier
#initialize pygame to open the window
pygame.init()
screen = pygame.display.set_mode((1280, 720))
#set the size for the screen
clock = pygame.time.Clock()
#this is to control the frame rate
running = True
#set the var running true to start a game loop
while running:
    # poll for events
    # pygame.QUIT event triggers when the x is clicked in the top right
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill the screen with purple
    screen.fill("purple")

    # flip() the display to refresh
    pygame.display.flip()

    clock.tick(fps) # limits FPS to 60
pygame.quit()
```



I first created a blank background with the colour purple just to set the surface on which each of the elements will be based, this gives a rudimentary view of how the background is and provides an abstracted view of how the game appears when all details have been stripped, this also uses decomposition as it breaks down the problem of making the game into making the smaller steps that are shown, one of these steps is the creating of the surface, this will be the bottom layer with more detail being added like the sprite of the frog being added on top, this makes it

easier for me to start the program and makes it more manageable since the vision for the programs development is more robust and clear and I can have it ready months in advance rather than doing it as I go along, the use of the waterfall cycle mixed with the agile methodology means the development lifecycle of the program is easier to keep a track of and reaches the

```
pygame 2.5.2 (SDL 2.28.3, Python 3.11.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
Traceback (most recent call last):
  File "C:\Users\skrac\Documents\Python\Frogger Coursework\Test\test.py", line 21, in <module>
    pygame.draw.circle(screen, "green", player_pos=100)
  TypeError: function missing required argument 'center' (pos 3)
```

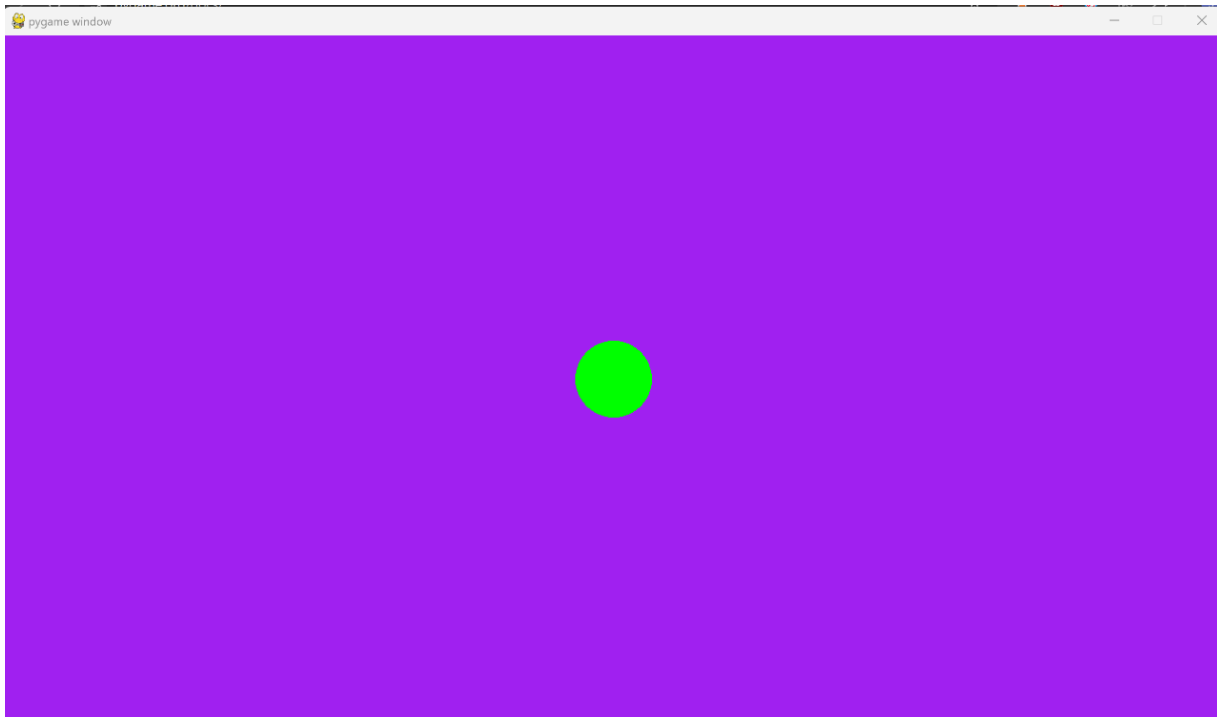
satisfaction of the customer, I incorporated agile by having a weekly stand-up meeting with my client Rucheer, this ensures that I have his feedback on any iterative changes made to my code and program as well as improving the UI to make it better for him to use. This balances out the downsides of the water fall cycle where all of one part is done at once which gives less weight to customer satisfaction and is not perfect on its own.

```
i C:\Users\skrac\Documents\Python\Frogger Coursework\Test\test.py
fps = 60
#import the library whichc is specified earlier
#initialize pygame to open the window
pygame.init()
screen = pygame.display.set_mode((1280, 720))
#set the size for the screen
clock = pygame.time.Clock()
#this is to control the frame rate
running = True
#set the var running true to start a game loop
while running:
    # poll for events
    # pygame.QUIT event triggers when the x is clicked in the top right
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill the screen with purple
    screen.fill("purple")
    pygame.draw.circle(screen, "green", player_pos=100)
    # flip() the display to refresh
    pygame.display.flip()

    clock.tick(fps) # limits FPS to 60
pygame.quit()
```

The above code sets the fps to 60, it then initializes the pygame library and then sets the screen size, it then assigns the clock function to the variable clock, and then sets the Boolean running to true and uses a while loop to check if the x in the top right has been clicked, and fills the screen as purple and draws a circle in the specified locations, this is meant to abstract as the sprite since it is easier to represent the sprite as a circle in early stages of development than to draw out the whole frog.



I drew the green circle using the `pygame.draw.circle` function, this is to abstract away the difficulty of adding in the picture of the frog and just render in the sprite representation of the frog. Next, I would also add in a movement feature of the using the `key.get_pressed()` function. Since the circle is a representation of the frog, I will also be adding in a skin for the frog in place and add a hitbox using the circle and the collide checker function.

```
import pygame

def initialize():
    pygame.init()
    screen = pygame.display.set_mode((1280, 720))
    player_pos = pygame.Vector2(screen.get_width() / 2, screen.get_height() / 2)
    clock = pygame.time.Clock()
    return screen, player_pos, clock
#this uses a while loop ot check whether the x is clicked to close the window
def handle_events():
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            return False
    return True

def update():
    pass # Add game state updates here if needed
#function that renders the looks of the window
def render(screen, player_pos):
    screen.fill("purple")
    pygame.draw.circle(screen, "green", player_pos, 40)
    pygame.display.flip()
#main function that runs the game
def main():
    screen, player_pos, clock = initialize()
    fps = 60
    running = True

    while running:
        running = handle_events()
        update()
        render(screen, player_pos)
        clock.tick(fps)

    pygame.quit()
#runs when the script is run and not when it is imported as a module
if __name__ == "__main__":
    main()
```

I also used functions and put everything in functions, this makes the program more modular, and this means I can see any problems by isolating the problematic functions and using print functions to check for errors, this also means I can more easily use error handling if one part start I can use try and except and a different print statement for each module to see which module isn't working.

I also used object-oriented programming since this makes it easier to call functions and the inheritance feature is very useful when i want a function to have some parameters that previous function had.

```
import random
water = []
for i in range(10):
    water.append(['w'])
print(water)
```

I am going to generate the environment in advance and cache it so that it can be loaded into the game just by using lists as each of the next obstacles, above I created a list of just water to test out the generation of the obstacles by just generating water.

```
randoms = []
translate = {
    'w': 'water',
    'r': 'rock',
    'g': 'grass',
    'l': 'log',
}
for j in range(10):
    randoms.append(random.choice(['w', 'r', 'g', 'l']))
```

I defined a dictionary that show what each of the characters correspond to, and then created a program to randomly choose between each of the four options an generate a list with them.

```
[[['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w']]
 ['g', 'r', 'g', 'r', 'l', 'w', 'g', 'r', 'r', 'l']]
```

The screenshot above shows the output that I got from running the program and how it is randomly generated.

```
[[['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w']]
Traceback (most recent call last):
  File "C:\Users\skrac\Documents\A-Level\Computer Science\Coursework\Dev\generator.py", line 16, in <module>
    print(translate(randoms))
          ~~~~~^~~~~~
TypeError: 'dict' object is not callable
```

When I tried printing the list randoms with the dictionary converting to the actual names, I tried doing this in the print function itself, this gave me a syntax error saying the dictionary isn't callable that way, to remedy this i just made it print the list randoms since it isn't required anywhere in my specification that the list is needed to print the actual names especially when this is just to generate the obstacles and won't be seen by the user.

```

for j in range(10):
    randoms.append(random.choice(['w', 'r', 'g', 'l']))
print(randoms)
rangen = open('random_generations.txt', 'w')
rangen.write('/n', str(randoms))
rangen.close()
# print(translate[randoms])

```

I created a separate file to which the randomly generated lists all get printed one after the other to make it easier to cache and refer to when the game is being rendered, this can reduce the CPU load that is required when the game is being played and this can help keep the program within the required minimum specs but it will be hard since pygame is not the most efficient

```

[['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w'], ['w']]
['w', 'w', 'w', 'g', 'w', 'r', 'r', 'w', 'l', 'r']
Traceback (most recent call last):
  File "C:\Users\skrac\Documents\A-Level\Computer Science\Coursework\Dev\generator.py", line 18, in <module>
    rangen.write('/n', str(randoms))
TypeError: TextIOWrapper.write() takes exactly one argument (2 given)

```

Above is the error I got because I tried to print the str of the variable called randoms above and /n which is used to add a new line between everything printed into the text file, however, the comma that was used to space out the /n and randoms, this meant that it was taken as a second argument / parameter inside the function which isn't possible as correct syntax, the way I corrected this was by inputting two lines which would input on after the other with the first being the variable randoms and the second being the /n which meant that there was a separator to add a new line each time one of the randomly generated strings was added to the text file for caching.

When I ran the program like this, the rangen.write was on the wrong loop which meant that the write function was repeating the same line all two hundred times, this means the same obstacle would be generated every time and would defeat the purpose of the game.

I also modified the program to use 'a' for append in the open function so that when the text file opens, it will be in append mode so that when data is printed, the data is added on top of what was already there so that there is no data loss.

I also modified the program to use 'a' for append in the open function so that when the text file opens, it will be in append mode so that when data is printed, the data is added on top of what was already there so that there is no data loss.

```

camerapn.py  generator.py M  random_generations.txt U x
Dev > random_generations.txt > [g, r, w, r, w, r, w, w, w, l]
175 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
176 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
177 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
178 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
179 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
180 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
181 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
182 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
183 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
184 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
185 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
186 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
187 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
188 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
189 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
190 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
191 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
192 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
193 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
194 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
195 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
196 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
197 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
198 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
199 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
200 ['g', 'r', 'w', 'r', 'w', 'l', 'w', 'w', 'w', 'l']
201

```

```

Dev > generator.py > ...
You, 29 seconds ago | 1 author (You)
1  import pygame
2  import random
3  water = []
4  for i in range(10):
5      water.append(['w'])
6  print(water)
7  randoms = []
8  translate = {
9      'w': 'water',
10     'r': 'rock',
11     'g': 'grass',
12     'l': 'log',
13 }
14 for j in range(10):
15     randoms.append(random.choice(['w', 'r', 'g', 'l']))
16 print(randoms)
17 rangen = open('random_generations.txt', 'a')
18 for z in range(200):
19     rangen.write(str(randoms))
20     rangen.write('\n')
21 rangen.close()
22 # print(translate[randoms])
23
24

```

I also modified the program to use 'a' for append in the open function so that when the text file opens, it will be in append mode so that when data is printed, the data is added on top of what was already there so that there is no data loss.

changed the program by modifying the loop to make it more nested so that every time the iteration happens, it will generate a new random list that can be printed into the text file so that they are actually all different, below are the results before, i tried debugging the program by limiting the range function with the variable i to having 2 characters, this helped me isolate that the problem wasn't coming from the top range function and that helped me look lower to find the actual problem with the nesting of the loops. Another problem i encountered was the strings being appended infinite times and iterating without stop despite the limit being in the range function unless i stopped the program manually. The problem was caused.

```
Dev > random_generations.txt > [r]
1972 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1973 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1974 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1975 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1976 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1977 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1978 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1979 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1980 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1981 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1982 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1983 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1984 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1985 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1986 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1987 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1988 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1989 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1990 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1991 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1992 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1993 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1994 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1995 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1996 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1997 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1998 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
1999 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
2000 ['r', 'g', 'w', 'r', 'g', 'l', 'w', 'g', 'r', 'r']
2001
```

275 ['n', 'g']
276 ['n', 'g']
277 ['n', 'g']
278 ['n', 'g']
279 ['n', 'g']
280 ['n', 'g']
281 ['n', 'g']
282 ['n', 'g']
283 ['n', 'g']
284 ['n', 'g']
285 ['n', 'g']
286 ['n', 'g']
287 ['n', 'g']
288 ['n', 'g']
289 ['n', 'g']
290 ['n', 'g']
291 ['n', 'g']
292 ['n', 'g']
293 ['n', 'g']
294 ['n', 'g']
295 ['n', 'g']
296 ['n', 'g']
297 ['n', 'g']
298 ['n', 'g']
299 ['n', 'g']
300 ['n', 'g']
301 ['n', 'g']
302 ['n', 'g']
303 ['n', 'g']
304 ['n', 'g']
305 ['n', 'g']
306 ['n', 'g']
307 ['n', 'g']

183	['g', 'g', 'w', 'r', 'l', 'l', 'w', 'g', 'r', 'w']
184	['g', 'l', 'g', 'l', 'l', 'w', 'w', 'r', 'l']
185	['l', 'g', 'w', 'g', 'l', 'w', 'l', 'r', 'r', 'r']
186	['l', 'r', 'l', 'l', 'r', 'l', 'g', 'w', 'r', 'w']
187	['l', 'l', 'r', 'g', 'r', 'g', 'g', 'w', 'w', 'w']
188	['w', 'r', 'w', 'l', 'l', 'l', 'g', 'g', 'l', 'l']
189	['w', 'g', 'l', 'g', 'w', 'w', 'l', 'g', 'r', 'g']
190	['g', 'g', 'r', 'l', 'w', 'l', 'g', 'r', 'g', 'l', 'g']
191	['r', 'l', 'g', 'r', 'g', 'w', 'r', 'g', 'r', 'w']
192	['l', 'r', 'l', 'w', 'g', 'g', 'r', 'l', 'w', 'l']
193	['g', 'r', 'l', 'g', 'w', 'r', 'r', 'l', 'g', 'w']
194	['l', 'w', 'w', 'g', 'r', 'l', 'r', 'l', 'g', 'r']
195	['r', 'r', 'r', 'l', 'g', 'w', 'r', 'w', 'r', 'l']
196	['g', 'w', 'r', 'w', 'w', 'w', 'g', 'l', 'w', 'w']
197	['w', 'r', 'r', 'w', 'r', 'r', 'r', 'w', 'l', 'g']
198	['w', 'r', 'r', 'w', 'w', 'g', 'g', 'l', 'r', 'r']
199	['w', 'g', 'g', 'g', 'r', 'w', 'l', 'w', 'g', 'r']
200	['l', 'r', 'r', 'r', 'l', 'w', 'g', 'r', 'r', 'r']
201	

```

Dev > generator.py > ...
You, 54 seconds ago | 1 author (You)
1 import pygame
2 rangen = open('random_generations.txt', 'a')
3 import random
4 water = []
5 for i in range(10):
6     water.append(['w'])
7 print(water)
8
9 translate = {
10     'w': 'water',
11     'r': 'rock',
12     'g': 'grass',
13     'l': 'log',
14 }
15
16
17
18 for z in range (200):
19     randomness = []
20     for j in range(10):
21         randomness.append(random.choice(['w', 'r', 'g', 'l']))
22     rangen.write(str(randomness))
23     rangen.write('\n')
24 print(randomness)
25 rangen.close()
26
27

```

This is the 3rd iteration of the program that automates the process of creating the obstacles, my next improvement for the program will be to give weights to each of the possible characters in w, l, g and r and make them randomly generate but within those weights, this improves the game since

```

You, 1 second ago | 1 author (You)
1 import pygame
2 rangen = open('random_generations.txt', 'a')
3 import random
4 water = []
5 for i in range(10):
6     water.append(['w'])
7 print(water)
8 randomness = []
9 translate = {
10     'w': 'water',
11     'r': 'rock',
12     'g': 'grass',
13     'l': 'log',
14 }
15 for j in range(10):
16     randomness.append(random.choice(['w', 'r', 'g', 'l']))
17
18
19     for z in range (200):
20         rangen.write(str(randomness))
21         rangen.write('\n')
22     print(randomness)
23 rangen.close()
24
25
26

```



```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QLineEdit, QVBoxLayout
#all the imports from the library
app = QApplication(sys.argv)

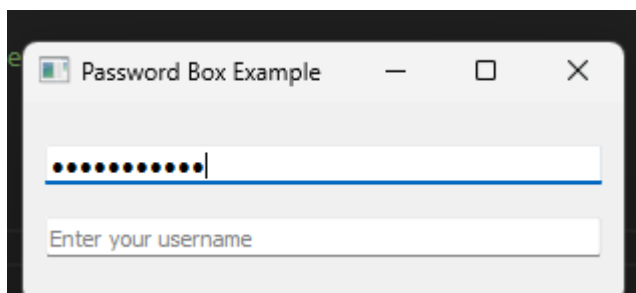
window = QWidget()
#set the window like pygame.init()
window.setWindowTitle('Password Box Example')
#sets the title of the window
window.resize(300, 100)
#sets the size of the window
layout = QVBoxLayout()

password_box = QLineEdit()
password_box.setPlaceholderText('Enter your password')
#creates a text box to type into
#makes the text look hidden so that the user can't see what they are typing
password_box.setEchoMode(QLineEdit.Password)

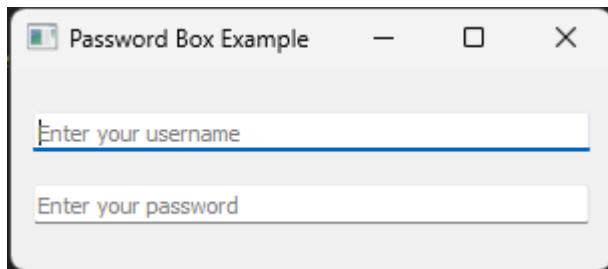
layout.addWidget(password_box)

window.setLayout(layout)
window.show()
#outputs the window like pygame.display.update()
sys.exit(app.exec_())
#exits the program like pygame.quit()
#no point of using oop since most of the code is in the library
```

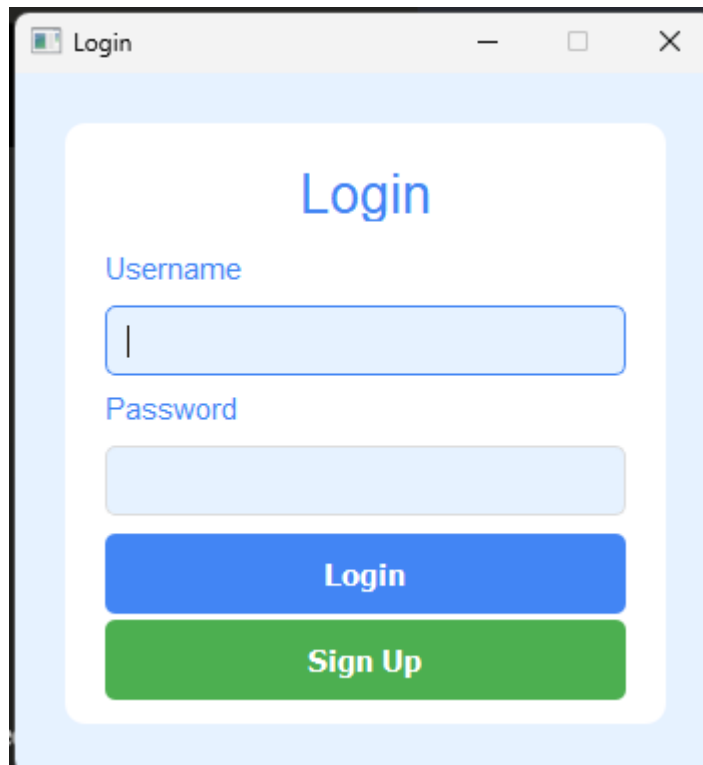
For the next part of my program I mocked up a window using PyQt5 to create a login window which is a part of my specification, I chose to use PyQt quite late in the project since I was initially going to use Pygame however saw how unnecessarily complicated that would be since PyGame doesn't have native support for any of the required features, I was going to use Kinter but saw PyQt when researching and thought it a better solution, the library has a lot of in built functions that are useful like the setEchoMode function that will obscure the inputted text to make it more secure, along with the window, I will be using rudimentary hashing along with databases to secure my game and implement a login system. I chose not to use OOP here since most of the code is in the library meaning there isn't much reason for OOP especially when each of the function isn't going to be repeated in the program multiple times, I will be using a function to run the program from this file, so this file only needs to be on its own.



I added the username part below, but I put it in the wrong order so that the password was above, this is the rudimentary version with just the input boxes and has no styling.



This is the fixed version with them in the right order, there are no event buttons like sign in, but the password does show the circles rather than the actual characters being inputted.



This is the finished version of the login window, I made it identical to the mock-up I showed in the design section of the document, I used css stylesheets instead of inline styling to increase modularity and separation, the use of import in python means that I can keep multiple different files e.g., one for the database, one for the SQL querying, one for the actual game and one for the menu, this means if I make a change to one of the files, then the change only happens in the specific place where it is required, it stops things like variable names being used multiple times and means that not everything has to be coded in the same paradigm since even though the game might be better with OOP, the login page doesn't require anything like that.

```
self.password_input.setFixedHeight(35) # Fixed height for inputs
self.password_input.setStyleSheet("""
    QLineEdit {
        border: 1px solid #ddd;
        border-radius: 5px;
        padding: 8px;
        font-size: 13px;
    }
    QLineEdit:focus {
        border: 1px solid #4285F4;
    }
""")

# Login button
login_button = QPushButton("Login")
login_button.setFixedHeight(40) # Fixed height for button
login_button.setStyleSheet("""
    QPushButton {
        background-color: #4285F4;
        color: white;
        border: none;
        border-radius: 5px;
        padding: 5px;
        font-size: 15px;
        font-weight: bold;
    }
    QPushButton:hover {
        background-color: #3b78e7;
    }
""")
```

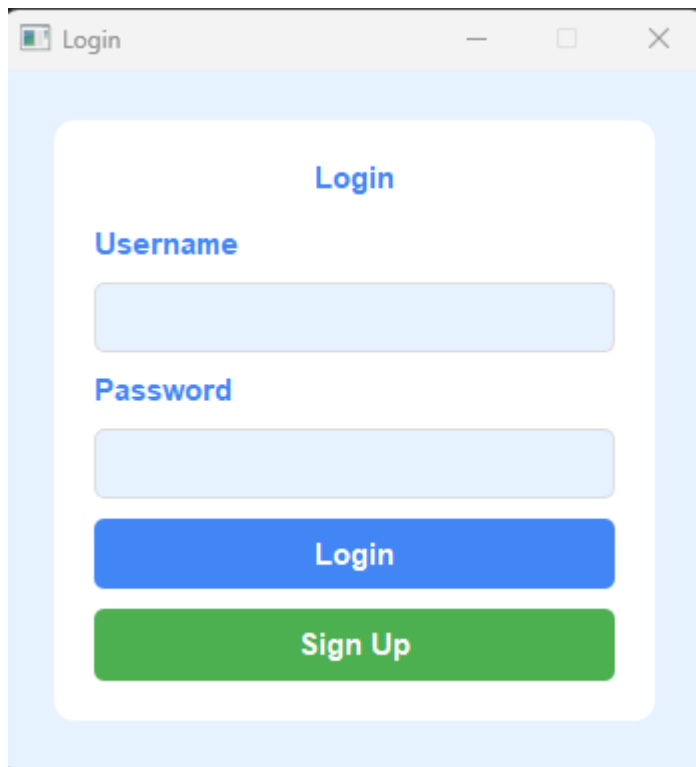
Above I used inline styling but to get to the required design of the window, I had to use too much and it looked ugly inside the python file, the lack of visual cues like what colour I had chosen made it harder than if I did it inside a CSS file since I have an extension that can show the colours, the inline styling also can't be used in conjunction with IntelliSense which is an extension that does things like suggesting the possible colours without needing an RGB code and closing brackets that are opened.

```
Dev > # style.css > QPushButton#loginButton
11  QLabel#loginLabel {
14  }
15
16  QLabel {
17      color: #4285F4;
18      font-family: Arial;
19      font-size: 11pt;
20  }
21
22  QLineEdit {
23      border: 1px solid #ddd;
24      border-radius: 5px;
25      padding: 8px;
26      font-size: 13px;
27      height: 35px;
28  }
29
30  QLineEdit:focus {
31      border: 1px solid #4285F4;
32  }
33
34  QPushButton {
35      border: none;
36      border-radius: 5px;
37      padding: 5px;
38      font-size: 15px;
39      font-weight: bold;
40      height: 40px;
41      color: white;
42  }
43
44  QPushButton#loginButton {
45      background-color: #4285F4;
46  }
```

The use of styling in a css file and importing the stylesheet meant that I could more easily change the code since it gave visual cues like the colour and by clicking on the colour square, I was also able to change the colour by clicking on the colour wheel rather than changing the HEX code. It also increased the modularity meaning if I wanted a certain type of button to be styled in a certain way across the whole file rather than having to individually change each one.

```
QWidget {
    background-color: #e6f2ff;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 14pt;
    font-weight: bold;
}
```

The QWidget attribute has 3 possible fonts that it can cycle through if one isn't available then it goes to the others, I set the font-weight to bold so that it is more visible by people with visually impaired, this makes the game more accessible, I also used bright colours that contrast well to make the game easier to see when someone's vision isn't as good



For generator.py I used an if statement to make the generator only generate rocks with grass and logs only with water. I also added a road option because it will make the game more challenging to have a road option.

```
def update_camera(self):
    previous_camera_y = self.camera_y
    self.camera_y += self.camera_speed_y

    # Keep player in the same relative position
    camera_change = self.camera_y - previous_camera_y
    self.player.y -= camera_change

def draw(self):
    self.screen.fill((255, 255, 255))
    pygame.draw.rect(self.screen, (0, 0, 0), self.player)
    pygame.display.flip()

def run(self):
    running = True
    while running:
        running = self.handle_events()
        self.update_camera() # Added camera update
        self.draw()
        self.clock.tick(30)
    pygame.quit()

if __name__ == "__main__":
    game = ChangingScreen()
    game.run()
```

I changed the run function to have the camera_update method inside which changes the

```
rangen = open('random_generations.txt', 'w')
#opens the files to write into
translate = {
    'w': 'water',
    'r': 'road',
    'g': 'grass',
    'l': 'log',
    's': 'stone'
}
```

The code opens the file that will be used to store all the random generations for the game, the dictionary translates, give the value to the key to the acronym for each like w meaning water and r meaning road.

```

for z in range(200):
    randoms = []
    if z % 3 == 0: #I want there to be more land than water or road since that makes more sense
        row_type = "water"
    elif z % 3 == 1:
        row_type = "road" #goes through variable z and alternates between land, water and road
    else:
        row_type = "land"

    You, 1 second ago • Uncommitted changes

    You, 1 second ago • Uncommitted changes
    for j in range(10):
        if row_type == "water":
            randoms.append(random.choices(['w', 'l'], weights=[65, 35])[0]) #sets weights to probability of getting water and log
        elif row_type == "road":
            randoms.append(random.choices(['r'], weights=[100])[0]) # sets a 100% probability of getting road since the road will always be road
        else:
            randoms.append(random.choices(['g', 's'], weights=[70, 30])[0]) #sets the weights to probability of getting grass and stone

    if row_type == "water" and 'l' not in randoms: # makes it so that the game isn't unplayable by making sure there is atleast 1 log per water strip
        random_position = random.randint(0, 9)
        randoms[random_position] = 'l'

    rangen.write(str(randoms)) #converts the list to a string and writes it to the file
    rangen.write('\n')

print("Generated 200 rows of terrain for Frogger")
rangen.close()

```

The code counts from 1 to 200 and at constant intervals makes sure that each of the obstacle layers are different types like making an equal number of water and road and more land since that is how the game should be set out, it uses the MOD function to decide based on the result, the code also makes it so that you can only get logs when you have a water layer and you can only have stone when the layer is land, this stops illogical layers like having water with land and stone with water, it also makes sure that there is at least 1 log per water layer since that keeps the game unplayable which would defeat the purpose of the game. I also set weights to each of the obstacles possible in each layer because there should always be more land/ water than logs and stone. The program then converts everything to a string just to make sure and then appends it to the previously opened file, it then prints a message saying the process has been completed, closes the file to save memory and finishes the program.

```

178 ['l', 'w', 'w', 'w', 'l', 'w', 'l', 'l', 'l', 'w']
179 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
180 ['s', 'g', 's', 's', 'g', 's', 's', 'g', 'g', 'g']
181 ['l', 'w', 'w', 'l', 'l', 'l', 'w', 'w', 'l', 'l']
182 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
183 ['s', 'g', 'g', 's', 'g', 'g', 'g', 'g', 'g', 'g']
184 ['l', 'w', 'l', 'w', 'w', 'l', 'l', 'w', 'w', 'w']
185 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
186 ['s', 's', 'g', 'g', 'g', 'g', 'g', 's', 'g', 'g']
187 ['w', 'w', 'w', 'w', 'w', 'w', 'l', 'w', 'l', 'l']
188 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
189 ['s', 's', 'g', 'g', 'g', 'g', 's', 'g', 'g', 'g']
190 ['w', 'w', 'w', 'l', 'w', 'w', 'w', 'w', 'w', 'w']
191 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
192 ['g', 'g', 's', 's', 'g', 's', 's', 'g', 'g', 's']
193 ['l', 'w', 'l', 'w', 'l', 'w', 'w', 'w', 'w', 'w']
194 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
195 ['s', 's', 's', 'g', 's', 'g', 'g', 'g', 's', 'g']
196 ['w', 'w', 'w', 'w', 'l', 'w', 'w', 'w', 'w', 'l']
197 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
198 ['g', 's', 's', 's', 'g', 'g', 'g', 'g', 's', 's']
199 ['l', 'w', 'l', 'w', 'l', 'l', 'l', 'w', 'w', 'w']
200 ['r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r', 'r']
201

```

This shows the results of the random generation in the above program, it also shows how the above program regularly generates each of the obstacle layers and this means there a certain number of each one of the layer types.

```

def handle_login(username_input, password_input, login_window):
    username = username_input.text()
    password = password_input.text()

    if not username or not password:
        QMessageBox.warning(login_window, "Error", "Please enter both username and password")
        return
    #error handling for no inputs
    if validate_user(username, password):
        print("Login successful")
        login_window.close()
        #validation for the user by checking the database

        import menu#imports the file menu from the same folder since it is required for the next step
        menu_window = menu.menu(username)

        login_window.menu_window = menu_window
    else:
        QMessageBox.warning(login_window, "Login Failed", "Invalid username or password")
        #error handling for invalid username or password shows a field that says invalid username or password

```

The above function assigns the variables username and password to whatever has been entered into the text box and checks if the user has entered both a username and password and gives an error message if they haven't, it then enters the variable username and password into the function validate_user which checks to see if the username and password are both the same as any in the SQL database.


```
def handle_signup(username_input, password_input, login_window):#for the sign up button
    username = username_input.text()
    password = password_input.text()

    if not username or not password:
        QMessageBox.warning(login_window, "Error", "Please enter both username and password")
        return#error handling for no inputs

    try:
        add_user(username, password)#try and expect for the add user function to add the user to the database
        QMessageBox.information(login_window, "Success", "User signed up successfully")
    except Exception as e:
        QMessageBox.warning(login_window, "Error", f"Failed to sign up: {str(e)}")
```

The program opens the css file and the creates a window with the size 350 by 350 pixels, it then creates two text input boxes for username and password, it makes the password look like circles so that it doesn't show the actual password

```
def create_login_window():
    login_window = QWidget()

    # Load stylesheet
    with open("style.css", "r") as file:#uses a css stylesheet to make the code more modular and add styling to the login window
        stylesheet = file.read()
    login_window.setStyleSheet(stylesheet)

    login_window.setWindowTitle('Login')
    login_window.setFixedSize(350, 350)
    #sets all the conditions for the login window like size and width
    container = QFrame(login_window)
    container.setGeometry(25, 25, 300, 300)

    layout = QVBoxLayout(container)
    layout.setContentsMargins(20, 20, 20, 20)
    layout.setSpacing(10)

    # Login header
    login_label = QLabel("Login")
    login_label.setAlignment(Qt.AlignCenter)
    login_label.setFont(QFont("Arial", 18))
    login_label.setObjectName("loginLabel")
    layout.addWidget(login_label)

    # Username input
    username_input = QLineEdit()
    username_input.setPlaceholderText("Username")
    layout.addWidget(username_input)
```

```
self.start_y = 0
self.width, self.height = width, height #how big the game window is

# Calculate grid sizes first
self.size_w = self.width // 10 #splits screen into 10 columns
self.size_h = self.height // 10 #splits screen into 10 rows
self.spacing = 0
self.size_w = self.size_w - self.spacing

self.camera_x, self.camera_y = 0, 0 #where the camera starts
self.camera_speed_x, self.camera_speed_y = 0, -5 #only goes up
self.objects = [] #empty list for all the squares
self.white = (255, 255, 255)
self.green = (0, 255, 0)
self.red = (255, 0, 0)
self.blue = (0, 0, 255)
self.brown = (139, 69, 19) #for the logs
self.black = (0, 0, 0)
self.grey = (128, 128, 128) #for roads
self.yellow = (255, 255, 0) #for the yellow lines on roads

#try to grab the log image
try:
    self.log_image = pygame.image.load("assets/log.png") #log pic
    self.log_image = pygame.transform.scale(self.log_image, (self.size_w, self.size_h))
except pygame.error:
    print("Couldn't find log image. Using boring brown rectangle instead.")
    self.log_image = None
```

The game uses OOP to set variables to set the colours that can be used and sets the rgb values to what they should show up, the game also uses log.png which is an image file that I created to show up in as the representation for the logs, it uses a try and except file to check if the image file exists and then uses a brown rectangle instead and sets the image variable to nothing.

```
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
Whoops! Camera update error: 'ChangingScreen' object has no attribute 'add_new_row'
```

When I was writing code using the function add_new_rows, I accidentally misspelled the function name and took away the s at the end, this cause an error to pop up that was caught by my try and except and this stopped the game from crashing and made the bug easier to diagnose since it just printed it there in the console.

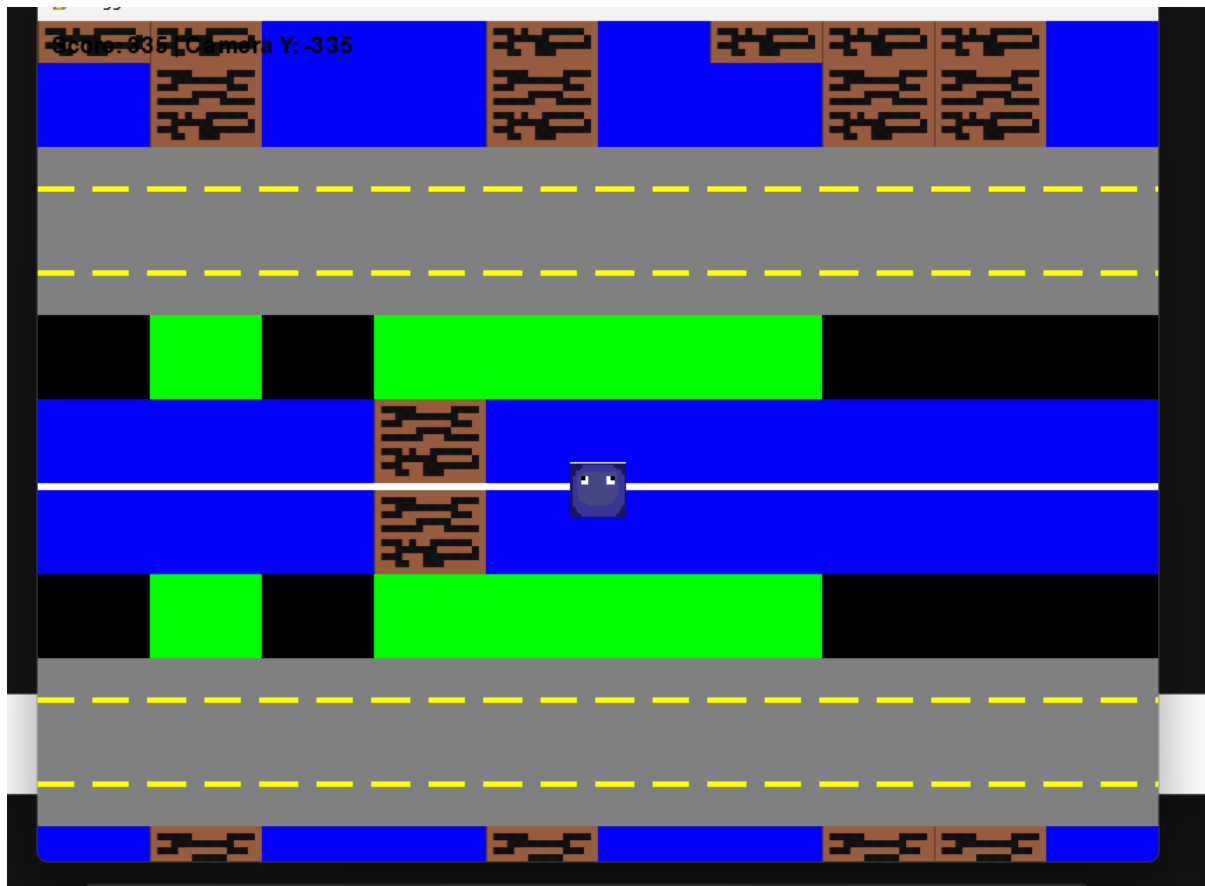


The Strips keep repeating for the length of the window each time a new one had to be generated so I changed the code to try and make it so that only 1 strip was generating each time, but this caused the game to crash before anything was rendered since there were too many iterations of the game refreshing which pygame can't handle, so i change the code to change the order in which the logs were rendered to increase the time they had to be rendered




I managed to create skins using pixel art that can be used in the game, however due to time constraints I am unable to make any more skins so the 3 that I have are a bee, squid and of course a frog since the game is inspired by frogger. Below is the bee skin, I also managed to make the logs (the brown tiles with the black lines) fit the whole square by using the `pygame.transform.scale()` function which will help to make it fit the space.


I also changed the order in which the calculations were made to increase the time that is given to make the log get drawn since that is a more intensive task. This means it can concurrently load the log in faster meaning there is time left at the end of each cycle for everything which is less resource intensive.



This is a picture of the squid skin in the game, since i was unable to cut off the top, there is a bit of white left which isn't meant to be part of the skin however I rendered the skin onto the surface of the sprite so the whole snip that was used is pasted onto the surface of the sprite. This could be remedied by cutting the white out using the freehand snip however time is short at this point in the project.



```
QScrollBar:vertical {
    border: none;
    background:  #f0f0f0;
    width: 10px;
    margin: 0px;
}

QScrollBar::handle:vertical {
    background:  #4285F4;
    min-height: 20px;
    border-radius: 5px;
}

QScrollBar::add-line:vertical, QScrollBar::sub-line:vertical {
    height: 0px;
}
```

multiple passwords having the same space. When I was writing the code for this file, I accidentally installed the wrong library so when I was trying to use the functions in the documentation the functions kept showing up as not existing.

```
#Add a new player
def add_user(username, password):
    try:
        hashed_password = hash_password(password)
        cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_password))
        connect.commit()
    except sqlite3.IntegrityError:
        print(f"Username '{username}' is taken already. Pick another one.")

#Check if login is legit
def validate_user(username, password):
    hashed_password = hash_password(password)
    cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, hashed_password))
    return cursor.fetchone() is not None

#Update player's high score
def update_high_score(username, score):
    cursor.execute("UPDATE users SET high_score = ? WHERE username = ? AND high_score < ?", (score, username, score))
    connect.commit()

#Get the best players
def get_high_scores():
    cursor.execute("SELECT username, high_score FROM users ORDER BY high_score DESC LIMIT 10")
    return cursor.fetchall()

#Setup the database
create_tables()
```

The function `add_user()` uses a try and except to check if the user already exists and gives an error if there is an error to the integrity of the data in the database, otherwise, it hashes the password and inserts the username and password into the database. The function `validate_user()` checks if the username and password entered `pyqt.py` are the same as the details in the database, it does this by checking the hashed version of the password in the database and hashing the password entered. The function `update_high_score()`

<https://youtu.be/PSyTE-NfdWA> Initial Game being played.

<https://youtu.be/7le0ezm4LtQ> The Login window and menu working.

<https://youtu.be/qBkg2Pj0O0w> The Sprite no longer being stuck to the camera and having to be moved backwards.

<https://youtu.be/s5U8wL90Q08> The Obstacles generate one of each at a time rather than ten of the same in a row.



```

self.obstacle_types = {
    'g': {'color': self.green, 'type': 'grass'},
    'r': {'color': self.grey, 'type': 'road'},
    'l': {'color': self.brown, 'type': 'log'},
    'w': {'color': self.blue, 'type': 'water'},
    's': {'color': self.black, 'type': 'stone'},
    't': {'color': self.red, 'type': 'truck'} #adds a truck type
} #links letters to obstacle types and colors

```

The next step in the implementation is adding the trucks in, I added a truck representation into the obstacle_types of dictionaries, this means that it will have all the data required, stored in one place.

```

for obj in self.objects:
    if obj["type"] == "truck":
        obj["rect"].x += 5 #Moves truck to the right
        if obj["rect"].left > self.width: #makes it go back to the left side of the screen
            obj["rect"].right = 0 #makes the trucks go back to the left side of the screen

```


I then added in a for loop that would iterate moving the truck until it gets to the end of the screen at which point it would be moved back to its starting position.

```
for obj in self.objects:
    if obj["type"] == "truck" and self.player.colliderect(obj["rect"]):
        self.game_over = True
        self.game_over_message = f"Game Over, {self.username}! You were hit by a truck!"
        return False
```

The game uses a conditional to check if the player is hitting the truck and if they are then the game ends and the boolean changes, otherwise, nothing changes and the game keeps going on.

```
if obj["type"] == "truck":
    pygame.draw.rect(self.screen, obj["color"], rect) # Draw the truck
```

The code draws the truck into the space using the “color” attribute.

```
for col, terrain in enumerate(terrain_list):
    if terrain in self.obstacle_types:
        props = self.obstacle_types[terrain]
        x = col * self.size_w
        obj = {
            "rect": pygame.Rect(x, y, self.size_w, self.size_h),
            "color": props['color'],
            "type": props['type']
        }
        if props['type'] == "road" and random.random() < 0.2: # 20% chance to add a truck
            obj = {
                "rect": pygame.Rect(x, y, self.size_w * 2, self.size_h), # Truck is wider
                "color": self.red,
                "type": "truck"
            }
        self.objects.append(obj)
```

The code checks if the surface that is being rendered is a road and if it is, it has a 20% chance of a truck being rendered onto the road , it is represented by a red square for now, the truck is wider than 1 tile as it needs to be bigger.

```
for obj in self.objects:
    if obj["type"] == "truck":
        obj["rect"].x += 5 # Move truck to the right
        if obj["rect"].left > self.width: # Reset position if it moves off-screen
            obj["rect"].right = 0
```

This piece of code moves the truck to the right iteratively and keeps checking if the truck has reached the other side of the screen, if it has, then the truck will be moved back to the starting position on the other side of the screen by using the 5th line to set the position horizontally to 0.

```
for obj in self.objects:
    if obj["type"] == "truck" and self.player.colliderect(obj["rect"]):
        self.objects.remove(obj) # Remove the truck if it overlaps with the player
```

This checks to see if the trucks have collided with the player at the start of the game and if it has, then it will remove the truck from the map, but this only works at the start of the game since we want a grace period when the player must get oriented.

```
print(f"Player position: {self.player}")
for obj in self.objects:
    if obj["type"] == "truck":
        print(f"Truck position: {obj['rect']}")
```

<https://youtu.be/7XDr7859Q4M> This video show the truck causing the game to end before the player is ready to play the game, this is caused by the trucks being spawned in too early right when the whole game is rendered, this shuts down the game and makes it unplayable, the way I will improve upon this is add a grace period and zone where there are no trucks.

To do this I will add a condition that checks whether we are at the start of the game

```
def add_new_rows(self, terrain_list, y):
    cols = int(self.width / self.size_w)
    for col, terrain in enumerate(terrain_list):
        if terrain in self.obstacle_types:
            props = self.obstacle_types[terrain]
            x = col * self.size_w
            obj = {
                "rect": pygame.Rect(x, y, self.size_w, self.size_h),
                "color": props['color'],
                "type": props['type']
            }
            # Avoid placing trucks in the player's starting row
            if props['type'] == "road" and random.random() < 0.2 and y != self.height // 2:
                obj = {
                    "rect": pygame.Rect(x, y, self.size_w * 2, self.size_h),
                    "color": self.red,
                    "type": "truck"
                }
            self.objects.append(obj)
```

The second line calculates the number of columns in the window, this means the game can be scaled for any resolution. The third line iterates over the terrain variable and col variable, then it checks if the terrain is valid by checking if the letters in the generated list exist in the dictionary at the top. A rectangle is created and then coloured with the respective colour based on the terrain and then a type is given, the if statement checks if the terrain being generated is a road by using the type to check, it then looks to see the coordinate of the top of the camera and either draws the truck if the camera is far enough along otherwise the condition isn't met and nothing happens. The self.objects.append() function adds the trucks into the self.objects() list.

```
elif obj["type"] == "water" and self.player.colliderect(adjusted_obj_rect):
    print(f"Water collision detected! Player at {self.player}, Water at {adjusted_obj_rect}")
    self.game_over = True
    self.game_over_message = f"Game Over, {self.username}! You fell into the water!"
    return False
```

The game checks if the type of the object that the player is on is the water and if it is then the game outputs a debug statement(this is only there while developing the game) that outputs the cause of death, sets the game over variable to true and outputs the cause of death of the player

```
if obj["type"] == "water":
    water_tiles += 1
    # Add a blue border around water tiles for visibility
    pygame.draw.rect(self.screen, (0, 0, 128), rect, 1)
```

The code checks to see if the type of the object is water and if it is, then it adds a border around the water to make it more visible.

```
# Add a truck only if more than 5 seconds have elapsed
if elapsed_seconds > 5 and random.random() < 0.2 and y != self.height // 2:
    truck_obj = {
        "rect": pygame.Rect(x, y, self.size_w * 2, self.size_h),
        "color": self.red,
        "type": "truck"
    }
    print(f"Created truck at {truck_obj['rect']} at elapsed time {elapsed_seconds}s")
    self.objects.append(truck_obj)
```

This checks to see if at least 5 seconds have elapsed in the game and only spawns in if those 5 seconds have elapsed.

```
random.shuffle(self.level_data)
#ensure first row is always grass
grass_row = ['g'] * 10
self.next_rows = [grass_row] + self.level_data[:19]
self.level_data = self.level_data[19:] if len(self.level_data) > 19 else []
```

I was having problems with the sprite spawning on water so I first tried making a grass tile generate at the starting location of the sprite however wasn't able to get the maths to work so I made sure the first few layers are always fully grass, this means that the sprite will always spawn in a valid place and give enough time for the player to react

```

# Always include at least 3 grass rows at the beginning
grass_rows_at_start = 3
if not self.next_rows or len(self.next_rows) < grass_rows_at_start:
    self.next_rows = []
    for _ in range(grass_rows_at_start):
        self.next_rows.append(['g'] * 10)

for i, terrain_list in enumerate(self.next_rows):
    y = top_y - (i * self.sizeh)
    # For the first 3 rows, always force grass
    if i < grass_rows_at_start:
        terrain_list = ['g'] * 10
    self.add_new_rows(terrain_list, y)

#replenish next_rows when we're running low
if hasattr(self, 'level_data') and self.level_data:
    # Always include 3 grass rows at the beginning for safety
    self.next_rows = []
    for _ in range(3):
        self.next_rows.append(['g'] * 10)

    # Add the rest from level data
    for i in range(5):
        if i < len(self.level_data):
            self.next_rows.append(self.level_data[i])

```

I found that it would make the game more enjoyable if there were only five instead of twenty at the start of the game.

```

def is_safe_input(input_string):
    # Check for common SQL injection patterns
    sql_patterns = [
        r'(\s|^)(SELECT|INSERT|UPDATE|DELETE|DROP|ALTER|UNION|CREATE|EXEC|INTO)\s',
        r'(--|;|/|\*|\\*)',
        r'(=\s*\'.*\')',
        r'(1\s*=\s*1)',
        r'(\s*\s*OR\s*\s*)',
        r'(\s*\s*OR\s*1\s*=\s*1)'
    ]

    for pattern in sql_patterns:
        if re.search(pattern, input_string, re.IGNORECASE):
            return False

```

The above code checks the input that is give into the login page and checks if there are any of the normal SQL keywords like SELECT that could be used to retrieve the data from the database.

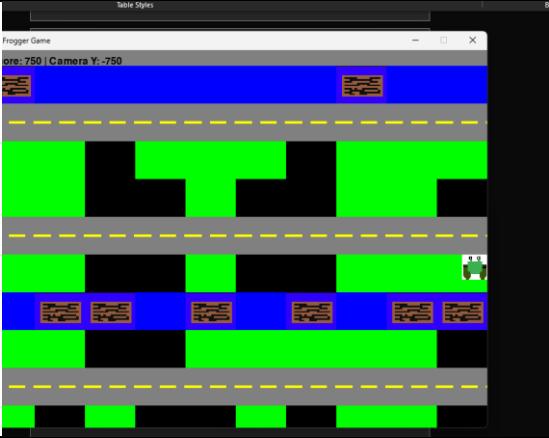
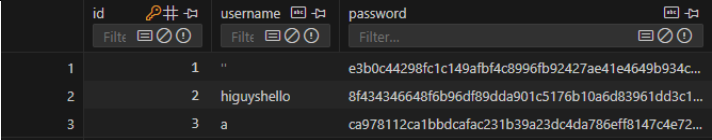
```
import os
import subprocess
import shutil
import sys

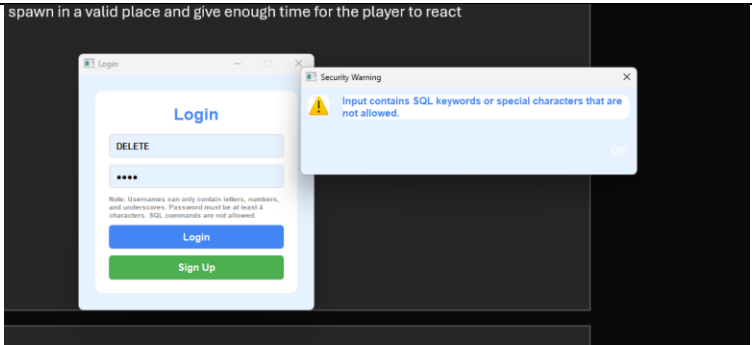
def build_executable():
    print("="*50)
    print("Building Frogger Game Executable")
    print("="*50)

    # Make sure PyInstaller is installed
    try:
        import PyInstaller
        print("PyInstaller is already installed.")
    except ImportError:
        print("Installing PyInstaller...")
        subprocess.call([sys.executable, "-m", "pip", "install", "pyinstaller"])
```

I tried using PyInstaller but was unable to create an exe file.

Testing

Test No. and Pass/Fail	Evidence	Outcome
1		The Character can get to the side without causing game over and game over happens only when the back is hit
2	https://youtu.be/CC1IVGawJGc	I can choose whichever skin I want out of the options
3	https://youtu.be/CC1IVGawJGc	I can login to the game with my own username and it is stored in the .db file
4		The .db file has the passwords hashed so that they cannot be read in plaintext if the file is accessed
5	https://youtu.be/QH3O5aKE6XQ	The Player dies when they hit the back of the screen
6	https://youtu.be/lxuTC-HhF4A	The sprite is visible with the correct skin selected
7	https://youtu.be/YTztDJVw3r4	The sprites stop when they hit the obstacles.
8	https://youtu.be/xsU4usPzJA	The game ends when the player hits a truck
9	https://youtu.be/KO_E8v8CLD0	The player dies and the game

		ends when they step onto water
10	https://youtu.be/I2YuoDRHWX0	The number of points goes up when the player lives for longer
11	https://youtu.be/uKPXqQ8UJ0o	Gives a message saying the details are wrong
12		The new high score is saved into main.db
13	https://youtu.be/0oTaFSh_8yQ	If the keys are opposing, then they cancel out but if they are not then the player moves in that diagonal
14	spawn in a valid place and give enough time for the player to react 	The game gives an error saying that SQL keywords are not allowed
15	https://youtu.be/gYRoX1bv1Bg	The game goes to the menu when the player dies
16	https://youtu.be/kZkWFQy0gLk	The data is saved in the .db file and can be retrieved later
17	I was unable to get the PyInstaller library to work as the importing of dependencies was too complicated for the time, I had left.	The game opens onto the login page when the exe file is clicked

Fixes after testing

- Fixed the trucks to render in correctly and be visible in their actual positions.
 - Did this by giving them a rectangular border around where they are, this means the player can see and avoid the trucks, used the draw method in pygame for this?

Evaluation

One of the ways I could have improved my games would have been to make it in C and use a lightweight library to create the game, this would have kept me well within the limits of the spec that I gave myself for the system resources that can be used and would have made the game more accessible. The best library for the job would have been SDL which can make 2D games easier.

Specification

These were my success criteria at the start of the project:

- I will use the checker in pygame to move when the WASD buttons are clicked.
- I will use PyInstaller to compile the game into an exe file that can run.
- I will use PyQt to create a GUI that can interact with the user and allow a login page.
- I will create a .db file that will use SQL to save high scores which can be retrieved.
- I will use PyQt to create a menu window that has start game, exit, skins and settings in it.
- I will incorporate graphics from the original frogger game as well as create some newer art.
- I will use PyGame to add obstacles which will have a car skin and cause the game to be over if the player touches them.
- I will code in representations of obstacles and make the game not allow the player to go through.
- I will include a menu using PyQt that will have clearly labelled to help the player navigate.
- I will add a button in the menu that can help change the speed of the character and the camera to make it harder for the character to get forward.
- The frog can ride on moving logs without falling in the water.

For my game one of the things I could do in the future is include a pathfinding Bot that checks whether adding certain layers will keep the game playable and will take appropriate action to stop that like stopping that layer from being placed and making another one be chosen randomly instead, I could do this by writing an A* or Dijkstra's algorithm that will check the layers before they are loaded in.

WASD being used to move the character because the survey chose this as being the best way of movement. I will use the checker in pygame to move when the WASD buttons are clicked.	The game uses WASD as show in the code in the implementation above
The game should be an exe file as the player should not be able to access the source code to cheat and mod the game and compiling it before, makes it easier to run without touching the console. I will use PyInstaller to compile the game into an exe file that can run.	I chose this requirement as I wanted to prevent cheating and being able to access the source code to change it however, I was unable to get this success criteria done since I was unable to get the dependencies to work correctly an open one after another.
There should be a login system so the player can carry on his progress with his name attached to see his progress and so multiple sessions can be started. I will use PyQt to create a GUI that can interact with the user and allow a login page.	Login system shown in test three where there are the options to create and store users as well as login to the users to store and save the data in the .db file along with the usernames and passwords.
There should be high scores saved so that you can look at what your best score and beat it each time. I will create a .db file that will use SQL via sqlite3 to save high scores which can be retrieved.	https://youtu.be/jLOY2Bnanxw The high score is being saved in the .db file like the specification asks for and sqlite3 is used since it is the most effective way
There should a menu that can be interacted with so that the player can use it. I will use PyQt to create a menu window that has start game, exit, skins and settings in it.	There is a menu which uses menu.py and has clearly labelled buttons that are easy to navigate and a good order for the UX design

	like exit being at the bottom and start game being at the top
A mix of a retro theme and a modern theme since the form that the stakeholders submitted said they wanted a mix. I will incorporate graphics from the original frogger game as well as create some newer art.	The game looks like a good mix of retro and modern themes, the bright green colour for the grass makes the game look more retro along with the dark blue water but the style of the pixel art is a bit more modern like the crossy road game.
Trucks as obstacles since I want it to be inspired by Crossy Road. I will use PyGame to add obstacles which will have a truck skin and cause the game to be over if the player touches them.	I was able to use trucks as obstacles that move when the player is trying to get past, this is shown in my testing above
Obstacles that stop movements to make the game a bit harder so you cannot cross anywhere, I will code in representations of obstacles and make the game not allow the player to go through.	The game renders all the obstacles using the specific colours and assets, they also stop the game if a player hits them
A GUI in the game to make it easier to understand and use the application. I will include a menu using PyQt that will have clearly labelled to help the player navigate.	The game has a clearly labelled menu with large buttons to help the player navigate easier
Multiple Levels to make the game more interactive, I will add a button in the menu that can help change the speed of the character and the camera to make it harder for the character to get forward.	I was unable to include multiple levels as I did not have enough time left because I spent too much time on trying to make the obstacles and trucks work better
The frog can ride on logs without falling in the water.	The frog can get past parts of the river using the logs it can fins as shown in my testing above



For my game one of the things I could do in the future is include a pathfinding Bot that checks whether adding certain layers will keep the game playable and will take appropriate action to stop that like stopping that layer from being placed and making another one be chosen randomly instead, I could do this by writing an A* or Dijkstra's algorithm that will check the layers before they are loaded in. I could also have used better graphics by making the game 3D like how crossy road works, I would have implemented that by using Unity instead of pygame for the main game since Unity is better suited for 3D game development since it has better tools like being able to visualize the game in development and the ability to create objects in the editor itself. I could have also used Panda3D if I wanted to keep using python since I am more comfortable with the syntax of python. If I had more time, I would have created more skins since that can improve the user experience of playing the game, i would have also made it so that not all the skins are unlocked at the start of the game, and the player has to earn rewards to unlock each of the skins, this would have made the game better since it would have given the player more motivation to play the game.

Feedback from client:

"The game is very fun to play however there are a few bugs where the trucks don't seem to render in correctly some of the time, I like that the game has a login system that works well, the security also seems good as I can't see the underlying files and databases. The game is fun to play, and I can play for hours on end. The skins make the game more enjoyable to play however there do not seem to be enough skins and I would prefer to have more skins. One of my gripes with the game is that the skins don't match the environment, for example, the squid goes on land and dies if it touched the water which is the opposite of what should happens, the game plays very smoothly normally and the 30 fps is fine for the game and I can't even tell that the frame cap is lower than 60. One of the things I do not like is that the movement of the sprite does not work like most normal games, the character moves too smoothly and does not move like in a tile-like fashion. I do like the menu system and each of the buttons and the styling of the game is well put together. I like the graphics of the game, and they are a perfect blend of the original graphics as well as the newer ones from crossy road."

I am happy with the response from the client as it seems mostly positive, and I am especially happy that he likes the graphics as I spent a longer time on that. I do understand that there are still a few bugs in the game like the trucks not rendering normally. My next port of call would have been to fix the specified bugs and add a few more skins.

