Riyad Hussain

Analysis:

Description of the problem

During exam season students will feel the pressure and stress. They would want to take their mind off it and relax but at the same time having a guilt feeling that they could revise. My game is the perfect solution to this problem. It is a game that you can play and enjoy but at the same time revise with quick fire questions helping you learn as you also have a sense of escapism forgetting about the stress as well as learning.

Using Python, In my project, I am planning to do a spinoff of a popular game in the 1980s called Frogger which I will be naming my game 'Chicken Run'. This is an arcade game which I think is more pragmatic to do due to the simplicity of the game. This will have an AI feature in the game which will learn what level of maths you are at and will ask you a question from that level. The AI feature in this program makes this solvable using computational methods using an iterative learning based method.

This program will be fit for under 18s due to it being a game and has a revision aspect in it by asking maths questions.

Frogger consists of a frog who has to try and get to the other side but the catch is the player will need to avoid the cars and water and reach the lily pads at the end. The aim of the game in Frogger is to try and get all the frogs to the end; the player is given 3 lives to try and complete it. As the player completes more and more levels it will become faster and harder to complete. The player starts off with 3 lives, the player will lose lives by hitting any cars or falling into the water and other obstacles. My game will be much similar to this but I will have a chicken who has to get the food to her chicks at the end.

I will use the module PyGame inside of Python which is an easier way to program games that run on Python.

The overall problem definition summarised is trying to create a game where the player needs to get to the other side without hitting the obstacles. When this is understood and solved the rest of the problem will be following algorithms that I will create such as hitboxes and answering the questions and what happens when they reach the end.

Identification of all the stakeholders

The genre of this game will be an arcade-educational game, therefore, the main target this game will appeal to is a teenager range audience around 9 - 17 as this game isn't too difficult to play and the upper quartile of my audience may play this as a time-passing game. Some parents may not want their children to play this game as it promotes running on roads while cars are coming, although there is an educational side in the game these 2 can cancel out. To be able to find out more information about my target I will have to carry out research and to do that I will interview and use questionnaires to find out how to make my game the most enjoyable. This game is also available to people around the world as there isn't a specific language the player needs to know to play but I may add a point system or timer which I will number in English also the maths questions will also be in English.

The game I am going to create is mainly created for casual gaming but as there is an aspect of education in the game it can appeal to a wider range of stakeholders.

Due to stakeholders, I will have to design a program which would be suitable for the age range of my stakeholders and to do that I should make it with a range of different and vibrant colours as that would be more aesthetically pleasing for my target. Making it simple and easy to navigate around the menu should be important due to the age of my stakeholders. I can't make it too complicated. I wouldn't expect people to burn out the game and play it every day, this game that i am making would usually be something to cure boredom for or if you're waiting for something and it'll help pass a little time. This game will be created on python which means it can only be playable on PC that has python installed although I am able to convert it into an .exe file which means it can be runnable on most modern computers.

The research will be carried out onto my school's year 12 and year 13 and I will email a form to everyone to fill out a form. This will help me get a wide range of answers and make it more accurate when it comes to deciding what the majority answers are.

Why it is suited to a computational solution

My problem will be solved computationally for numerous reasons such as; this problem will require a keyboard and mouse to navigate throughout the game which can only be done on a computer. My game will also be solved computationally due to it having an AI feature in the game. I will have independant AI mechanic in the game which can only be possible on a computer. The AI will use machine learning to understand the player where the more the player plays the game and answer the maths questions throughout the game the more the questions will be suited towards the player and it'll also be helpful for them.

- **Justification -** The game can be played by anyone and their data will be saved with a simple log in this is much better as they do not need to restart every time they close the window and will be stored permanently, along with that the AI aspect will make

the game perform better using probability to determine what the user will choose as the answer and give questions based on trial and error of each question answered.

Decomposition

This problem can be decomposed into specific detailed problems which are:

- Controlling the player using the keyboard
- Hitbox detection when collision with obstacle
- Producing maths questions using machine learning

After these main blocks of the problem have been solved the rest is usually simple, such as creating a menu system for the user to navigate around but the 3 blocks that have been listed will take most of my time and I will need to spend more time on them.

Abstraction

I will include abstraction in the game. I can't use too much data processing power as this will cause lag on some computers so in this game the images will be pixelated 8 bit style vector images.

- Simple sprite images to save time
- Tree's are the same images to save time
- Animation 5 frames or less to save processing power
- 8-bit vector images rescaling won't change resolution

The animations also won't be too complicated and will only make it less than a 5 image gif. I will also include nature such as trees and plants but I will most likely make them all the same e.g. one type of tree and one type of bush and I will copy them and place them around this will simplify and abstract the design it will also save time.

Agile Method

The way I will solve this is using the agile method which is where I follow the typical order and once I finish everything I go back to the start and amend anything and keep doing it till my problem is solved to the best standards. For my problem I will break it down in certain sections and work one section at a time and puzzle them together.

Research

Figure 1.4

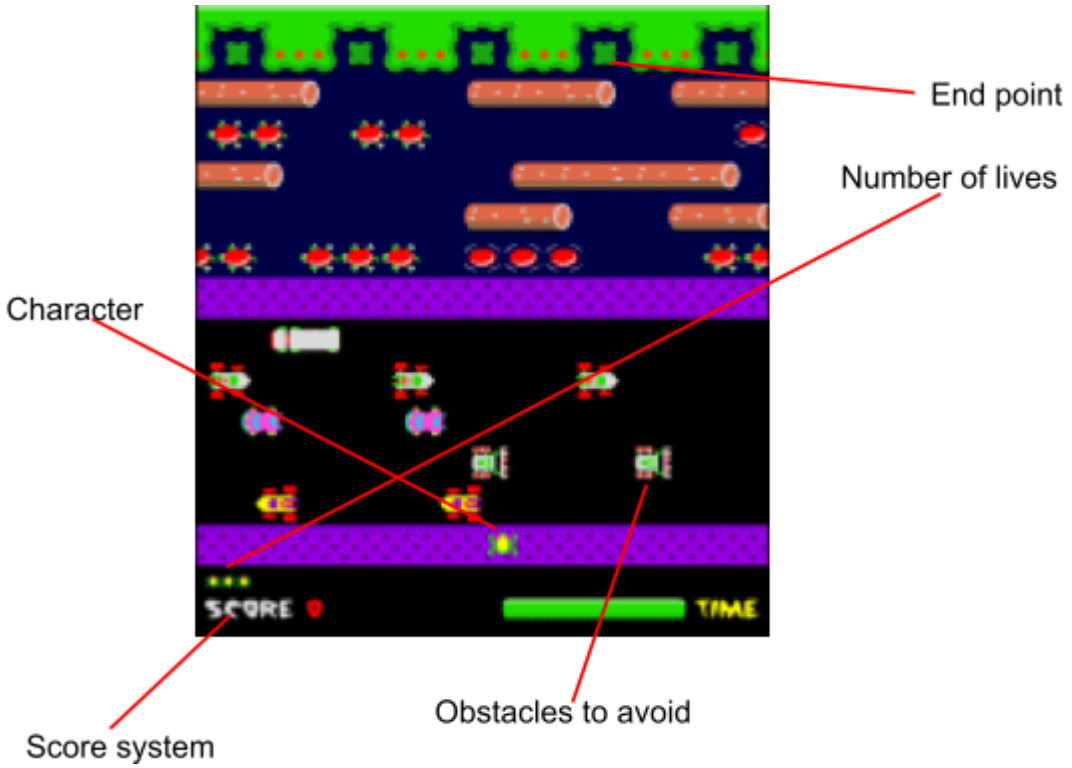


Figure 1.4: This is the original game ‘Frogger’ that my program will use as a scaffolding for my actual game. it is very simple, although the concept of the game is very interesting and fun to play, which is why I chose this game.

Score system: This is a numerical value which adds up a players score as they play. Below shows a table of every way a player can gain points the maximum amount of points a player can get is 99,990 the original arcade version only stores up to 5 digit numbers.

Successfully jumping Frogger forward	10 points
Successfully jumping Frogger home	50 points
Successfully jumping 5 frogs home	1000 points
Taking a lady frog home	200 points
Eating a fly	200 points
Extra points for time left	10 points

Character: The character is what the player controls/navigates around the game to reach the end. This character is controllable by keyboard arrows (UP, DOWN, RIGHT, LEFT). The character is a three frame animation which consists of the character switching from first frame which is the frog standing still to jumping then finally landing and standing still once a button is pressed then standing still instead for my game i will have a chicken that will flap its wings.

Obstacles to avoid: These are randomly generated sprites that move from left to right and vice versa it can be the cars on the road and also the water. If the player hits the cars or falls in the water they will respawn or the game ends.

Number of lives: Number of lives is a good factor to add when creating a game it gives a sense of pressure if you are on your last life and also doesn't make the game too easy. In this game you start off with 3 lives and every time you hit an obstacle you go down a life but you respawn if you lose all 3 lives the game ends.

End Point: This is the goal of the game you need to reach the lily pads for every frog there is 5 lily pads so you need to reach the end 5 times to win. For my game it will be different, there will be 3 chicks at the end and the chicken will need to bring the food to each one to win.

Figure 1.5

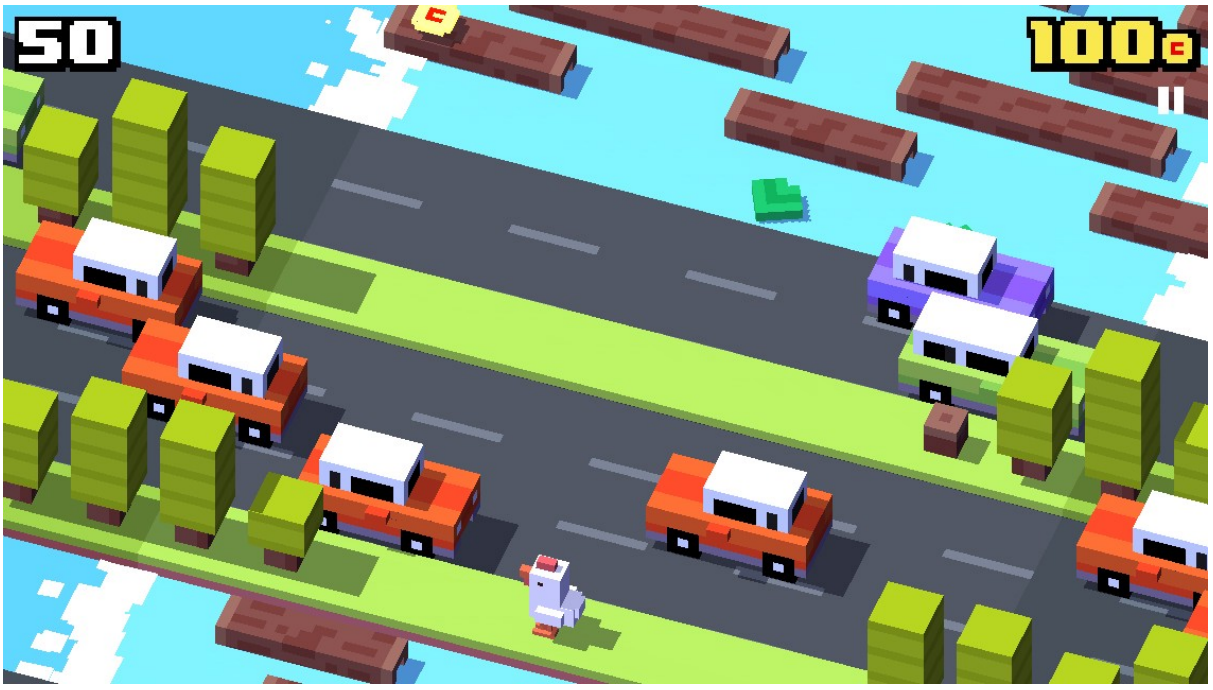


Figure 1.5 - This is the exact same concept as frogger but has been adapted to suit the current target audience by adding more sound to the game such as sounds of cars driving past and water splashing. This game is also 3D designed instead of the 2D pixel style like frogger. This game was originally released on mobile as a mobile app so the controls of the game is a tap on the screen to move forward.

These 2 games have similar mechanics to my game and I will use Frogger as my scaffolding whilst creating my game I liked the idea of having logs moving across the water and the player will have to avoid the water by jumping on the logs and I this can be a second level that I can include into my game.

Questionnaire

I have created a Google Forms questionnaire for students in my school to fill out this will help me understand what my stakeholders want in a game. I can also find alternatives to problems by using this and find out new suggestions and ideas.

Figure 2.1

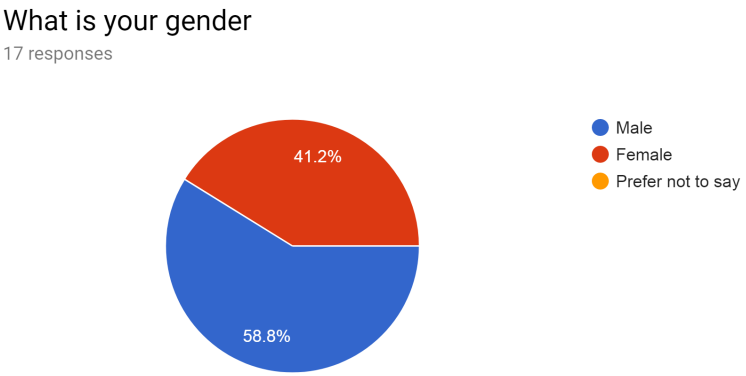


Figure 2.2

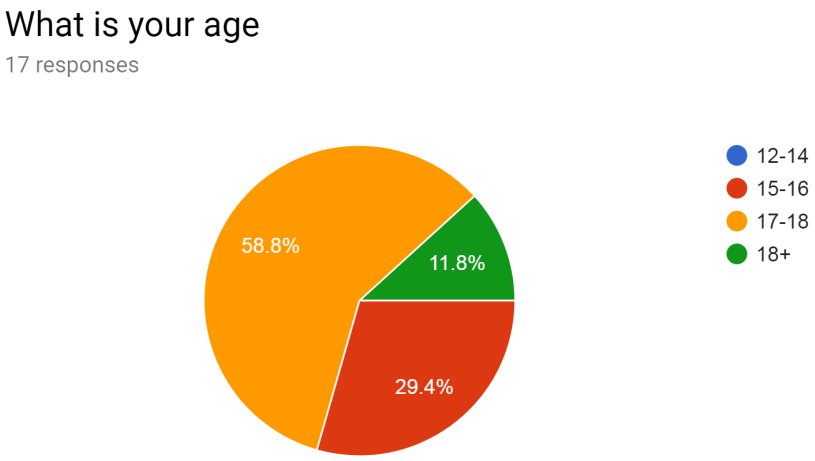


Figure 2.3

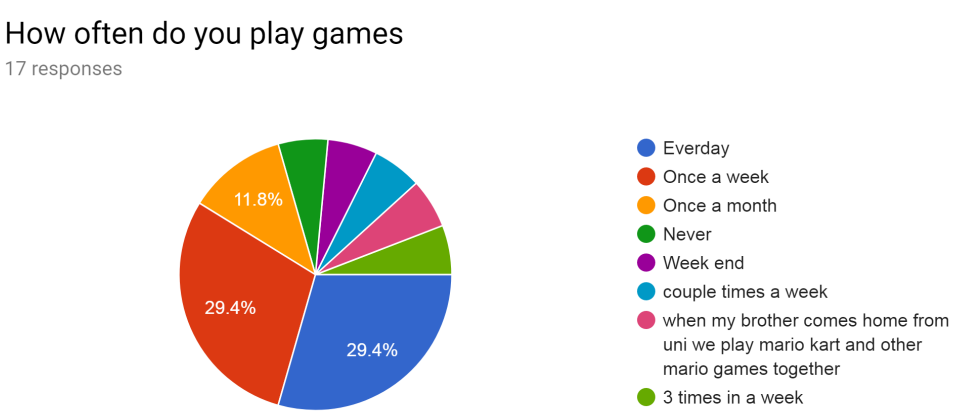


Figure 2.4

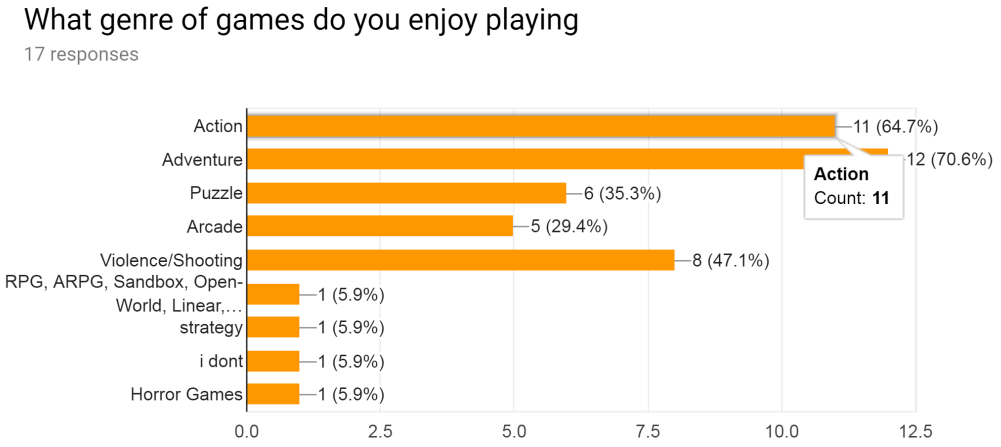


Figure 2.5

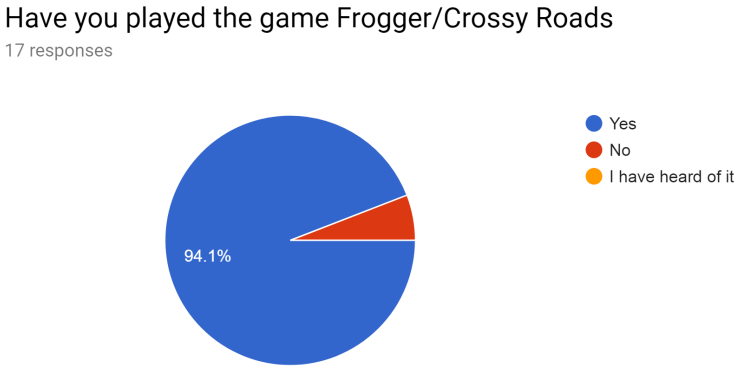


Figure 2.1 - I asked 'What is your age', to find out what age range people that I asked are which may affect the next answers. I also had to consider not all
answers will be accurate for example I asked people in year 12/13 for their age, some people put 18+ which may not be true. The majority (58.8%) were ages 17-18 which means my game should appeal to the older age of my stakeholders.

Figure 2.2 - Gender is another important factor while making the game it'll help me find out who I should appeal to more. 58.8% were male so I knew it was around 50/50 for my gender so I can make my game towards both genders.

Figure 2.3 - Asking people how often they play games is important I can find out what category my game will fit into. It was a tie between every day and once a week which helps me understand that there is a gap between people who play a lot and people who play moderately. My game would appeal to people who play once a week as it's not a game that people will play a lot.

Figure 2.4 - What genre of games people enjoy was a critical question to ask. I can create my game which will fit into the most popular choice which was an adventure but the action was also behind by 1 count. This will help me when creating my game I can aim it towards an adventure/action game.

Figure 2.5 - I asked if they have played the game Crossy roads/Frogger which would help for the next questions. 94.1% said that they did which is good as they'll understand how my game works.

Figure 2.6 - Asking people if they want an educational aspect in the game suprised me I was expecting the majority of people to say no but 70.6% said yes, which makes me more confident to add an educational feature into the game.

The next 3 questions were short answer questions:

1. **If yes, what did you enjoy most about Crossy roads/Frogger** - This is a follow-up question from when I asked have you ever played Crossy roads/Frogger. The majority of answers were the game was addictive, simple and not much brain power needed.

2. **If you did/do play games what was the most enjoyable game you have played and why** - The mode of the answers was Minecraft which was a very popular game and has a lot of unique ideas which I can implement in my game.

3. **If a new game was created like Crossy roads/Frogger what new features would you like to see in the game** - The answers to this question were quite interesting i wanted to see other peoples opinions on what makes a game like Crossy roads/Frogger better which i can add to my game. The majority of people said to add power-ups which I also was thinking to add and not to make it repetitive; extra minigames.

After looking at the responses to the questions it helped me understand what is needed in the game and what isn't needed. There will be some limitations although people have asked to see it in my game such as having a multiplayer based game which I don't think will be achievable in my project.

- **Justification -** The reason for this research will allow me to understand what the users want to see in this program. I can add and remove any previous ideas I have had based on majority ideas from the survey. As I go along with my design I can implement new features and get constant feedback from the stakeholders to see if it is a good idea or not.

Features of the proposed solution

After researching on the original and similar games and getting target market feedback it has given me a brief idea of what my game should include and disclude:

- I am going to make this an adventure game which means it has a story to it and reaches the end of the game and for the user they will need to explore and solve puzzles to complete the game. The adventure game choice was from my research as the majority of the people that answered the questionnaire chose adventure as the genre.

Justification

The reason for this is because the user has a reason to come back and keep playing the game to complete the story. This will keep the game entertaining and new.

- Another original idea I want to add in my game is adding power-ups and maybe some traps in the game. This will create a sense of excitement whilst playing the game. I also added it because it was a popular response people asked me to add.

Justification

Adding power-ups enhances the gameplay making it more exciting and due to research and responses from the stakeholders.

- To complete the original game you would have to get to the end 5 times. I will not be adding that, instead you only reach the end once but it gets harder every level. The reason I chose this is because reaching the end 5 times may get repetitive and people will find it boring completing it once will give that sense of victory until you move onto the next level.

Justification

The game will get harder with harder questions each time until you run out of lives. This way there is no limit and I can implement a score system such as how many questions you got right in the amount of lives you had left.

- I will include some level of complexity such as if you die you can answer a maths question to get an extra life and depending on how quickly you answer it you get lives for that e.g. if you answer the question in 1 second out of 5 you get all 3 lives back. This will also be an AI integrated method such that as you answer more questions it will use machine learning to find what you struggle with and what you find easy. The reason for this is because adding an educational section to the game will help appeal to students who want a break from i.e. revising but whilst they are still doing some mental maths they are also playing a game.

Limitations of the solution

I have identified i won't be able to add everything I want in this game and I have resolved it by finding alternatives:

- Some responses to my questions were to make it an online multiplayer game which will make the game more competitive and enjoyable with friends. This would be a limitation as I would need to be constantly in the command prompt of the computer which would need administrator mode which may be unsafe for the school. My knowledge in python for this is also limited although i may be able to research and find out i assume this will need a lot of research into network programming which I do not know.
    - **Justification -** This game will not be an online game due to my lack of knowledge and pragmatics of the situation. Another reason is I will not be able to test the online multiplayer version in school as it will not work due to security reasons.

- Another limitation is the fact that to implement AI i will need to use spreadsheets as part of my plan to introduce a Q table with probability distributions but python does not have the correct modules to be able to read and write on spreadsheets therefore I will need to solve this limitation another way.
    - **Justification -** Due to the module not installed on the computer I will need to use another module which will still work such as csv and store the data values in an excel csv file as an alternative option.

- Another limitation is that the game will include sound; the computer systems in the school do not output sound therefore to listen to the sound in the game I will need to test and program that at home.
    - **Justification -** This may slow the production due to me only being able to do this part at home but it shouldn't be too much of a problem. The only issue is I won't be able to hear anything on the school computers where an error may occur in the sound.

- Another limitation is that I can only use python to make the game due to my lack of knowledge on other languages and learning another language will take too long so I can stick to python as I have learnt that previously
    - **Justification -** From the listed usable programming languages Python was the only one I am familiar with there are many more programming languages that are designed for game programming but with enough research and practice Python is also a viable option.

Requirements

Minimum Hardware Requirements	Justification
Computer Keyboard	This is so the user can input key presses such as the arrows which will output the direction the sprite moves in.
Computer Mouse OR Trackpad	This so that the user can navigate through the Menu.
Audio Speaker	Audio output is optional for the game as it is still playable without it but it is more interactive if this is on.
2GB of RAM or higher	The minimum amount of RAM needed to run python IDLE.
2 GB of disk space	The amount of storage needed for the game files and images.
VGA or higher monitor display	This is needed to output the game on a screen.
Intel Atom® processor or Intel® Core™ i3 processor	This is the minimum processor requirement needed to run python IDLE.

Minimum Software Requirements	Justification
Windows* 7 or later, macOS, and Linux	Operating systems needed to run Python 3.6.0
Python 3.6.0	Software needed to run the game
Pygame module	For the game to run you will need pygame module to be installed which then Python can read

Success criteria

For my success criteria, I will want reachable goals for my design. This will help me set targets and understand what I have to do.

1. The program will have a login system which will allow the user to login and save permanent data such as their level and can close the program and re-open it without worrying about re-doing easy maths questions.
   Justification 1. - The reason for a login system is due to machine learning requiring a lot of generations of trial and error before it gets accurate results on what your level of maths is. With a login system the data will be stored and not reset so the machine learning can continue where it left off and get more accurate the more it learns..

2. After the user logs in they will be presented with a menu to navigate throughout the game. This will include 3 simple buttons play, leaderboards and quit the play button will be used to start up the game. Leaderboards to display the 5 top players with the highest scores and quit to close the program.
   Justification 2. - This is included to suit the stakeholders who will be children and teenagers so it needs to stay simple and easy to use. Making large buttons with clearly visible text is the best way to go about this problem. It also gives the user a sense of preparedness before they click play instead of instantly going into the game after they log in.

3. The player will be able to control the sprite chicken so that they can play the game and it will be controllable using the arrow keys. There will be a constant listener checking for if the player presses any keys and acts according to which key is pressed.
   Justification 3. - This is needed so that the user can play the game and complete the game task. Without the controls or listener the game will not be entertaining and no interaction between the user and the game.

4. The cars will generate randomly with different velocities and sprite images they will also generate from random positions and from left or right randomly and once one car reaches the other end it will respawn somewhere else randomly.
   Justification 4. - This is so the game doesn't get predictive by having the same velocity as well as easy to guess where the cars will come from if they don't spawn randomly.

5. As the cars will have different velocities a car behind a slower car will likely collide with each other that's why there will be a listener to avoid cars hitting each other this will work by slowing the car behind down.
   Justification 5. - This is needed as if the cars do collide they will just go through each other and will make the game look sloppy and not well designed.

6. The chicken sprite and cars will also have a collision detector in the program the hitboxes will be boxes as it is easier to work with and if they collide a maths question will be generated and asked.
   Justification 6. - The whole purpose of the game is for the player to answer maths questions and with a collision detector the car and chicken sprite will call a function if they collide instead of going through each other.

7. Chicken brings egg to nest egg will follow

   Points go up everytime egg reaches nest

   Music to keep the game intriguing

   Programs asks maths questions when collision occurs

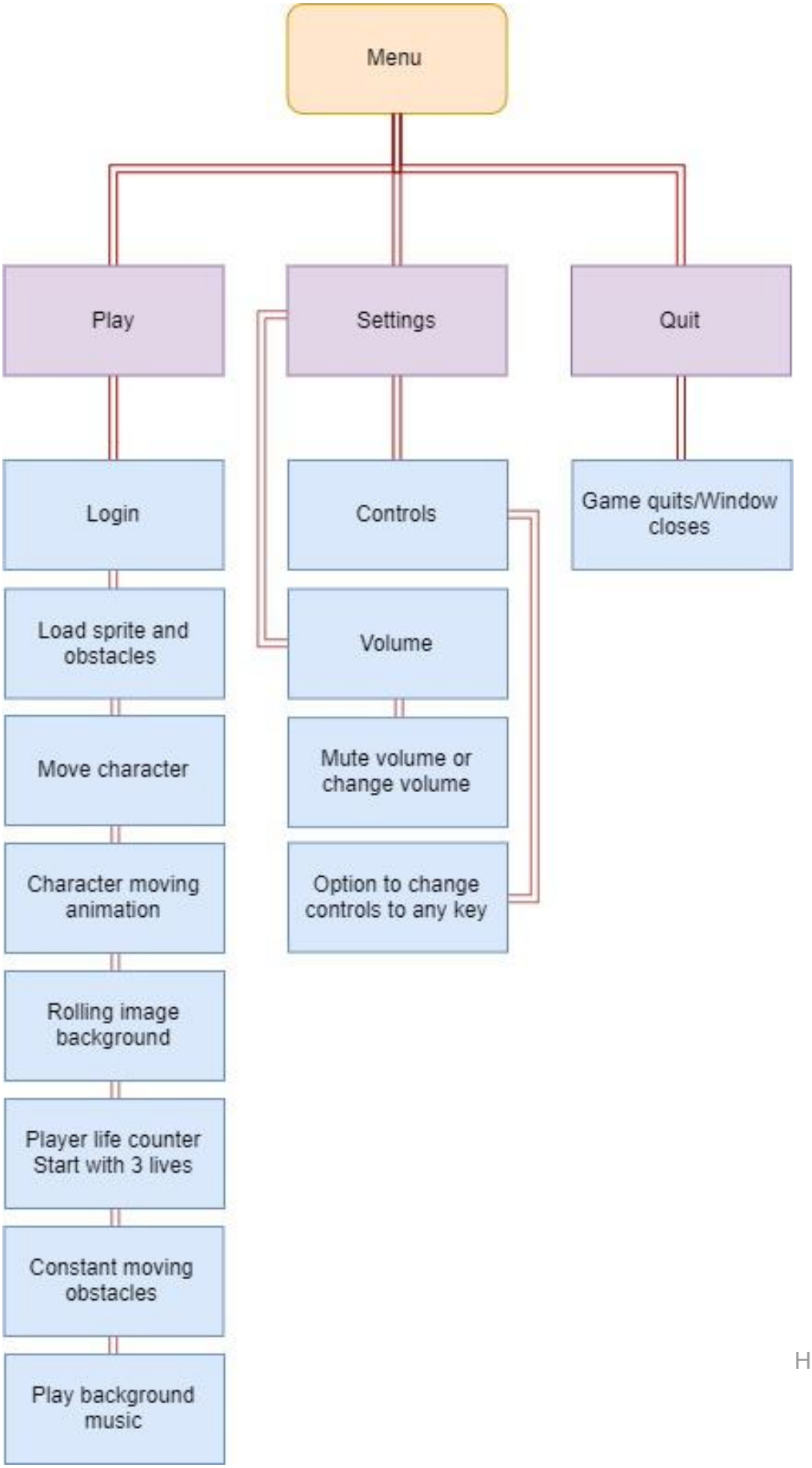
   Leaderboards using sorting algorithm to list top 5 users

   Program will act accordingly depending on the answer by the user to each question answered

   The AI will be put in use to ask question to user using Q Learning and ask a question fit for the user

Design

Decomposition



The decomposition will help me see every detailed steps I need to take when I start programming. This will allow me to work on individual tasks and follow the AGILE testing method.

MENU

- Play
    - Login
        - A login system will become a subroutine in my program this will include a simple SQL database which will use a hashing algorithm to encrypt the passwords. The encryption of passwords is to ensure no one else is able to see other peoples passwords if accessed to the database.

    - Load sprites
        - Sprites will be loaded in, using the pygame commands, as JPEG images which will constantly change as they move to create an animation.

    - Move character
        - A key listener will be used to listen to the keys that are being pressed so that the character moves in the direction of the pressed key. This will also be changeable by the user in the settings.

    - Rolling image background
        - As the game screen will be roughly 500 pixels a rolling image will be good to include to make the game last longer.

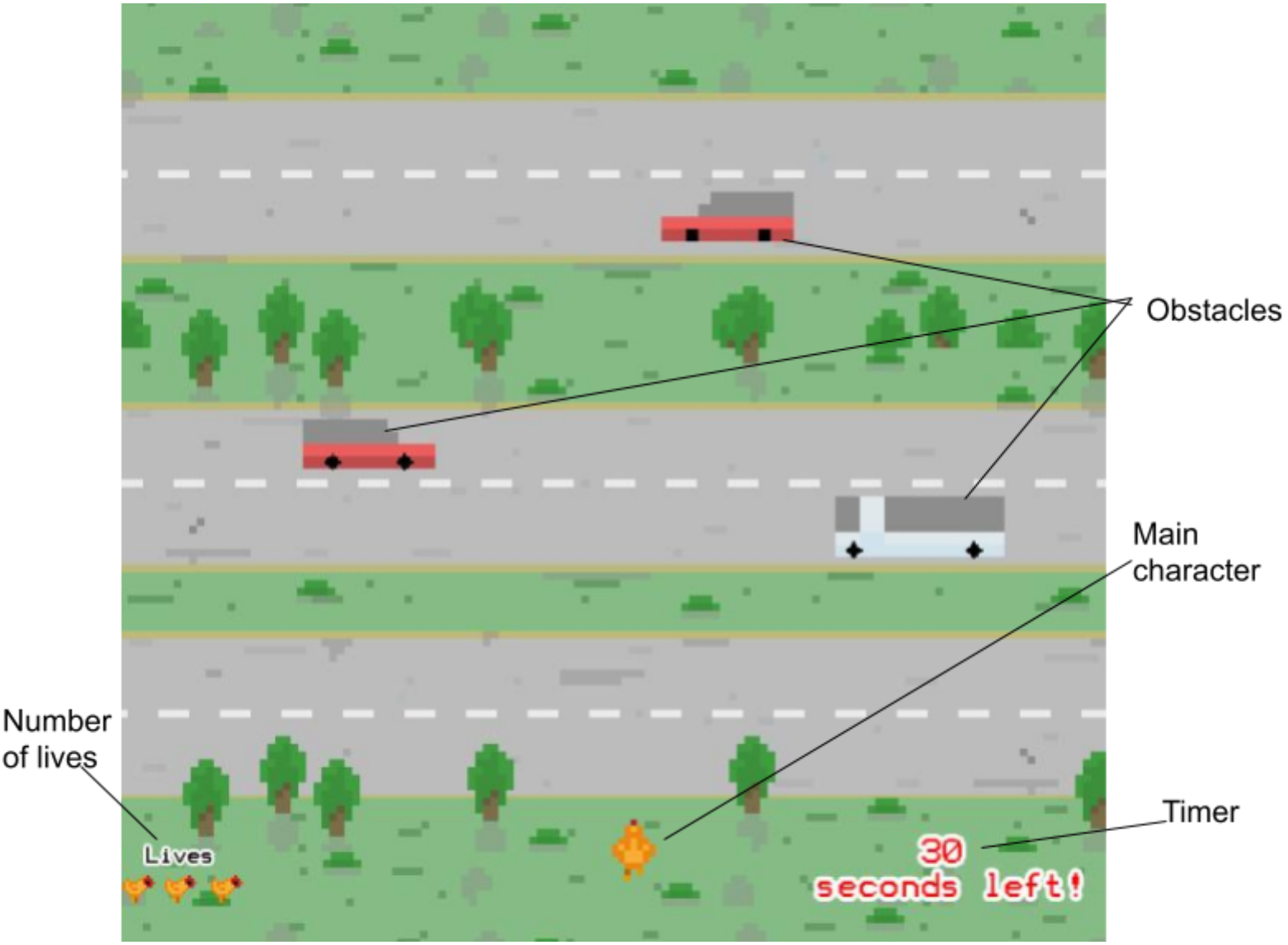
    - Life count listener
        - The user will start off with 3 lives when the game starts giving them 3 chances to complete the level. A life count listener will be included such that; as the user dies the life count will go down by 1 and when they reach 0 lives the game ends.

    - Constant obstacle movement
        - The obstacles will move from left to right making the user plan their moves and try to avoid being hit. The obstacles will have variable speeds depending on what it is e.g. if it is a bus it will be a slower speed.

    - Background music
        - Background music will be useful to include as playing in silent may bore the user by adding simple 8bit background music may make the game fun to play.

- Settings
    - Controls
        - Control feature will be included in the settings so the user can change the default controls to controls suited for them.
    - Volume
        - Volume will also be included so that you don't need to turn your whole system volume down.
- Quit
    - Game quit

Prototype diagram



Usability Features

This will need to be very user-friendly and easy to navigate and use. This shouldn't require a lot of brain power. It is just a quick revision as well as entertainment that you can use to pass time.

For the usability feature I will include many options to benefit the user. The buttons on the program will be very clear and basic to navigate around it will most likely be a 1 word button. The colour of the buttons will also be an indicator of what the button is used for, for example the play button will be green to indicate a "go/start" button and the quit button will be red this will indicate a "stop" or quit button

There will also be a settings feature on the program which will allow players to change the sound or mute. This is useful as they may not want to turn down the volume for the whole computer but having an ingame volume controls feature will not affect other program sounds.

The game will be 2D, simplistic and abstracted. This is so that not much power is used on the game and you won't need a high performance computer to play the game. The reason for this is so that many more users can run the program as modern games require a high end computer which stops some users from playing specific games as their computer cannot run the program. Due to my low power requirements many more users will be able to run this program.

The program will not require immense skill to play the game there are only 4 keys needed which is to control the sprite. As my stakeholder age bracket is 9 - 17 it shouldn't be too complicated for the younger audience likewise it shouldn't be too simple for the older aged stakeholders.

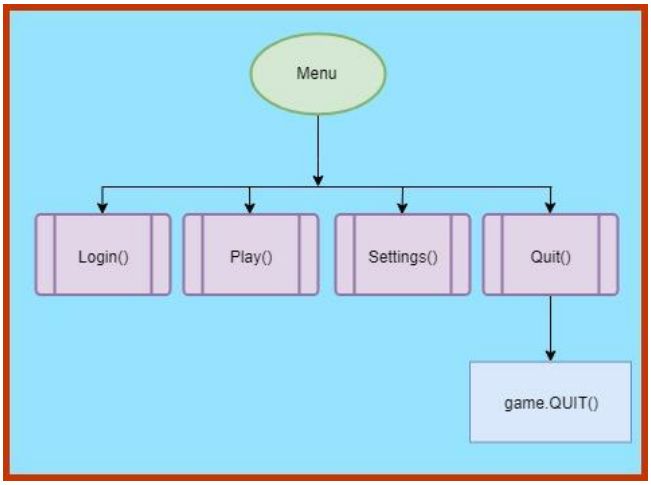
The AI that will be implemented is also a usability feature. As the AI will learn about you and your strength in maths. It will ask questions and as you get them right or wrong it finds your sweet spot and asks you questions from that level. This is useful because 17 year olds wouldn't want to answer questions made for 9 year olds as it would be too easy and 9 year olds wouldn't want to answer questions made for 17 year olds as this would be too hard but that doesn't mean they can't the AI will change questions depending on if you get them right or wrong.

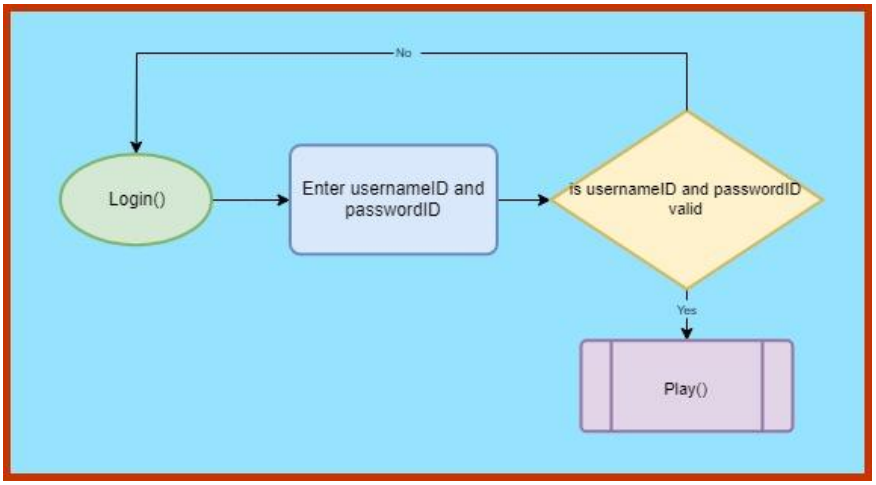
The game will include a tutorial on how to play the game. This will be a usability feature because it helps the user understand and know how to play the game before it starts.

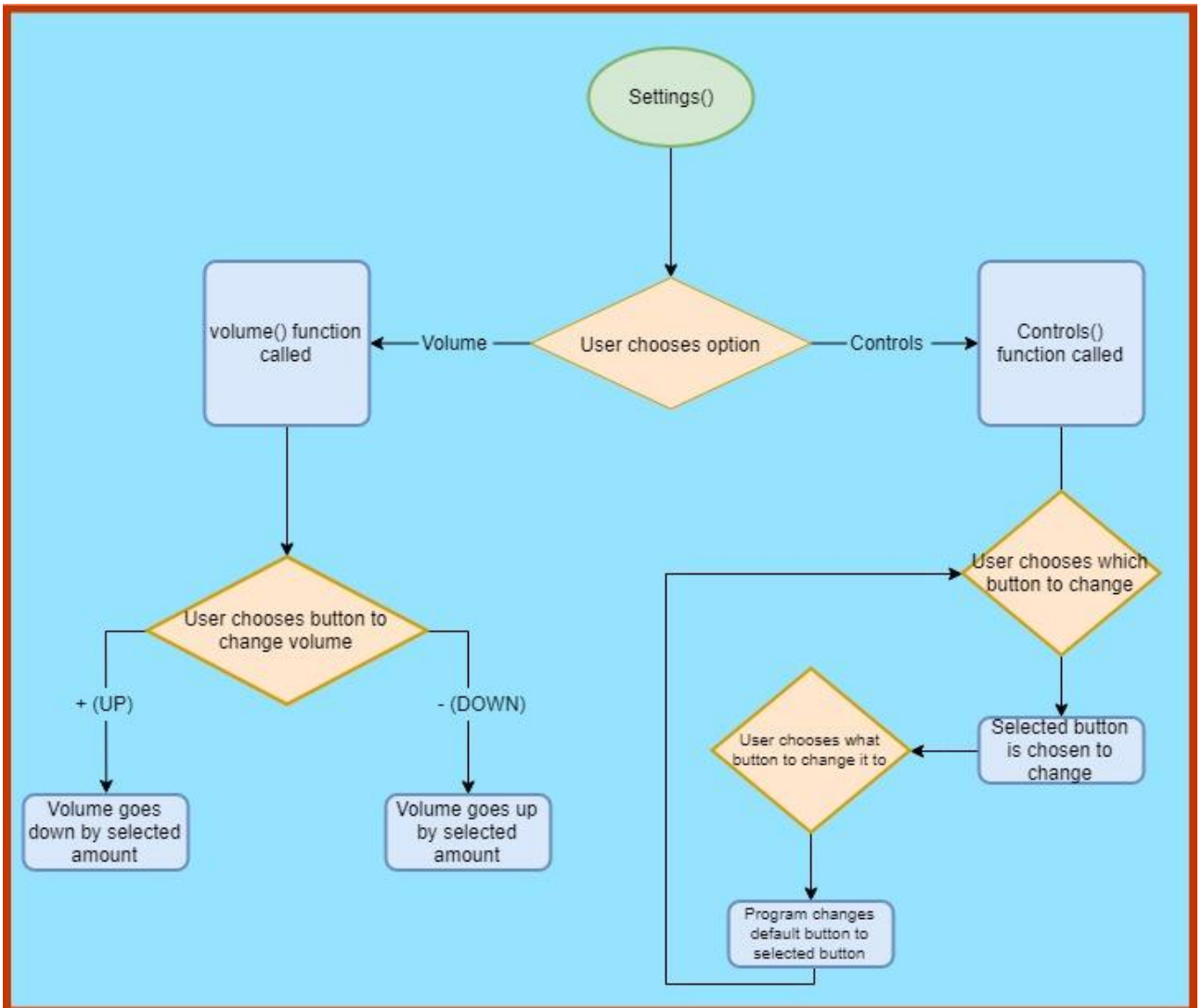
A timer will also be added which will always be there at the bottom of the screen out of the way but still visible to give you a time limit this gives you a goal. This will also be colour dynamic meaning it will change colour depending on what the time is e.g. 10 seconds left will

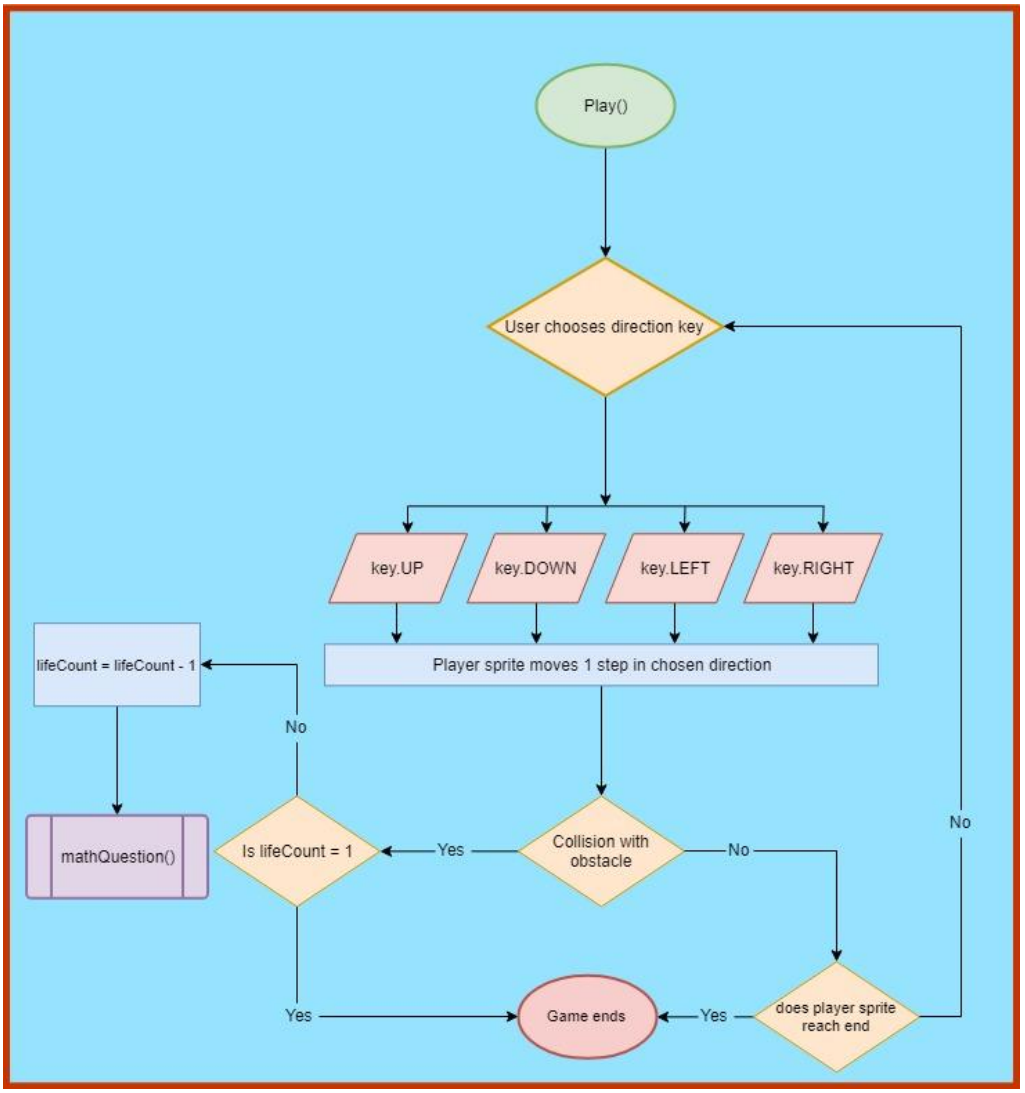
be green, 5 second will be yellow, 3, 2 and 1 will be red so that the user will have an estimate of how long they have left from a peripheral vision.

Abstracted algorithms









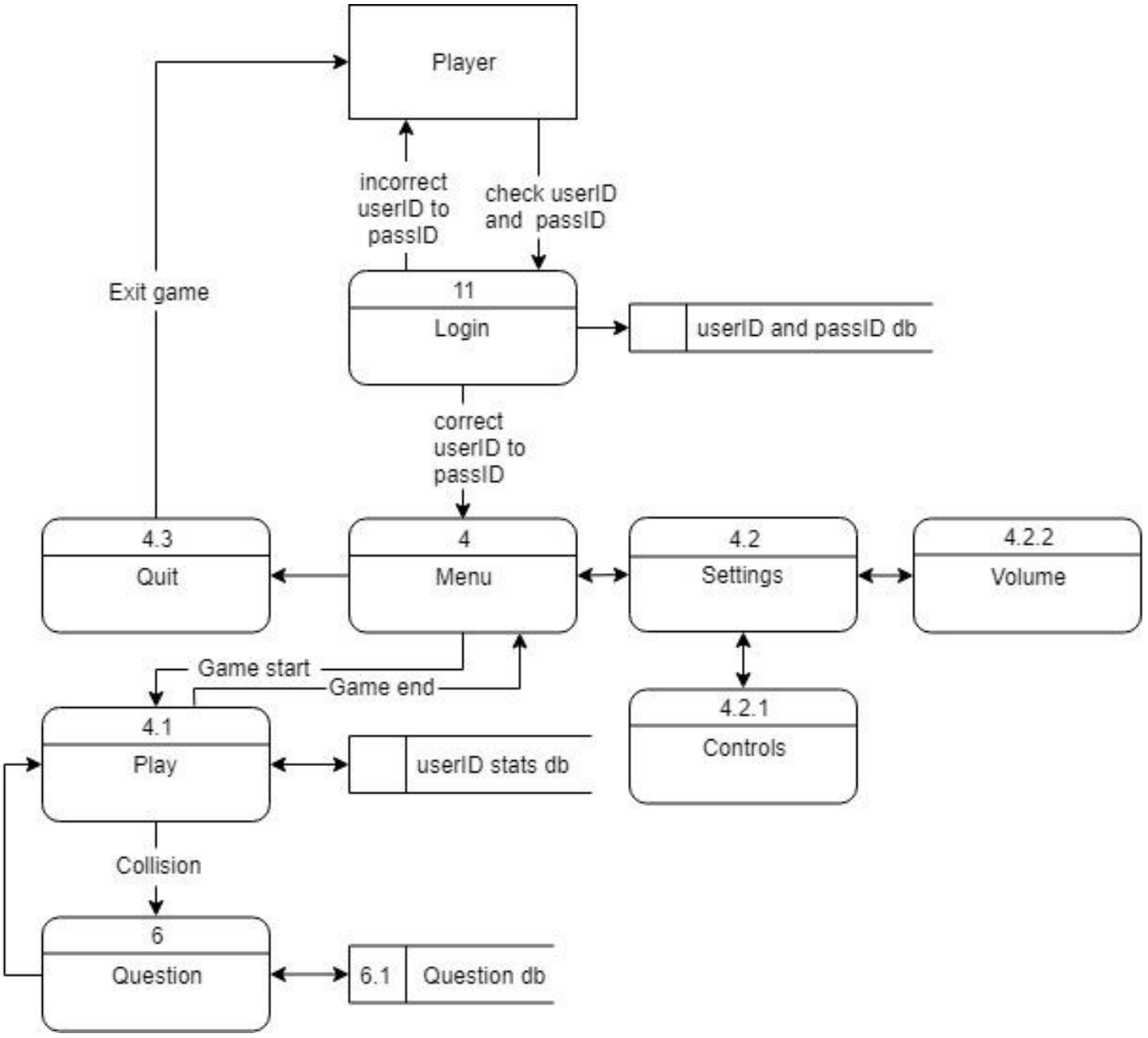
Data Structures

I will have 3-4 databases in my program. The first database will include the questions and answers for the maths question which I will link to my python program. This will most likely be a spreadsheet file. Each question and answer will have a unique ID to link them together.

The second database will include the user's password and username which I will set up an encryption tool to keep them secured. The python program will use a basic SQL system to find the usernames correct password when the user enters the username. This will use a linked list to match the userID to the passwordID in the same record.

I will also need a database to save user's data as they will be able to login and see their progress. This will also be used for the AI as the data for the user will need to be read by the AI to see what level of maths they are on.

Data flow diagram



Key variables

Variable	Data type	Variable name	Description of variable
x	integer	x	This will identify the x position of the objects. This will be useful for movement as the x position will be changing as the player or obstacles are moving horizontally.
y	integer	y	This will identify the y position of the objects. This will be useful for movement as the y position will be changing as the player or obstacles are moving vertically.
velocity	float	vel	Velocity is used to show the speeds of the objects. These will be different depending on the objects e.g a bus will have a smaller velocity whereas a sports car will have a higher velocity.

Left	boolean	left	When the user presses the left key the boolean value of False will change to True activating the left animation and left motion.
Right	boolean	right	When the user presses the right key the boolean value of False will change to True activating the right animation and right motion.
Up	boolean	up	When the user presses the up key the boolean value of False will change to True activating the up animation and up motion.
Down	boolean	down	When the user presses the down key the boolean value of False will change to True activating the down animation and down motion.
Walk count	integer	walkCount	This is needed for the animation of the objects to loop. walkCount is linked to the image e.g. if walkCount = 1 then image[1] will display each object will loop through 3 images therefore when walkCount reaches 3 it will be reset back to 0.

Hitbox	float	hitbox	Hitbox will check if the player has touched the obstacles and uses the x and y values to check if it has hit.
Walking right animation	list	walkRight	Each image will be in a list using walkCount it will go through each element in the list displaying the image.
Walking left animation	list	walkLeft	Each image will be in a list using walkCount it will go through each element in the list displaying the image.
Walking up animation	list	walkUp	Each image will be in a list using walkCount it will go through each element in the list displaying the image.
Walking down animation	list	walkDown	Each image will be in a list using walkCount it will go through each element in the list displaying the image.
Background image		bg	The background image will be behind every object this will be scrolling so most of the background will not be on the screen.

Test Plan

What is being tested	Input	Results expected
Player movement	UP DOWN LEFT RIGHT	Player should move in the inputted direction by 1 step and play the animation depending on the direction being pressed or held.
Player hit	Contact with obstacle	If the player is hit by the obstacles then the game should either restart with 1 life point taken away or the game should end
No collision between other obstacles		Start game multiple times and watch through to see if obstacles hit each other.
Smooth scrolling background image	UP DOWN	Background image should roll smoothly as the player moves forwards or backwards. If the player reaches a barrier the background should not move.
Life counter		When the user dies on their last life the program should call game_lose() function which ends the game as a loss. If the user has more lives the game takes 1 life away from the total amount of lives. The user continues on from where they died with the countdown carrying on from

		where it left off.
Questions function being called when the user dies		When a user loses a life the game will pause as well as the timer and a random question will come up. the user will have 3 options to pick the correct answer.
Questions answered correctly		If the user picks the correct answer then the game will unpause and carry on.
Questions answered wrong		If the user picks the wrong answer then the game will take one life away from the user. Giving the user a maximum of 3 chances to answer the question. If the user answers question wrong on their last life then the game also calling the game_lose() function.
Machine learning AI		The AI will need to find what level of difficulty the user is on. Answering wrong will -1 from the backend point system. The program will ask a question easier than the previous question next time. Answering correctly will +1 to the point system and next time should ask you a harder question.
Sound test		The program will have sound/music. Specific sounds will be played depending on the

		environment e.g. If the obstacle is a car there will be car noises.

Iterative development of a coded solution

For my first development I created a circle that you are able to move around and it changes colour every time you move. My next step from this is to replace it with my sprite where the frame will change as you move.

Justification

The reason for this is so I have an idea of the movement of the object and I can also adjust the frames per second of the game so it isn't too fast or too slow before I change it to my player sprite.

```
import pygame
import random
pygame.init()

win = pygame.display.set_mode((500,500))
pygame.display.set_caption("First Game")

x = 50
y = 440
width = 40
height = 60
vel = 5
r = 255
b = 0
g = 0
run = True
r = random.randint(0,255)
b = random.randint(0,255)
g = random.randint(0,255)
while run:
    pygame.time.delay(20)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT] and x > vel:
        x -= vel
        r = random.randint(0,255)
        b = random.randint(0,255)
        g = random.randint(0,255)

    if keys[pygame.K_RIGHT] and x < 500 - vel - width:
        x += vel
        r = random.randint(0,255)
        b = random.randint(0,255)
        g = random.randint(0,255)

    if keys[pygame.K_UP] and y > vel:
        y -= vel
        r = random.randint(0,255)
        b = random.randint(0,255)
        g = random.randint(0,255)

    if keys[pygame.K_DOWN] and y < 500 - vel - width:
        y += vel
        r = random.randint(0,255)
```

I developed the object by replacing the ball with my sprite image which changes as the player is moving. The in-game error I had with this is that the old image doesn't disappear and did some research on how to fix that problem.

Justification

The reason I added the animation to the player is just for a better design and a visual output of which way the player is moving. I also added if the chicken is idle then it will start to peck at the floor to add some slapstick design to the game.

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT or event.key == ord('a') and x > 0:
        x -= vel
        m1 = pygame.image.load("walk1.png")
        m2 = pygame.image.load("walk12.png")
        m3 = pygame.image.load("walk13.png")
if event.type == pygame.KEYUP:
    time.sleep(n)
    if event.key == pygame.K_LEFT or event.key == ord('a'):
        m1 = pygame.image.load("peck.png")
        m2 = pygame.image.load("peck2.png")
        m3 = pygame.image.load("peck3.png")



if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP or event.key == ord('a'):
        y -= vel
        m1 = pygame.image.load("fr.png")
        m2 = pygame.image.load("fr2.png")
        m3 = pygame.image.load("fr3.png")
if event.type == pygame.KEYUP:
    time.sleep(n)
    if event.key == pygame.K_UP or event.key == ord('a'):
        m1 = pygame.image.load("frs.png")
        m2 = pygame.image.load("frs2.png")
        m3 = pygame.image.load("frs3.png")
```

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_DOWN or event.key == ord('a'):
        y += vel
        m1 = pygame.image.load("back.png")
        m2 = pygame.image.load("back2.png")
        m3 = pygame.image.load("back3.png")
if event.type == pygame.KEYUP:
    time.sleep(n)
    if event.key == pygame.K_DOWN or event.key == ord('a'):
        m1 = pygame.image.load("peckf.png")
        m2 = pygame.image.load("peckf2.png")
        m3 = pygame.image.load("peckf3.png")


pygame.display.update()
clock.tick(10)
```

I optimised the stored variables of the player by creating a class so the attributes are easily changeable and better programming for object orientated programming. I also removed the

pecking animation as after some testing it did get repetitive and did get slightly annoying from feedback from peers and also suggested that I make this change.

Justification

The reason for the classes for the object is due to easier encapsulation of variables and data only needed for the chicken. I am able to create instances and also create initial values for the player which can change during run time.

```
#player class
class player(object):
    #initial variables of class assigned at instance
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.vel = 5
        self.left = False
        self.right = False
        self.up = False
        self.down = False
        self.walkCount = 0
        self.hitbox = (self.x, self.y, 36, 44)

    #changing image of player whilst holding key
    def draw(self,win):
        if self.walkCount + 1 >= 27:
            self.walkCount = 0

        if self.left:
            win.blit(walkLeft[self.walkCount//3], (self.x,self.y))
            self.walkCount += 1

        elif self.right:
            win.blit(walkRight[self.walkCount//3], (self.x,self.y))
            self.walkCount +=1

        elif self.up:
            win.blit(walkUp[self.walkCount//3], (self.x,self.y))
            self.walkCount +=1

        elif self.down:
            win.blit(walkDown[self.walkCount//3], (self.x,self.y))
            self.walkCount +=1

        else:
            win.blit(char, (self.x,self.y))
        self.hitbox = (self.x, self.y, 36, 44)
        pygame.draw.rect(win, (255,0,0), self.hitbox,2)
```

As each key is pressed game will start looping through the frames of the sprite

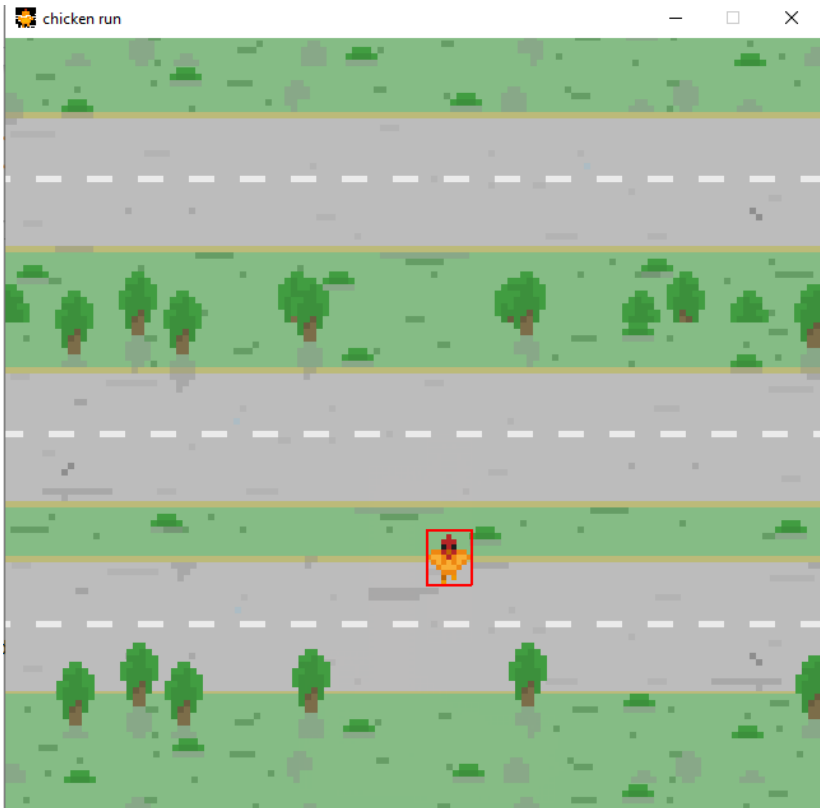
```
        k = pygame.key.get_pressed()
        #if key left pressed, left frames loop runs
        if k[pygame.K_LEFT] and chicken.x > 0:
            chicken.x -= chicken.vel
            print(chicken.x)
            chicken.left = True
            chicken.right = False
            chicken.down = False
            chicken.up = False

        #if key right pressed, right frames loop runs
        elif k[pygame.K_RIGHT] and chicken.x < (winx - chicken.width):
            chicken.x += chicken.vel
            print(chicken.x)
            chicken.left = False
            chicken.right = True
            chicken.down = False
            chicken.up = False
        #if key up pressed, up frames loop runs
        elif k[pygame.K_UP] and chicken.y > 0:
            chicken.y -=chicken. vel
            print(chicken.y)
            chicken.right = False
            chicken.left = False
            chicken.down = False
            chicken.up = True
        #if key up pressed, down frames loop runs
        elif k[pygame.K_DOWN] and chicken.y < (winy - chicken.height):
            chicken.y += chicken.vel
            print(chicken.y)
            chicken.right = False
            chicken.left = False
            chicken.up = False
            chicken.down = True

        elif k[pygame.K_ESCAPE]:
            run = False

        else:
            chicken.right = False
            chicken.left = False
            chicken.down = False
            chicken.up = False
            chicken.walkCount = 0
```

As key is pressed each boolean value changes and called in the class

I created a red box that follows the player and keeps the player inside the box. This was placed inside the player class so that when instances is created the box will change depending on the parameters

Justification

The reason for this is for me to see if the chicken sprite sticks out of the box and adjust the box this will also be useful when I am creating the hitbox detection algorithm using a basic polygon will be better for hitbox detection as i only need to worry about the x and y planes



I created a class for the car and setup so that it moves from left to right and when it reaches one end it spawns randomly on another lane but only able to spawn one car at a time the cars also have the same velocity which I also want to change.

Justification

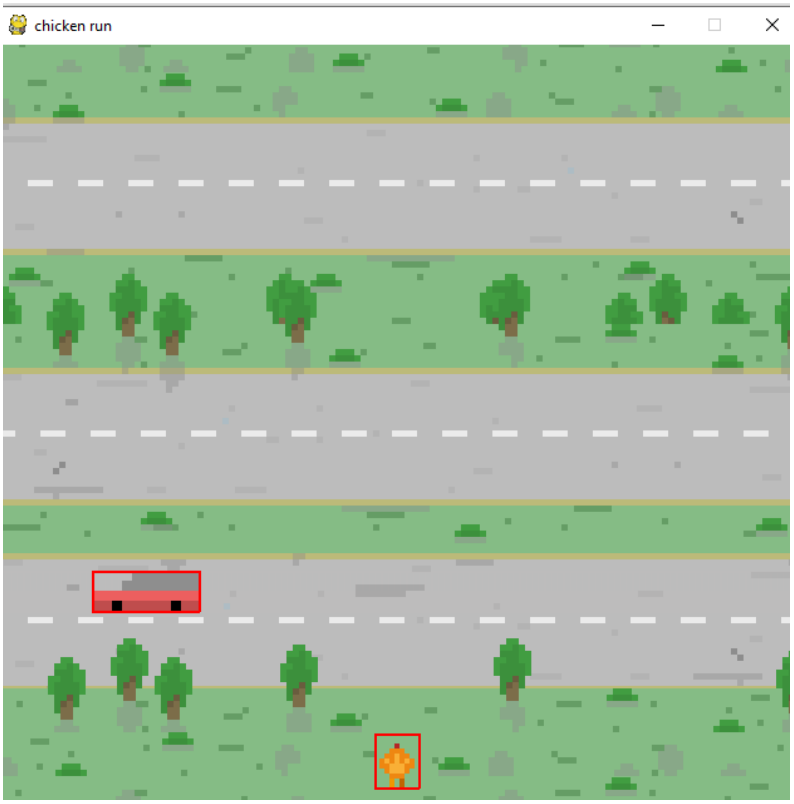
The class is used so that each car image can inherit this superclass but the only thing that it will change is the look of the car. I will also make the cars have a random velocity this will give it a more realistic and better feel of the game.

```
File  Edit  Format  Run  Options  Window  Help
#car class
class car(object):
    walkRight = [pygame.image.load('carright1.png'), pygame.image.load('carr
    walkLeft = [pygame.image.load('carleft1.png'), pygame.image.load('carlef
    def __init__(self, x, y, width, height, end):
        self.x = x
        self.y = y
        self.width = width
        self.end = end
        self.height = height
        self.path = [self.x, self.end]
        self.vel = 7
        self.walkCount = 0
        self.hitbox = (self.x, self.y, 86, 33)
    #changing image as car moves
    def draw(self, win):
        self.move()
        if self.walkCount + 1 >= 9:
            self.walkCount = 0

        if self.vel > 0:
            win.blit(self.walkRight[self.walkCount //3], (self.x, self.y))
            self.walkCount += 1
        else:
            win.blit(self.walkLeft[self.walkCount //3], (self.x, self.y))
            self.walkCount += 1
        self.hitbox = (self.x, self.y, 86, 33)
        pygame.draw.rect(win, (255,0,0), self.hitbox,2)


    #if car reaches end from right to left it will spawn left to right
    def move(self):
        if self.vel > 0:
            if self.x + self.vel < self.path[1]:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.walkCount = 0
        else:
            if self.x - self.vel > self.path[0]:
                self.x += self.vel
            else:
                self.vel = self.vel * -1
                self.walkCount = 0
```

Initial object values passed by value at the instance

Change image of car once it changes sides from where it spawns

Change velocity from + to - when reaching end of side



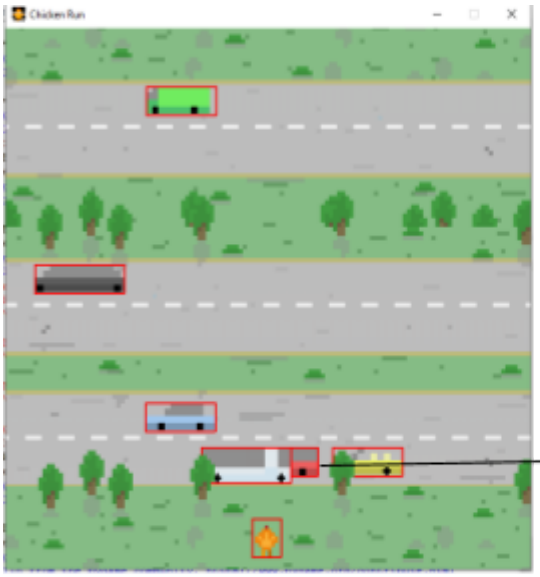
Created multiple classes which inherit from the car class but changed the images to the intended car images of the car. The velocities of the cars are all still the same and the cars

as they spawn will all come from the same side this will also need to change so that it is random.

Justification

All the cars spawn from the same side and this will make the game less realistic furthermore the cars all spawn in every lane and I want to make it that they follow the UK lane code which is driving on the left hand side of the road and will implement this later on.

```
#All sprites loading function
def spriteLOAD():
    win.blit(bg, (0,0))
    chicken.draw(win)
    carList = [RedCar, WhiteBus, BlueCar, YellowConv, GreenVan, WhiteVan, BlackLimo, YellowBus]
    #List of cars to be called
    carGen = [carList[0].draw(win),carList[1].draw(win), carList[2].draw(win),carList[3].draw(win), carL

    carGen
    win.blit(overlay, (0,0))
    #if RedCar.x == 30:
        #print("New Gen")
        #carGen2
    pygame.display.update()
```



Cars hitting each other usually once or twice each game could be avoidable but finding a solution for this will make my game more professional

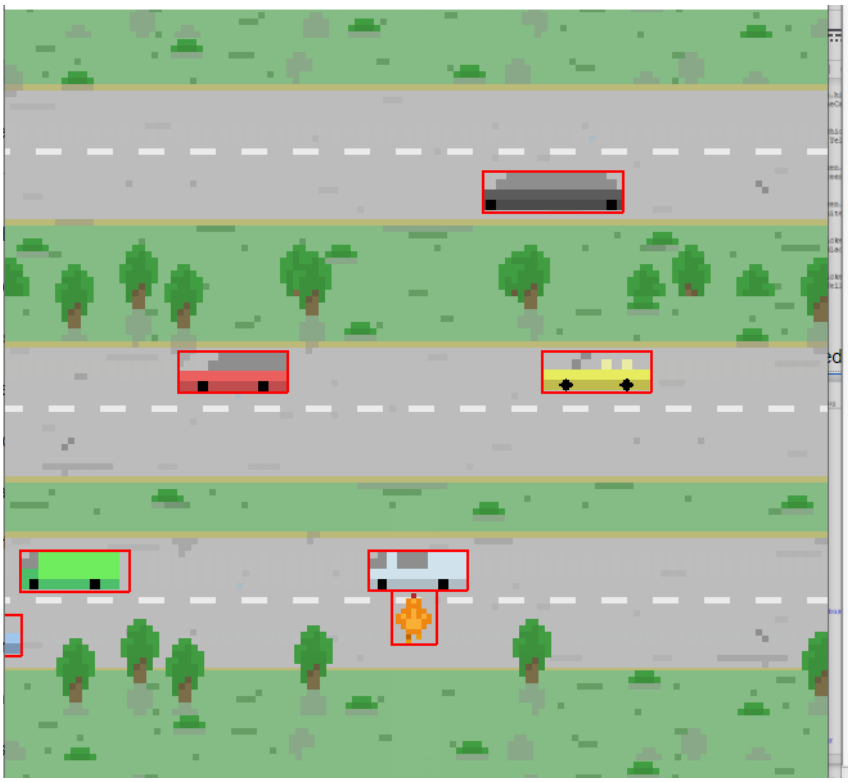
Creating this hitbox detection for every class with the chicken is tedious and in the back of my head I can tell there is a much more code efficient way of doing this but for now I can stick with this until I find how to solve that problem.

Justification

Although this hitbox detection with the chicken and every car works it makes the code messy and inefficient I will find a way to find a more optimised code for this section.

```
#road coordinates = 65, 125, 325, 265, 420, 470
run = True
while run:
    clock.tick(27)

    if chicken.hitbox[1] < RedCar.hitbox[1] + RedCar.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > RedCar.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > RedCar.hitbox[0] and chicken.hitbox[0] < RedCar.hitbox[0] + RedCar.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < WhiteBus.hitbox[1] + WhiteBus.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > WhiteBus.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > WhiteBus.hitbox[0] and chicken.hitbox[0] < WhiteBus.hitbox[0] + WhiteBus.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < BlueCar.hitbox[1] + BlueCar.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > BlueCar.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > BlueCar.hitbox[0] and chicken.hitbox[0] < BlueCar.hitbox[0] + BlueCar.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < YellowConv.hitbox[1] + YellowConv.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > YellowConv.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > YellowConv.hitbox[0] and chicken.hitbox[0] < YellowConv.hitbox[0] + YellowConv.hitbox[2]
            chicken.collision()

    if chicken.hitbox[1] < GreenVan.hitbox[1] + GreenVan.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > GreenVan.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > GreenVan.hitbox[0] and chicken.hitbox[0] < GreenVan.hitbox[0] + GreenVan.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < WhiteVan.hitbox[1] + WhiteVan.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > WhiteVan.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > WhiteVan.hitbox[0] and chicken.hitbox[0] < WhiteVan.hitbox[0] + WhiteVan.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < BlackLimo.hitbox[1] + BlackLimo.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > BlackLimo.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > BlackLimo.hitbox[0] and chicken.hitbox[0] < BlackLimo.hitbox[0] + BlackLimo.hitbox[2]:
            chicken.collision()

    if chicken.hitbox[1] < YellowBus.hitbox[1] + YellowBus.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > YellowBus.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > YellowBus.hitbox[0] and chicken.hitbox[0] < YellowBus.hitbox[0] + YellowBus.hitbox[2]:
            chicken.collision()
```

Very inefficient I will research using books on how to use code optimisation



```
540
535
530
525
520
515
510
505
500
495
490
485
hit
Hit with blue car
545
540
535
530
525
520
515
510
505
500
495
490
485
480
475
470
465
460
455
450
hit
Hit with white van car
```

I created a main menu screen which will allow the player to navigate around the program from player to settings. They will view this screen after they log in.

Justification

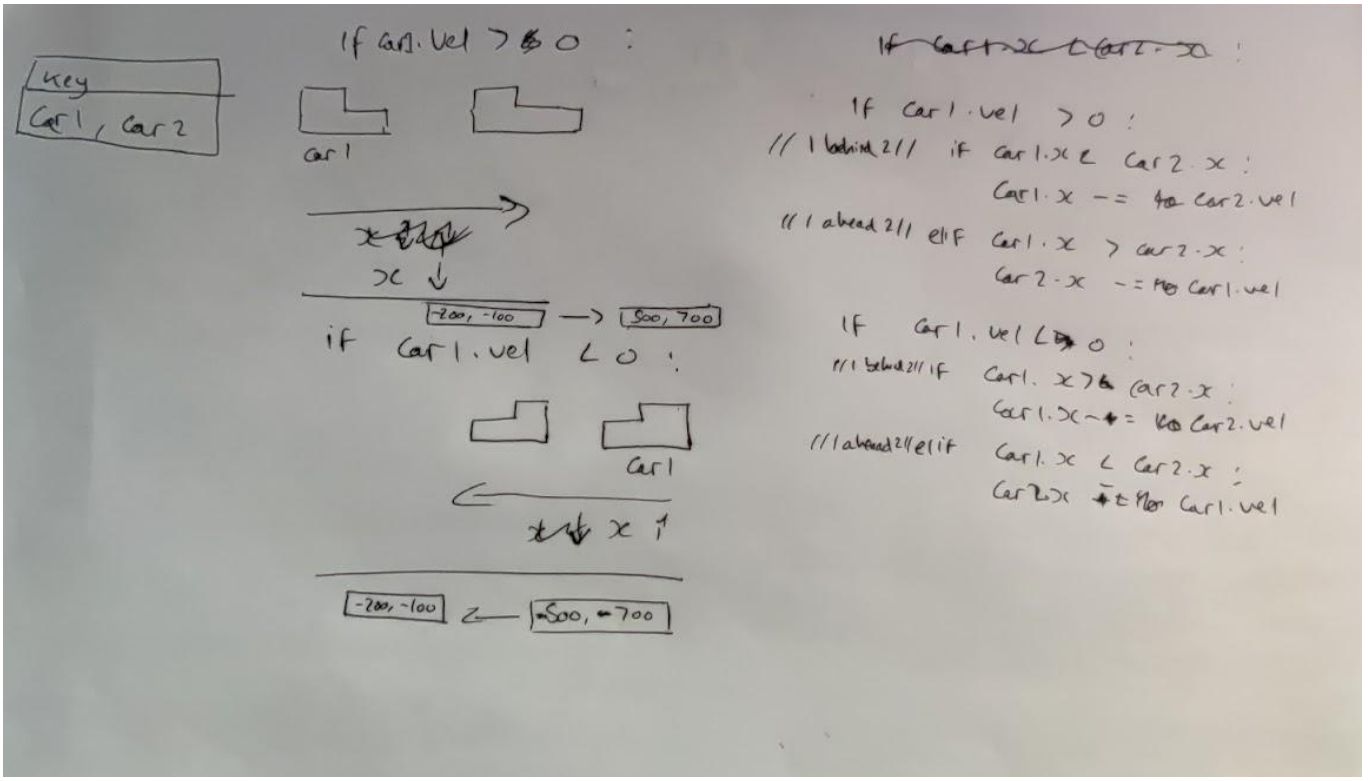
The reason for this is so that the game has a sense of professionalism like actual games that are out there. The background also scrolls downwards giving it an arcade style theme.

Optimised the collision between the chicken and each car by making it continuously scroll through every car checking if it hits the player.

Justification

This was very useful to me by figuring this out. I had a plan on how to make the cars stop hitting each other also and the code optimisation made me reduce the code from 27 lines to 5 lines.

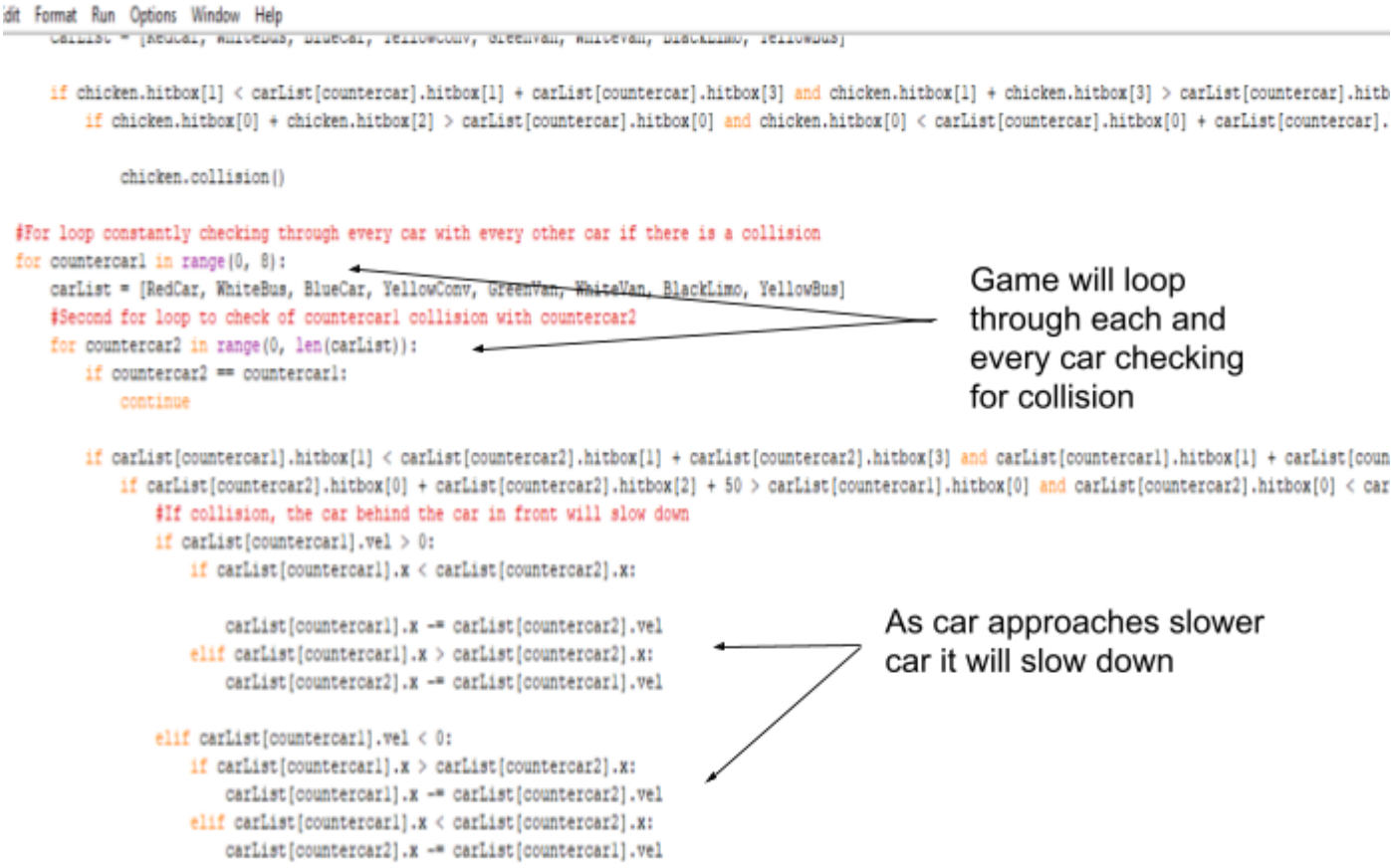
```
#loop going through every car checking if it has hit the chicken
for countercar in range(0,8):
    #list of every car
    carList = [RedCar, WhiteBus, BlueCar, YellowConv, GreenVan, WhiteVan, BlackLimo, YellowBus]
    #if statement checking if the x and y values of the 2 entities are the same
    if chicken.hitbox[1] < carList[countercar].hitbox[1] + carList[countercar].hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > carList[countercar].hitbox[1]
        if chicken.hitbox[0] + chicken.hitbox[2] > carList[countercar].hitbox[0] and chicken.hitbox[0] < carList[countercar].hitbox[0] + carList[countercar].hitbo

            chicken.collision()
```



Using my inspiration from the car and chicken hitbox detection I decided to approach the car to car hitbox detection with the same method by adding a for loop but instead it will detect when the cars are near each other by 50 pixels instead of touching and when they are near the car behind will slow down. I was confused on how I will make the game constantly check for cars hitting each other. I decided to draw it to get a visual idea and came up with an algorithm and implemented it into my code.

Justification

I was able to randomise the velocity by changing it to a fixed range of values it can choose from now that if a faster car is behind a slower one it is able to also slow down using this algorithm.



I added a life counter to the game which will reduce by 1 every time the chicken collides with a car. At the same time I added the egg and nest they will spawn in random x locations making it more interactive. Once the chicken reaches the egg it'll follow the chicken until it reaches the nest.

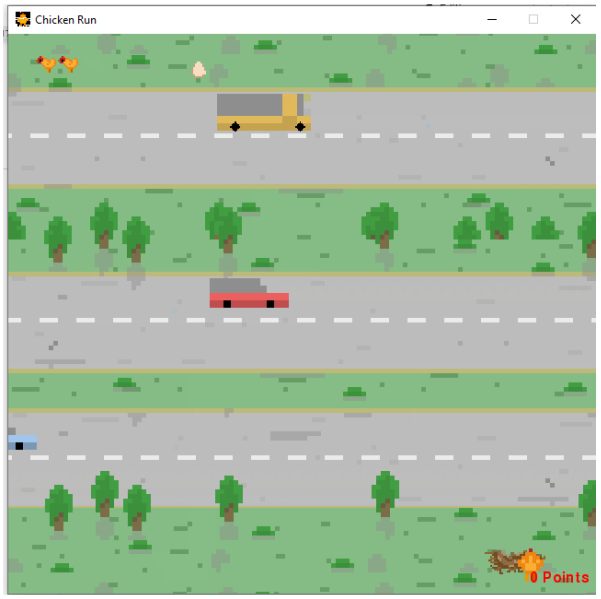
Justification

This will give the game purpose instead of just having the chicken reach the end. It'll make the game more competitive by letting players try to get a higher score each time as well as asking maths questions once they collide. The life counter will add some intensity to the game after each collision the life score will go down and also if not added the game will be never ending defeating the whole purpose of a score.

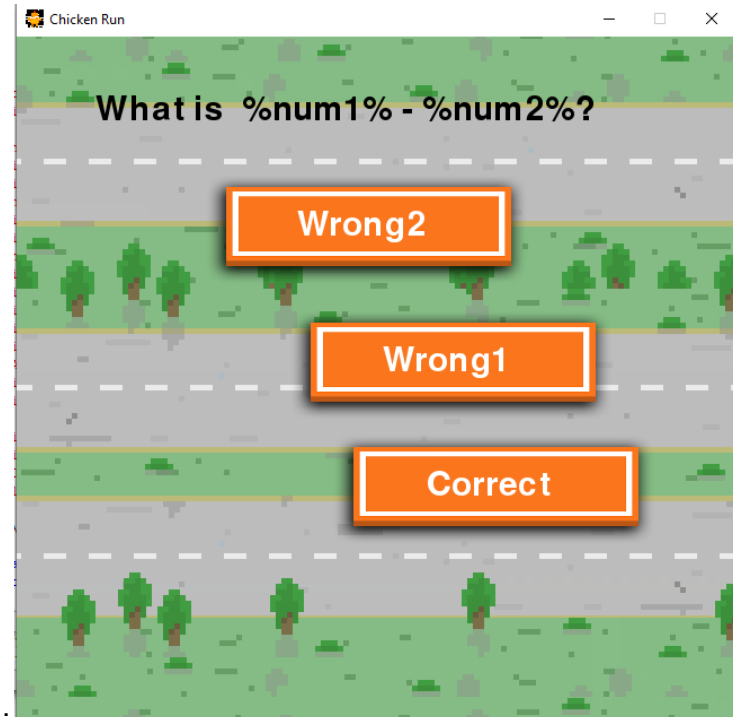
```
class items():
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height
        self.hitbox = (self.x, self.y, self.width, self.height)

    def draw(self, win):
        win.blit(self.frame, (self.x,self.y))
        self.hitbox = (self.x, self.y, self.width, self.height)


class egg(items):
    frame = eggIMG


class nest(items):
    frame = nestIMG
```

```
if chicken.hitbox[1] < eggsprite.hitbox[1] + eggsprite.hitbox[3] and chicken.hitbox[1] + chicken.hitbox[3] > eggsprite.hitbox[1]:
        if chicken.hitbox[0] + chicken.hitbox[2] > eggsprite.hitbox[0] and chicken.hitbox[0] < eggsprite.hitbox[0] + eggsprite.hitbox[2]:
            eggsprite.x = chicken.x + 10
            eggsprite.y = chicken.y + 10

if nestsprite.hitbox[1] < eggsprite.hitbox[1] + eggsprite.hitbox[3] and nestsprite.hitbox[1] + nestsprite.hitbox[3] > eggsprite.hitbox[1]:
        if nestsprite.hitbox[0] + nestsprite.hitbox[2] > eggsprite.hitbox[0] and nestsprite.hitbox[0] < eggsprite.hitbox[0] + eggsprite.hitbox[2]:
            pointcount.point += 1
            eggsprite.x = random.randint(10,650)
            eggsprite.y = 30
```

```
    def collision(self):
        time.sleep(1)
        self.QuestionsAI()
        self.lives -= 1
        self.x = 550
        self.y = 550

        if self.lives == 2:
            self.life1 = False


        elif self.lives == 1:
            self.life2 = False


        elif self.lives == 0:
            self.life3 = False
```



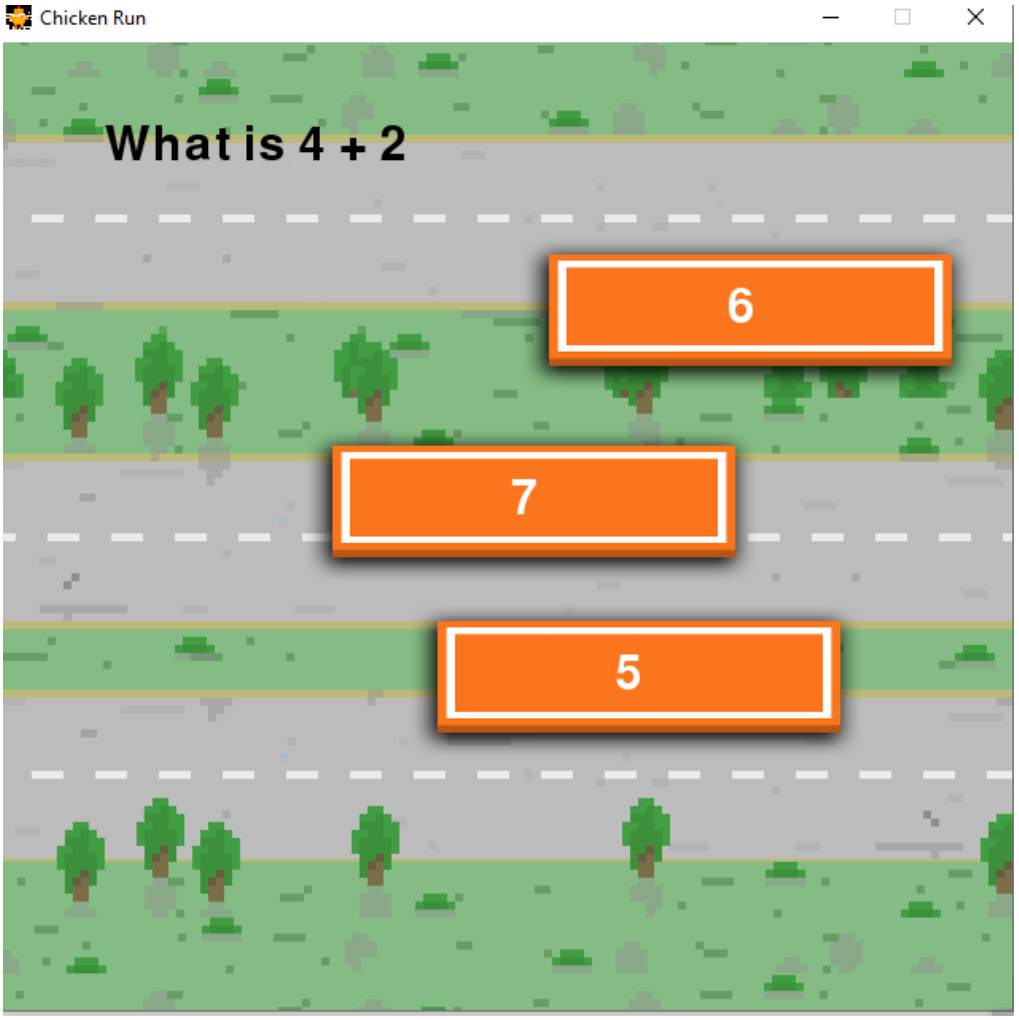
Question GUI when collision occurs if question right or wrong data will be stored permanently in the SQL database

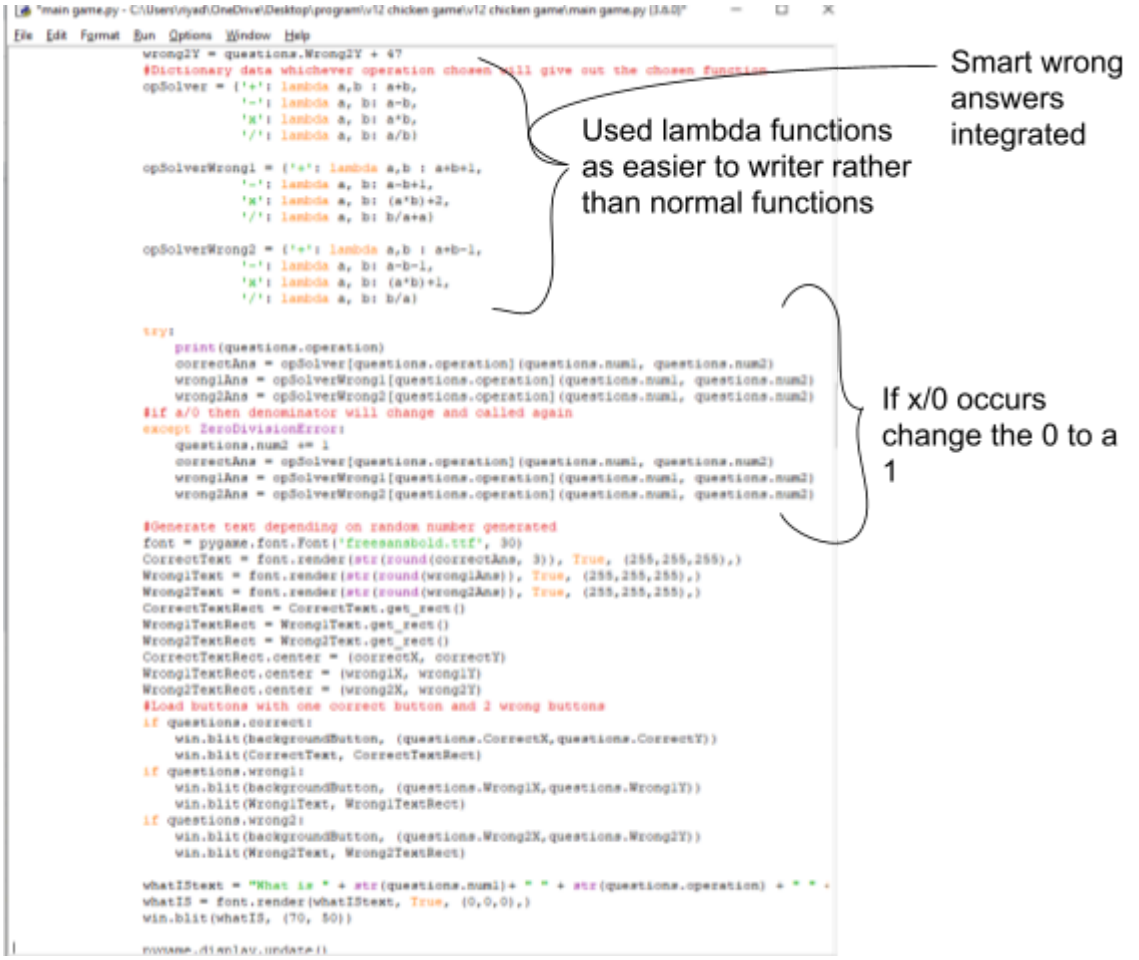

Chicken Run
What is %num1% - %num2%?
Wrong2
Wrong1
Correct

File Edit Format Run Options Window Help

```
    def QuestionsAI(self):
        backgroundButton = pygame.transform.scale(pygame.image.load('backgro
        backgroundButtonTest = pygame.transform.scale(pygame.image.load('pla
        list1 = [120, 240, 350]

        pygame.display.update()
        class questionsPicker():
            def __init__(self, level):

                self.level = level
                self.wrong1 = True
                self.wrong2 = True
                self.correct = True

                self.CorrectX = random.randint(0,330)
                self.Wrong1X = random.randint(0,330)
                self.Wrong2X = random.randint(0,330)

                self.CorrectY = random.choice(list1)
                list1.remove(self.CorrectY)

                self.Wrong1Y = random.choice(list1)
                list1.remove(self.Wrong1Y)

                self.Wrong2Y = random.choice(list1)
                self.ListY = [120, 240, 350]


                self.correctHitbox = pygame.Rect((self.CorrectX,self.Correct
                self.Wrong1Hitbox = pygame.Rect((self.Wrong1X,self.Wrong1Y),
                self.Wrong2Hitbox = pygame.Rect((self.Wrong2X,self.Wrong2Y),

            def Collision(self):
                self.CorrectX = random.randint(0,330)
                self.Wrong1X = random.randint(0,330)
                self.Wrong2X = random.randint(0,330)
                self.CorrectY = random.choice(list1)
                list1.remove(self.CorrectY)
                self.Wrong1Y = random.choice(list1)
                list1.remove(self.Wrong1Y)
                self.Wrong2Y = random.choice(list1)
                self.List1 = [120, 240, 350]

                self.correctHitbox = pygame.Rect((self.CorrectX,self.Correct
                self.Wrong1Hitbox = pygame.Rect((self.Wrong1X,self.Wrong1Y),
                self.Wrong2Hitbox = pygame.Rect((self.Wrong2X,self.Wrong2Y),
                return

            def textGen(self):
                X  = CorrectX + 150
                self.font = pygame.font.Font('freesansbold.ttf', 16)
                self.text = self.font.render("Test", True, (0,0,0),)
                self.textRect = self.text.get_rect()
                self.textRect.center = (X, Y)
                win.blit(self.text, self.textRect)

        def spriteGen():
            win.blit(bg, (0,0))
```

Ln: 21 Col:


Chicken Run
What is 4 + 2
6
7
5

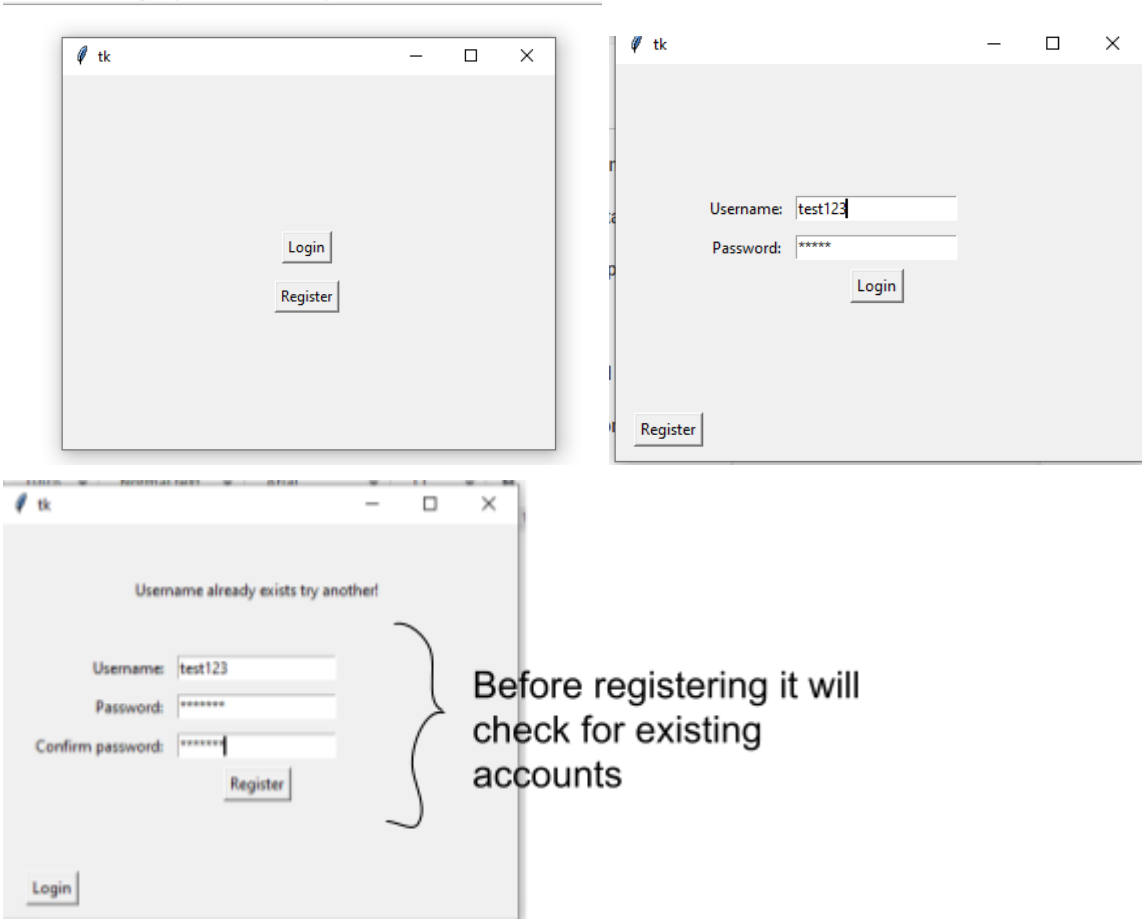


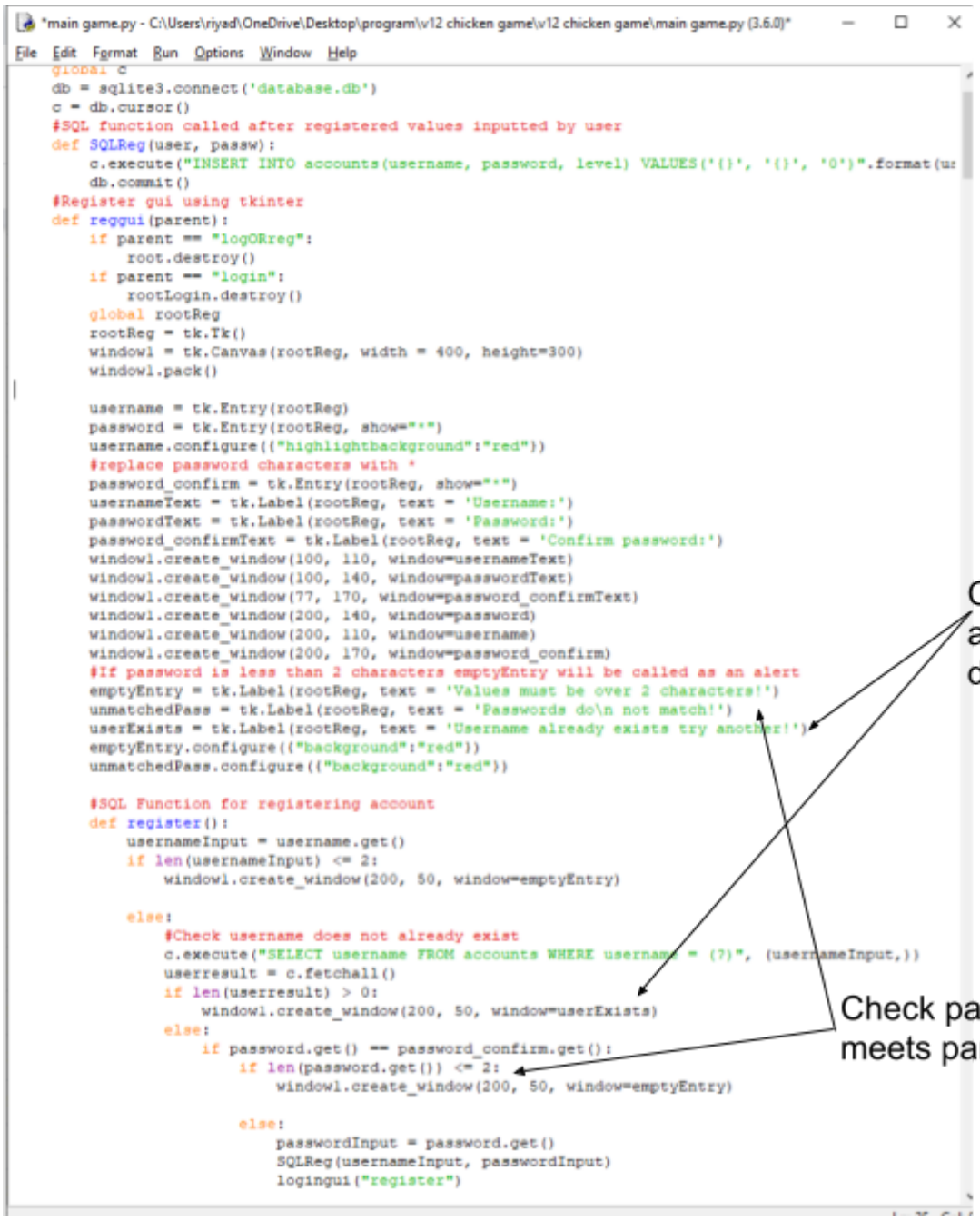
If number / 0 occurs then error will come up this was solved using a exception which will change the 0 to a 1 instead

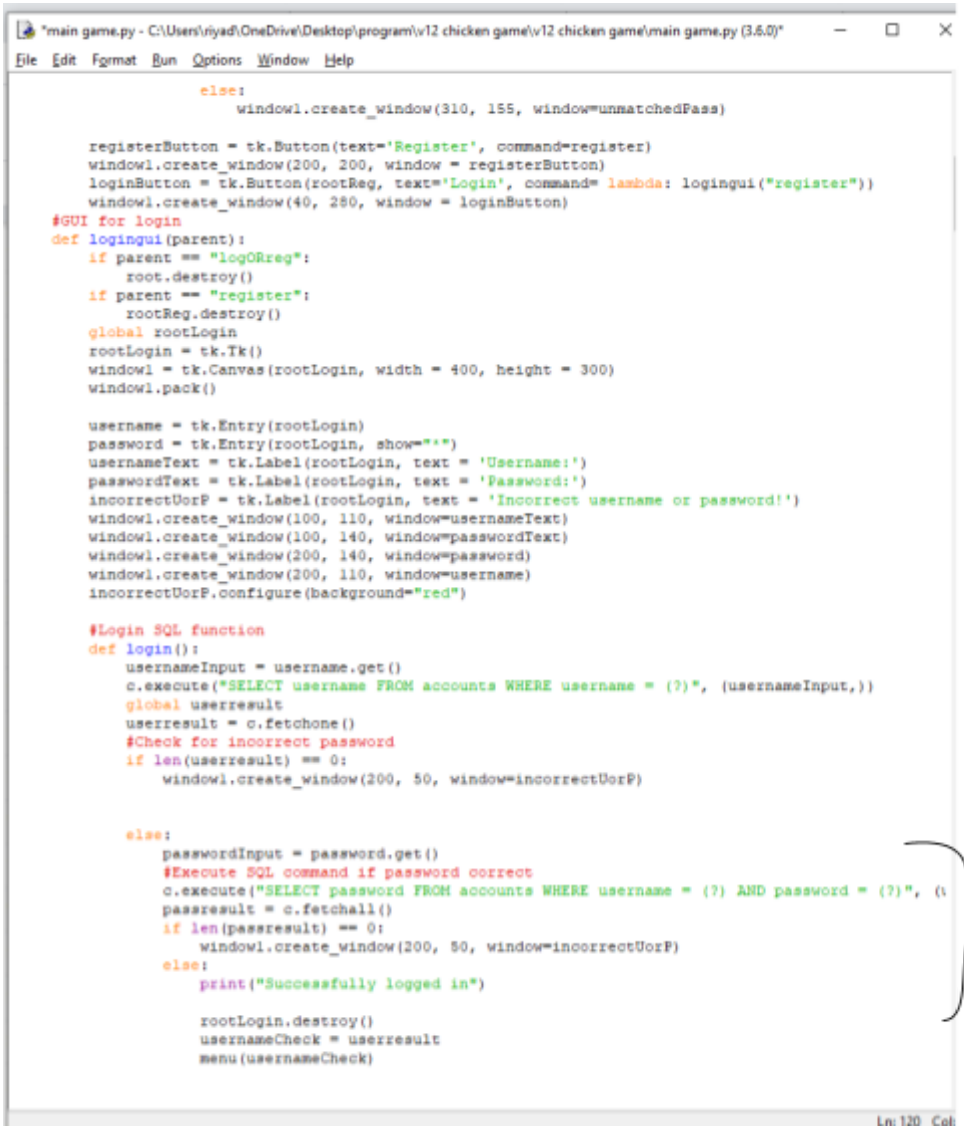
```
  File "C:\Users\riyad\OneDrive\Desktop\program\v12 chicken game\v12 chicken gam
e\main game.py", line 368, in spriteGen
    correctAns = opSolver[questions.operation](questions.num1, questions.num2)
  File "C:\Users\riyad\OneDrive\Desktop\program\v12 chicken game\v12 chicken gam
e\main game.py", line 366, in <lambda>
    '/': lambda a, b: a/b}
ZeroDivisionError: division by zero
```

```
            try:
                print(questions.operation)
                correctAns = opSolver[questions.operation](questions.num1, questions.num2)
                wrong1Ans = opSolverWrong1[questions.operation](questions.num1, questions.num2)
                wrong2Ans = opSolverWrong2[questions.operation](questions.num1, questions.num2)
            #if a/0 then denominator will change and called again
            except ZeroDivisionError:
                questions.num2 += 1
                correctAns = opSolver[questions.operation](questions.num1, questions.num2)
                wrong1Ans = opSolverWrong1[questions.operation](questions.num1, questions.num2)
                wrong2Ans = opSolverWrong2[questions.operation](questions.num1, questions.num2)
```

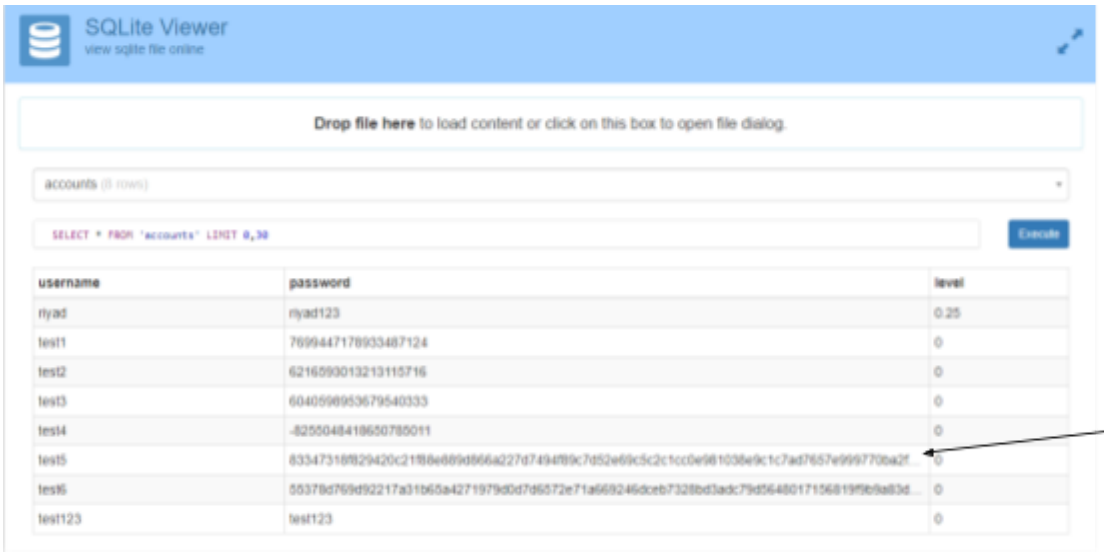
Login system - Login system works and need to close window after logging in and registering.
SQL database - SQL database created with fields username as the primary key and password and score.




main game.py - C:\Users\riyad\OneDrive\Desktop\program\v12 chicken game\v12 chicken game\main game.py (3.6.0)
File Edit Format Run Options Window Help
global c
db = sqlite3.connect('database.db')
c = db.cursor()
#SQL function called after registered values inputted by user
def SQLReg(user, passw):
c.execute("INSERT INTO accounts(username, password, level) VALUES('{}', '{}', '0')".format(us
db.commit()
#Register gui using tkinter
def reggui(parent):
if parent == "logORreg":
root.destroy()
if parent == "login":
rootLogin.destroy()
global rootReg
rootReg = tk.Tk()
window1 = tk.Canvas(rootReg, width = 400, height=300)
window1.pack()
username = tk.Entry(rootReg)
password = tk.Entry(rootReg, show="*")
username.configure({"highlightbackground":"red"})
#replace password characters with *
password_confirm = tk.Entry(rootReg, show="*")
usernameText = tk.Label(rootReg, text = 'Username:')
passwordText = tk.Label(rootReg, text = 'Password:')
password_confirmText = tk.Label(rootReg, text = 'Confirm password:')
window1.create_window(100, 110, window=usernameText)
window1.create_window(100, 140, window=passwordText)
window1.create_window(77, 170, window=password_confirmText)
window1.create_window(200, 140, window=password)
window1.create_window(200, 110, window=username)
window1.create_window(200, 170, window=password_confirm)
#If password is less than 2 characters emptyEntry will be called as an alert
emptyEntry = tk.Label(rootReg, text = 'Values must be over 2 characters!')
unmatchedPass = tk.Label(rootReg, text = 'Passwords do\n not match!')
userExists = tk.Label(rootReg, text = 'Username already exists try another!')
emptyEntry.configure({"background":"red"})
unmatchedPass.configure({"background":"red"})
#SQL Function for registering account
def register():
usernameInput = username.get()
if len(usernameInput) <= 2:
window1.create_window(200, 50, window=emptyEntry)
else:
#Check username does not already exist
c.execute("SELECT username FROM accounts WHERE username = (?)", (usernameInput,))
userresult = c.fetchall()
if len(userresult) > 0:
window1.create_window(200, 50, window=userExists)
else:
if password.get() == password_confirm.get():
if len(password.get()) <= 2:
window1.create_window(200, 50, window=emptyEntry)
else:
passwordInput = password.get()
SQLReg(usernameInput, passwordInput)
logingui("register")
Check if username already exists in database
Check password meets parameters



Check if password and username match the same in database



Attempted to add hashed password but with the time limit on my project i left it

Algorithm for probability checker - To find probability the values will be stored in an excel file but the school does not have a module that can read excel files so the values need to be stored in the sql file and the intelligent learning algorithm will need to be solved another way.

Implement Q-learning process using matrices to find probability of each question answered. If probability reaches 1 then they will level up.

	Lvl1	Lvl2	Lvl3	Lvl4	Lvl5	Lvl6	Lvl7	Lvl8	Lvl9	Lvl10
+	P(L1)	...								
-										
*										
/										

What is being tested	Input	Results expected	Actual outcome
Player movement	UP DOWN LEFT RIGHT	Player should move in the inputted direction by 1 step and play the animation	As expected

		depending on the direction being pressed or held.	
Player hit	Contact with obstacle	If the player is hit by the obstacles then the game should either restart with 1 life point taken away or the game should end	As expected
No collision between other obstacles		Start game multiple times and watch through to see if obstacles hit each other.	As the cars come close to each other they come to a stop Resolved by instead of decreasing the velocity as they get close switch the velocities of each car so they do not hit
Smooth scrolling background image	UP DOWN	Background image should roll smoothly as the player moves forwards or backwards. If the player reaches a barrier the background should not move.	As the image scrolls down it creates a windows xp tab like error where the last line of pixels is copied and displayed. This was resolved by adding 2 images of the background which go on top of each other as they go down.
Life counter		When the user dies on their last life the program should call game_lose() function which ends the game as a loss. If the user has more lives the game takes	As expected

		1 life away from the total amount of lives. The user continues on from where they died with the countdown carrying on from where it left off.	
Questions function being called when the user dies		When a user loses a life the game will pause as well as the timer and a random question will come up. the user will have 3 options to pick the correct answer.	After iterative tests one question gave an error which was x/0 ZeroDivisionError Resolved by adding a try and except function when ZeroDivisionError called then change the 0 to a 1
Questions answered correctly		If the user picks the correct answer then the game will unpause and carry on.	As expected
Questions answered wrong		If the user picks the wrong answer then the game will take one life away from the user. Giving the user a maximum of 3 chances to answer the question. If the user answers question wrong on their last life then the game also calling the game_lose() function.	As expected
Machine learning AI		The AI will need to	

		find what level of difficulty the user is on. Answering wrong will -1 from the backend point system. The program will ask a question easier than the previous question next time. Answering correctly will +1 to the point system and next time should ask you a harder question.	
Sound test		The program will have sound/music. Specific sounds will be played depending on the environment e.g. If the obstacle is a car there will be car noises.	As expected