Content Page:

1.0 Analysis

1.1 Problem

Education, the process of receiving or giving systematic instruction, especially at a school or university by a teacher or highly qualified person on the subject.

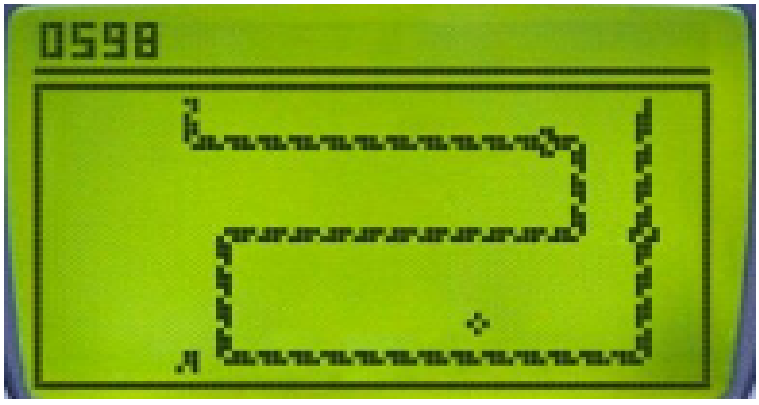
Traditional education methods often fail to engage modern students, leading to boredom and distractions. The challenge lies not in the students themselves but in how education is presented to them. Formal education, governed by official curricula, lacks the intuitive appeal needed to captivate today's learners. Conversely, informal education, rooted in lifelong learning experiences, holds promise for nurturing engaged and curious minds. Our goal as a society is to cultivate lifelong learners, ensuring a promising future by making education enjoyable and accessible. By reframing educational content in a fun and digestible format, I aim to spark interest and foster a lasting enthusiasm for learning, particularly in subjects like mathematics.

Snake Game Inspiration:

The original Snake game, known as "Snake Xenzia," achieved cultural significance despite its simplicity. Created by Finnish engineer Tanell Armato in the late 1990s, it gained popularity as a pre-installed game on Nokia mobile phones. Players controlled a growing snake, aiming to eat randomly generated apples while avoiding collisions with their own tail and the screen border. The game's addictive nature captivated millions worldwide, showcasing the power of simplicity and intuitive gameplay.Even though it was a simple game it became a cultural sensation entertaining millions of people around the world even at one point the game being ranked **41** on the “Top 100 Games of All Time” by Next Generation.

The aim of the game was simple: players controlled a snake whose length would grow every time it would eat the apple on the screen that was generated randomly by the program. All this while simultaneously avoiding collisions with the snake’s own tail and the border. The mission was to grow as long as you could without dying which would get more difficult as the snake would grow in size.

Example of how it looked like in peoples nokia screens back in the days when it was first launched:



Project Goals:

In this project, I aim to iterate upon the original game by retaining the classical game of Snake, leveraging its addictive gameplay to address educational challenges. I aim to revolutionise maths education by creating a gamified learning experience that engages students and fosters

lifelong learning habits. By making maths concepts more accessible and enjoyable, I strive to empower students to excel in their mathematical journey from an early age.

Role of Gamification:

Gamifying maths concepts is integral to my approach, as it motivates students to actively engage with the content. Through game mechanics such as scoring, level progression, and rewards, I aim to make learning enjoyable and rewarding. Immediate feedback reinforces learning and encourages continuous improvement, fostering a positive attitude towards maths

1.2 Stakeholders

Primary Stakeholders:

Primary schools are the primary stakeholders in this project, seeking to enhance their students' educational experiences beyond the classroom. They prioritise reliability and excellence in teaching and learning, utilising the platform for various purposes such as in-class activities and homework assignments. Additionally, special education needs students, such as those with ADHD, benefit from the software's ability to maintain a stable learning pace despite challenges in focus.

Mrs. Begum, a mathematics teacher at Marner Primary School, serves as a key stakeholder. Facing health issues that prevent her from attending school, she relies on the software to guide substitute teachers and stay informed about her students' progress. Addressing Mrs. Begum's needs is essential to ensuring that students do not fall behind due to unforeseen circumstances. To further support her, a report card feature will be implemented, enabling her to track students' performance and identify areas needing additional attention.

How will they use the software and what makes it suitable for their target audience?

Mrs. Begum, will actively engage with the game by integrating it into the curriculum for in-class activities and homework assignments, providing students with opportunities to practise maths skills in an engaging and interactive manner. Mrs. Begum, as a mathematics teacher, will utilise the software to guide her lesson plans and track students' progress remotely, leveraging its real-time insights and remote teaching support capabilities. The game's interactive nature and educational content make it

highly suitable for primary school students, offering a fun and effective way to learn maths concepts while meeting curriculum requirements

Secondary Stakeholders:

Parents are secondary stakeholders who may utilise the software for their toddlers to foster early learning. The user-friendly interface and engaging gameplay make it more akin to a game than a chore, encouraging toddlers to spend productive hours learning. For parents like Ahmet, who works long hours and is concerned about his son Mehmet's education, the software provides peace of mind and facilitates his son's early learning journey. Recognizing the importance of catering to children's preferences, features such as vibrant colours, music, and sounds are integrated to enhance engagement and retention of maths concepts.

Addressing the needs and preferences of primary schools, teachers, and parents ensures that the software meets the diverse requirements of its stakeholders. By providing reliable teaching support and opportunities for early learning, the software aims to empower educators and parents in facilitating children's academic success and lifelong learning.

How will they use the software and what makes it suitable for their target audience?

Parents can monitor their children's progress, provide additional support as needed, and appreciate the software's educational value, making it a valuable resource for supporting their children's learning journey outside of the classroom. Its intuitive interface and engaging gameplay make it accessible for parents and ensure that children enjoy learning while using the software, fostering a positive attitude towards maths education.

Why is it a computational method:

A computational method is one in which a problem that can be solved using an algorithm is computable. Problems that can be solved computationally typically consist of inputs, outputs and calculations. This must run on a computer or a modern device that is enabled to perform calculations rapidly,access user input through GUI and is a reasonable size. This makes a maths game inherently computational as it involves problem solving. Within the game, computational methods play a pivotal role in generating, evaluating, and managing mathematical content. For example, when presenting maths problems to players, computational methods are utilised to determine the correctness of their responses and adjust the game's dynamics accordingly. This

interactive approach not only engages players but also reinforces their maths skills and knowledge as they progress through the game.Computational methods facilitate the generation,evaluation and management of mathematical content within the game, making it interactive and engaging for players while reinforcing their maths skills and knowledge.

Problem recognition:

Primary Problem:

The primary problem revolves around ensuring the accuracy of user responses, managing high scores, and generating reports for stakeholders. This is crucial as it directly impacts the software's effectiveness and value proposition. Additionally, ensuring the game's stability and preventing crashes is essential for providing a seamless user experience.

Secondary Problem:

The secondary problem involves balancing the difficulty of the questions to meet national educational standards. Striking the right balance is critical to ensure that the software is widely applicable and beneficial for primary schools across the nation. It requires careful consideration of the curriculum and the target audience's proficiency levels.

Tertiary Problem:

Promoting healthy competition while mitigating toxic competition is a tertiary problem that requires attention. Toxic competition can lead to demotivation and hinder students' willingness to improve. Implementing mechanisms to foster a supportive and encouraging competitive environment is necessary to maximise the educational benefits of the software.

Final Problem:

Crafting an enjoyable user interface (UI) and user experience (UX) is the challenge. A designed UI/UX plays a role, in captivating players and ensuring their satisfaction with the game. Subpar UI/UX may lead to frustration. Diminish the gaming experience potentially affecting how well the software is received and utilised.

Effectively tackling these issues will be pivotal, for the success and effectiveness of your project. Each challenge presents its hurdles and aspects that must be carefully managed throughout the development phase.

Problem Decomposition:

1. **Primary problem:**

   -Checking if the user has selected the correct answer.

   -Keeping a high score and producing a report for stakeholders.

   -Ensuring the game runs smoothly without crashes.

2. **Secondary problem:**

   -Adapting the program for national use in primary schools.

   -Balancing question difficulty to meet educational standards.

3. **Tertiary problem:**

   -Promoting healthy competition among students.

   -Avoiding toxic competition, which can demotivate students.

4. **Final problem:**

   -Developing a fun, interactive User Interface (UI) and User Experience (UX).

   -Ensuring the interface is intuitive and user-friendly to enhance the player's gaming experience.

Objects in my game:

-Snake→ the one the user will control

-fruit → the ones that the user will have to select

-stones→ makes the game more challenging since if the user makes contact with it, the game will end instantly.

-boms→ a more sophisticated version of the stone object which will produce an explosion animation

Problem Abstraction:

To effectively solve this problem it is essential to ignore all specific details and any patterns that will not help us solve our problem. This helps us to form our idea of the problem.In this problem we could ignore the snake having any turning animations when going from left to right and rather worry about its main objective which is to move. We could also not worry about making destruction sound and animation excessive as follows when the snake makes contact with the stone or bomb sophisticated as this occupies more RAM causing the program to produce lag and has no significant impact on the final objective of the program. There is also no need for high quality graphics as the game is meant to give a retro game feeling and experience.

Divide and Conquer

Divide and conquer is a problem-solving technique that involves breaking down a complex problem into smaller, more manageable sub-problems, solving each sub-problem independently, and then combining the solutions to solve the original problem.I will systematically break down the complex task into smaller, more manageable components. This involves segmenting the development process into distinct modules, with each module focusing on a specific aspect of the game, such as graphical user interface (GUI) design, game mechanics, or audiovisual elements., I can address each sub-problem separately without being overwhelmed by the entire project . Once each module is developed independently, I will integrate them together to form the complete game, ensuring seamless interaction between modules through rigorous testing and refinement. This iterative process allows me to continuously improve and optimise each module based on feedback and testing results, leading to a structured and efficient solution. This could be done by making prototypes of each component and than iterating upon it after doing the test and asking for feedback from stakeholders.

Thinking Ahead

This is a way of looking ahead and knowing what you want. It allows you to plan a bit, giving the solution of your problem a bit of structure. For instance, for my problem:

• I would plan to use pygame as this is good software to write games with

• Inputs for the game will most likely be the use of a keyboard

• The outputs for the game will consist of sound effects and various animations for the game

• Score system should be included to give the user something to aim towards

Thinking Procedurally and Logically

Approaching problem-solving with a procedural and logical mindset involves logically breaking down tasks into manageable steps, enhancing efficiency and organisation throughout the development process. Here's how I plan to apply this approach in my game development:

-Breaking down the development tasks into smaller, more manageable components

-Implementing each aspect of the game systematically, addressing one component at a time, e.g finishing the functionalities of the snake function then going on to the fruit.

-Iteratively refining and optimising each component based on feedback and testing results.

-Ensuring seamless integration of all components to create a cohesive and well-structured game, e.g integrating the login system with the main software

-By thinking procedurally and logically, I aim to streamline the development process and create a polished and efficient final product.

Thinking Concurrently

In developing this game, a key focus will be on ensuring that various aspects of the game, such as graphics rendering, user input processing, and game logic execution, occur simultaneously to deliver smooth and responsive gameplay. Utilising multithreading techniques, tasks like rendering animations and handling player interactions will run concurrently with the main game loop, allowing for seamless interaction between the player and the game environment. By employing concurrent processing, the game will maintain a high level of interactivity and immersion, enhancing the overall gaming experience for the player.

Interview:

I will outline the main questions I have asked one of my stakeholders, my reasoning for asking that specific question and what the stakeholder answered.

The questions were:

1)Are you currently satisfied with the education your son has been receiving?

2)What are the advantages to this type of education?

3)What is one flow in this type?
4) How would you propose a solution?

Reasoning behind it:

-Question one was to mainly to initiate a conversation easing the stakeholder into the interview so that they could answer the rest of the questions truthfully.

-Question two highlights the things the stakeholder liked about their current education provider and why they liked this attribute

-Question three contrasts the things the stakeholder disliked about their current education provider and why they disliked this attribute

-Question four is to receive an opinionated answer so that I can add it to my record of all the things stakeholders want from the software

Answers by stakeholder Ahmet

1)Are you currently satisfied with the education your son has been receiving?

”For now yes because he is still in year 2 so he is not been confronted with very hard content so he is able to digest it without issues so far”

2)What are the advantages to this type of education?

”Well, one advantage is the classroom setting. It allows students to support each other, especially if someone is struggling. Also, there's a bit of healthy competition – the top performers in weekly tests get rewarded with sweets, which motivates them to do well.”

3)What is one flow in this type?

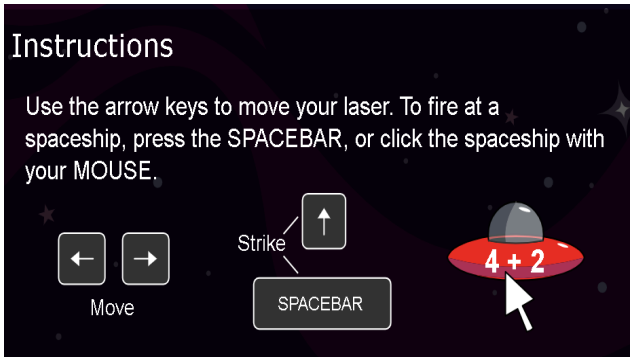
”One flaw is that the teachers aim to keep the whole class moving at the same pace. It's tough because every class has a mix of bright students and those who need more time to grasp concepts. So, some students may fall behind if they don't receive the extra attention they need”

4) How would you propose a solution?

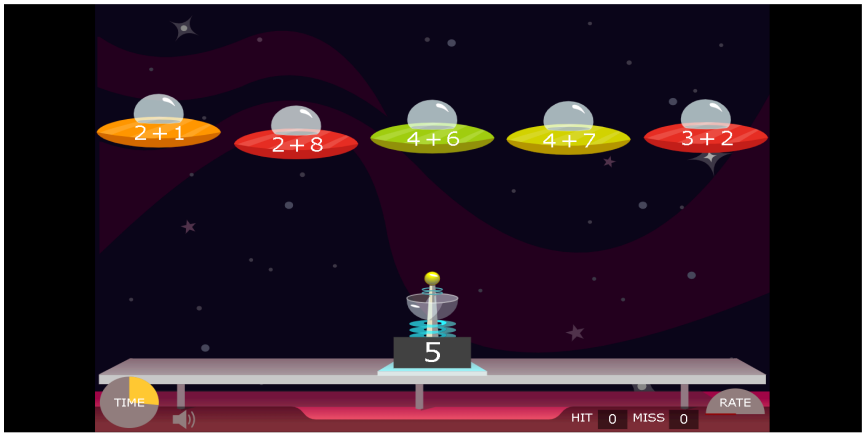
”I think the teachers should identify students who are struggling and provide them with extra support. Maybe offer additional after-school sessions or pair them up with classmates for extra practice on challenging topics. It's about ensuring that every student gets the help they need to succeed.”

1.3 Research

Existing solution 1 : **Alien addition**



In this game we are greeted initially by a set of instructions to explain new users how the game works and to current players a refresher incase they forget how the game is played



Link: [Alien Addition | Math Playground]

Game Summary:

In this game you are a laser beam which can move only horizontally. You are responsible for shooting at enemy ships that will be moving down vertically. Your laser will contain an answer and the enemy ships will contain a question. Your job is to match the answers with the corresponding questions. If you select the correct answer your score will go up and the time function will reset. However if you select the wrong answer the miss counter will go up and the timer function won't reset.

Problems I have found:

1) The spaceships come down all at once

2)Users can spam shots as there is no cooldown timer

3)you can only use the arrow keys and not the “WASD” keys

4) The enemies would not shoot back at the user

5)There would be no other objects in the game such as power ups or missiles making it boring

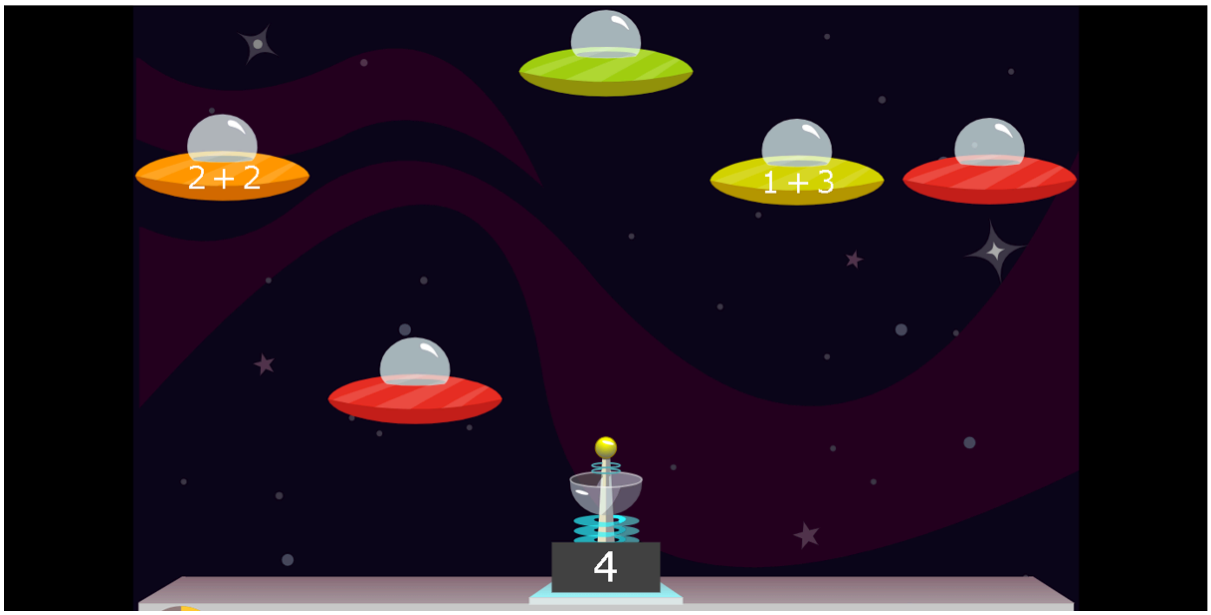
6) There is only one mode with that being addition

7)No pause functionalities

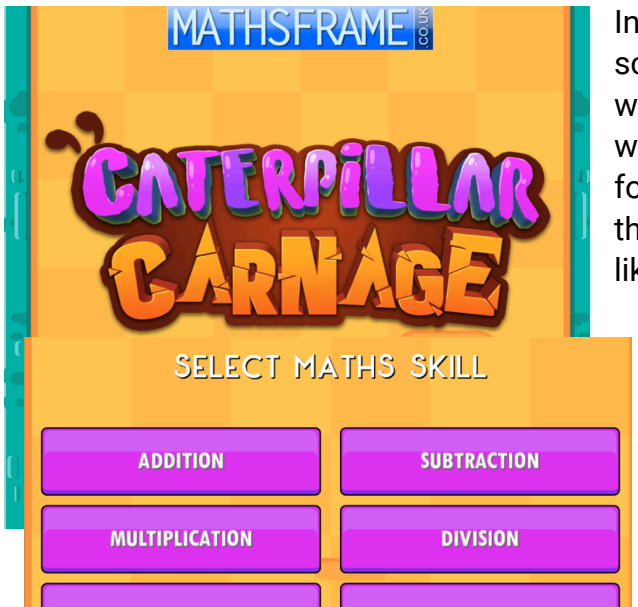
8)No leaderboard functionality to check high score

9) Game can only be accessed when internet connection is present

10) Regular glitching and bugs as seen in the image below:



Existing solution 2 : **Caterpillar Carnegie**



In this game we are greeted with a colourful screen perfect for kids. We are also greeted with 2 options one for the scores function which displays the leaderboard and another for the play button which allows the user to then click and select the mode they would like to play.

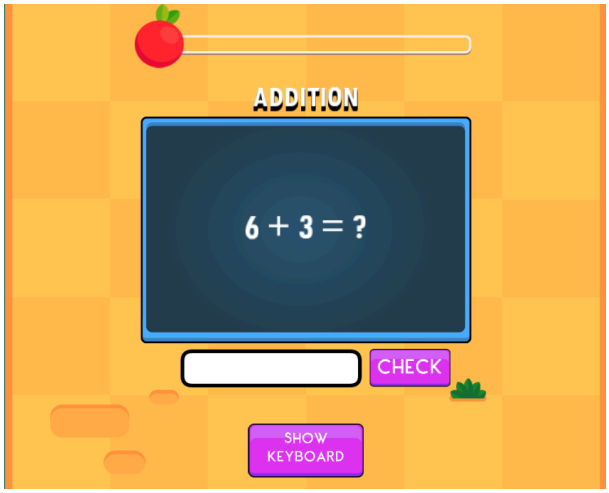
The options includes addition,subtraction,multiplication and division(modes I aim to have) but also has more complicated modes such as percentages and even fractions as seen in the image at the left:

Link: [Caterpillar Carnage - Mathsframe]

Game summary:

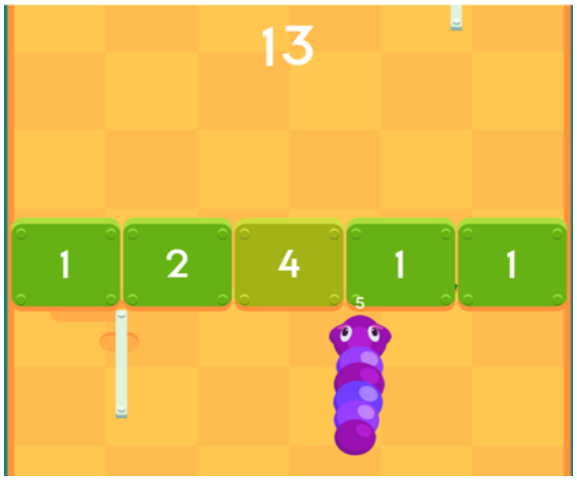
In this game, players control a caterpillar moving vertically upwards to collect apples of varying values. Each collected apple increases the length of the caterpillar, with higher value apples resulting in greater length increments. However, the caterpillar must navigate through walls, each representing a numeric value, to continue its ascent. Players must strategically choose the wall with the lowest value to decrease the caterpillar's length and prolong its survival.

Before accessing the game, players must answer initial questions displayed on the screen correctly. Only upon providing accurate responses can players proceed to play the game. This is seen below:



The apple on the top acts like a loading widget to show how many questions are left until the game starts

Example of how the game would look once answered the questions:



Problems I have found:

1) Game can only be accessed when internet connection is present
2) Game cannot be played without a mouse as it dont validate keyboard inputs

3) There is not timer function when answering the questions
4) No login system

Further interview with stakeholder(ahmet):

After having researched the games I showed my findings to my stakeholder ahmet and asked for him if how he felt about my research:

What he had to say:

“Your research is optimum for a game that could be used for a primary kid student. After thoroughly looking at both of your games it is quite clear that the second one is way better than the first one. However, I think some of the features are a bit excessive for a simple maths game. By this I mean in the snake game there are just too many options for the kid to choose which might overwhelm them and demotivate them with the sheer amount of work and practice they will have to do.I would rather recommend having the basic arithmetic(addition,subtraction,multiplication,division) down. This is because having a strong understanding of the main operations in maths is paramount for having a shallow knowledge across all”

Parts that I can apply to my solution:

After researching existing games and considering feedback from stakeholders, I aim to develop a solution that combines features from both games while excluding those that are not pertinent. Emphasising quality over quantity, my solution will offer four game modes - Addition, Subtraction, Multiplication, and Division - each with its own algorithm for generating randomised questions. Additionally, I will implement a leaderboard function to track high scores and a timer feature to introduce a challenging element. While mouse input will be used for mode selection, keyboard controls will be employed for gameplay, accommodating both arrow keys and "WASD" keys to ensure accessibility for users with different keyboard configurations. To address the issue of internet dependency, I will create a desktop application with a local login system to store user account details securely. Drawing inspiration from my research, I will also integrate features such as the ability to pause the game, catering to user convenience and enhancing overall gameplay experience.This is seen below:

However, with my research I will abstract some useless parts with the main one being an instruction page as my game wont have any instruction page as it will be intuitive for everyone to know what the controls will be.

1.4 Essential features

The essential features of the software prioritise simplicity and accessibility, evident from the user-friendly menu upon launch, offering four distinct options for addition, subtraction, multiplication, and division. Each game mode corresponds to a specific mathematical operation, providing tailored practice sessions for users based on their desired skill set. This customisation ensures that users can focus on honing specific maths abilities relevant to their learning goals, enhancing the software's effectiveness as an educational tool. Direct access to the leaderboard from the main menu enhances user engagement and motivation, allowing users to readily monitor high scores and track their progress relative to other players, fostering a sense of competition and achievement within the software's community. The software's streamlined design enables users to navigate the game seamlessly and concentrate on improving their maths skills, minimising unnecessary complexities in the user interface and gameplay mechanics. Behind the scenes, the software efficiently manages user data, including registration information and leaderboard rankings, ensuring that data is securely stored and readily accessible, facilitating personalised experiences and maintaining the integrity of the software's educational features.

1.5 Limitations

1)My software will not have online functionality, preventing teachers from remotely monitoring student performance. However, to address this limitation, I will incorporate a leaderboard feature to track individual student progress within the game environment.

2)Another limitation is the lack of mouse input; all game commands will be executed using the keyboard. Players can navigate using either the arrow keys or the 'WASD' keys for movement.

3)Due to data protection regulations, sensitive information such as gender and age of students will not be stored within the software, adhering to the Data Protection Act 2016.

4)The software will be limited to testing on computers and laptops, restricting compatibility with other devices.

1.6 Requirements

Software and Hardware requirements

Hardware:

A computer or laptop which is able to run the software with the standard IOS peripherals. Even though a mouse will not be needed when playing the game it will still be needed when selecting the options from the menus. The keyboard will be the most essential component and the user must ensure that the keys and the “WASDS” keys are working properly since without those the user won't be able to shoot. The computer must also have speakers or some sort of sound system to allow the music and the sound to play.

Additional features:

-minimum 2GB RAM as this is optimal for simple 2D games.

-minimum 60Hz even though 120Hz would be optimum

- minimum of 10 GB of storage is recommended, with a solid-state drive (SSD) being preferable to a traditional hard disk drive (HDD)

Software:

Windows, Mac or Linux→ these OS are are supported by python, open CV and pygame

Integrated Development Environment(IDE) → I will use visual studio code as its lightweight, meaning its more simple as it contains minimalist interface suitable for an A-Level project

Python interpreter→ This will allow the code to be run

Pygame→This is a python module that allowed games to be created and run

Stakeholders requirements:

Requirements	Explanation
Simple colourful menu	Ease of navigation by minimising confusion and fostering a user-friendly experience. Its straightforward design promotes quick comprehension, allowing young users to focus on the educational content without unnecessary complexity.

A leaderboard button	Promotes healthy competition between kids
Secure Login	To ensure it's them connecting to their account and not others. This is so that unauthorised users cannot use the software under someone else's account as this will fall under the Computer Misuse Act 1990
Large writings	To ensure text is visible allowing users to select the correct options as I have taken into account that some students might have vision problems.

Hardware and software:

Requirements	Explanation
Standard peripherals e.g keyboard and mouse	The user needs to navigate through the menu with a mouse to select the modes and it needs a keyboard to actually function the snakeT
Dual processor 1.6GHz or equivalent	Provides basic processor power for smooth gameplay
RAM 2GB	Sufficient memory to run a 2D retro game smoothly and other background applications
Pre-installed: Python, Pygame and SQLite.	Game will be developed using python and a special module named pygame. Database functionality will be done with the SQLite module.

1.7 Success Criteria

No.	Criteria	Justification
1	Display the score	This is to tell the player how well they are doing,

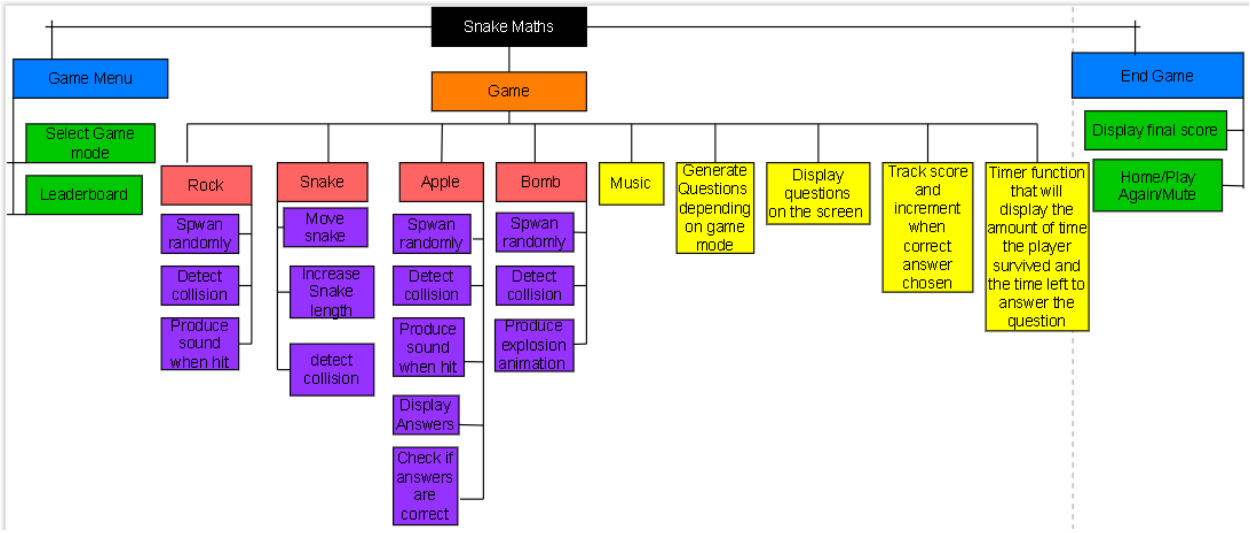
		encouraging them to collect more points
2	Display the time the user has survived	This is to tell the player how well they are doing, encouraging them to survive for longer
3	Display the time left to answer question	To let the player know how long they have left
4	Display two fruit objects	Allows the player to select the correct answer between the 2 options
5	Randomly display a rock	This makes collision more likely, making the game progressively harder
6	Display a boundary the player cannot cross	Restrict movement beyond the game space, keeping the gameplay within the specified location.
7	Display questions	This is to let the player formulate the answer in their head before selecting the right answer
8	Have a leaderboard to show highest scores	To promote healthy competition
9	Have a registration and log in system	To allow only user of that organisation to use the software
10	Having a pin for registering	This is do that only admin can register students
11	Having sound for different collisions	This is to let the player know if they collected a point or died due to a rock or bomb
12	Music playing	To add more life to the game

13	Mute function	To stop the music playing
14	Reload Button	To allow the user to start a new game without having to open the program again
15	Home Button	To allow the programmer to access the home menu without having to start the program again

2.0 Design

2.1 Problem decomposition

Structure Diagram:



Why structure a diagram?

The use of a structured diagram to visualise the problem because it offers several advantages that greatly aid in understanding the complexities of my game development project. Firstly, a structured diagram provides a clear and organised representation of the relationships between different components and classes within the game, helping me conceptualise the overall architecture more effectively. By visually depicting the hierarchy of classes and their interactions, I can easily identify dependencies and design patterns, facilitating smoother development and debugging processes. Moreover, the diagram serves as a valuable communication tool, allowing me to convey the system's structure to collaborators or stakeholders in a concise and intuitive manner.

Explanation of every colour

BLACK = Represents the game name, providing an overarching identifier for the project.

BLUE = Denotes key interfaces visible to the user, such as menus, prompts, or interactive elements that drive user decisions and influence program flow. These interfaces are pivotal as they directly affect the progression and outcome of the game.

Game Menu: gives players the chance to pick their game mode and check out the leaderboard. It lets players customise their gaming experience by choosing how they want to play and seeing how they stack up against others. This feature boosts player engagement and satisfaction by giving them choices and feedback.

End Game: This interface displays the player's final score and provides options to return to the home screen, play again, or mute the game.

Justification: The end game interface provides closure to the gaming session, allowing players to review their performance and make decisions for their next steps. It enhances user experience by offering clear navigation and feedback options, promoting continued engagement and enjoyment

GREEN = Utility functions fulfil various aspects of software functionality. These functions are not specifically tied to any certain class.

Select Game Mode: This allows players to choose their preferred gameplay mode. They can select addition, subtraction, multiplication, or division. Justifying this feature, it presents players with the ability to adapt their gameplay. They can tailor it to their skill levels or personal preferences. This significantly enhances user engagement. It also increases satisfaction by offering varied play styles. The customization caters to a range of different learning needs.

Leaderboard: This feature showcases top performing players' scores. It provides a source of motivation for users, encouraging them to improve their performance and promote competitiveness. It enhances engagement and user retention by encouraging players to aim for higher scores. The board cultivates a sense of community. Players can compare their achievements with those of others, leading to shared success celebrations.

Justification:By consolidating all game-related functionalities within the Game class, including collision detection, score tracking, and obstacle spawning, the program benefits from enhanced code organisation, modularity, and maintainability. This feature highlights the highest scores of top-tier players. It serves as a motivational tool for users, spurring them to enhance their performance and foster a spirit of competition. By inspiring players to strive for superior scores, it bolsters engagement and helps retain users. Moreover, the board fosters a sense of community. Players have the opportunity

to measure their success against the accomplishments of their peers. This inevitably leads to collective celebrations of success.

ORANGE = Highlights the classes intended for the program, with a specific emphasis on the **GAME** class. This colour delineates the core structure of the program, encapsulating the main game logic and functionality.

Game Class: The Game class will serve as the central controller for the entire game, encapsulating the core gameplay mechanics and orchestrating interactions between various game elements. It will include essential methods for initialising the game, updating the game state, and handling player input. Additionally, the Game class will instantiate other key game components, such as the Snake, Fruit, Stone, and Mode objects, which will collectively define the game environment and dynamics.

Justification :by consolidating all game related functionalities within a single class such as collision detection score tracking and obstacle spawning the game class will promote code organisation modularity and maintainability this design approach will simplify debugging facilitate code reuse and enhance scalability making it easier to manage and extend the game s functionality as development progresses additionally encapsulating game logic within a dedicated class will promote separation of concerns and adhere to object oriented programming principles leading to cleaner more structured code and a more cohesive development workflow

YELLOW = Represents methods belonging to the **GAME** class, which operate in conjunction with other nested classes within the **GAME** class.

Music: This method controls the background music played during gameplay; it enhances the game's atmosphere and immerses players in the gameplay experience.

Justification : Incorporating music into the game adds a sensory element to the overall experience, contributing to player engagement and enjoyment.

Generate Question: This method generates mathematical questions based on the selected game mode (addition, subtraction, multiplication, or division). It randomly generates operands and operators to create diverse and challenging questions for players to solve.

Justification: The Generate Question method ensures that players encounter a variety of maths problems during gameplay, fostering learning and cognitive development while keeping the game engaging and educational.

Display Questions: This method displays the generated mathematical questions on the game screen, presenting them to players in a clear and visually appealing manner. It ensures that questions are prominently featured and easily accessible, allowing players to focus on solving them while navigating the game environment.

Justification : Clear presentation of questions is essential for facilitating player comprehension and interaction, promoting effective gameplay and educational outcomes.

Track Score: This method tracks and updates the player's score based on their performance in the game incrementing the score for each correct answer it provides feedback to players on their progress and performance motivating them to strive for improvement and mastery

Justification : It adds a competitive element to the gameplay, encouraging players to challenge themselves and achieve higher scores, enhancing long-term engagement.

Timer Function: This method manages the game timer displaying the elapsed time since the start of the game and the remaining time for the player to answer each question it creates a sense of urgency and pressure prompting players to make quick decisions and improving their time management skills

Justification: Incorporating a timer adds an extra layer of challenge and excitement to the gameplay, creating suspense and training players to think and act quickly to succeed.

RED = The decision to nest instantiated classes within the **GAME** class ensures a more organised and streamlined approach to managing game logic. By encapsulating these classes within **GAME**, I promote modularity, enhance code readability, and facilitate easier coordination of game elements. This approach reduces namespace clutter, minimises naming conflicts, and promotes code reuse and scalability, ultimately contributing to a more efficient and maintainable codebase.

Rock Class: Represents the rocks or obstacles within the game. These objects are placed randomly on the game grid and serve as obstacles for the snake to avoid.

Justification: By encapsulating rock functionality within a class, I can easily manage their behaviour, including spawning, collision detection, and sound effects, providing a modular and organised approach to game development.

Apple Class: Each apple object within the game shares identical visual characteristics, including shape, size, and colour. The only distinguishing feature between apples is the numerical value they display, representing the answer options for the player. When instantiated, apples are assigned either the correct or incorrect answer randomly. During gameplay, the player-controlled snake interacts with these apples, and upon collision, the game evaluates whether the selected answer is correct. If the player chooses the correct answer, the game continues, and the score increments. However, selecting the incorrect answer results in the game ending, with the final score displayed to the player.

This uniformity in appearance ensures consistency in gameplay mechanics while providing players with challenging decision-making opportunities.

Justification: Standardising the visual presentation of apples simplifies game development and enhances player experience by offering clear and intuitive interactions. By maintaining consistency in appearance, players can focus on gameplay without being distracted by unnecessary visual differences between apples. This approach also promotes accessibility and inclusivity, ensuring that all players can engage with the game regardless of their background or ability. Additionally, it streamlines the development process by reducing the need for complex visual assets and animations, allowing for faster iteration and deployment of new features.

Bomb Class: Represents the bombs or hazards within the game. These objects pose a threat to the snake and lead to negative consequences upon collision.

Justification: Employing a separate class for bombs enables precise control over their behaviour, including spawning, collision detection, and sound effects, facilitating clear and organised code structure for easier debugging and modification.

PURPLE = Denotes the methods associated with the instantiated classes . These methods define the behaviour and functionality of each instantiated class, contributing to the overall gameplay experience and game mechanics.

Spawn Randomly: This method generates obstacles, such as fruits or bombs, at random positions within the game grid. When called, it selects random coordinates within the game area and creates obstacle objects at those positions. This randomness ensures varied and unpredictable placement of obstacles throughout the game, adding challenge and excitement to each playthrough.

Justification: This approach emulates the behaviour of classic snake games, where drops appear randomly on the game board. By introducing this unpredictability, the game becomes more engaging and challenging for players, enhancing their overall experience.

Detect Collision: The collision detection method continuously checks for collisions between the snake and obstacles present in the game grid. It verifies if the snake's position overlaps with any obstacle, signalling a collision event when detected. This functionality is crucial for enforcing game rules and providing immediate feedback to players regarding their interactions with obstacles.

Justification: Accurate collision detection ensures that players experience fair and consistent gameplay. By promptly detecting collisions, the game maintains realism and responsiveness, enhancing player immersion and enjoyment.

Produce Sound when Hit: This method triggers sound effects when the snake collides with an obstacle. It plays an auditory cue to provide immediate feedback to the player upon collision events. These sound effects enhance the overall gaming experience by adding an additional sensory dimension to the gameplay.

Justification: Incorporating sound feedback upon collision events enriches player engagement and immersion. By providing auditory cues, the game reinforces player actions, heightens tension during critical moments, and amplifies the overall gaming experience, leading to increased player satisfaction.

I will now produce a list of objectives after deciding on the specific requirements of my stakeholders. I will develop the game in python mainly using the pygame module to create the main functionality and design of the game.Tkinter will be used to create the GUI and the functionality of the settings.SQLite will be used to create the databases and store the information.

Design Objectives:

Aesthetics:

General Features:

1)Window will be full screen

2)Black text that once hovered over will be highlighted as green

3)Orange buttons

Specific Features:

-Main menu

1)Title of the game

2)Snake Icon for the game

3)colourful main menu

Highscore:

1)Organised tiles of score

Input:

1)Button to exit the game

2)Button to select what game mode you want to play

Game Screen:

1)Key to pause the game

2)The program will listen to keyboard inputs to control the player

Game over:

1)Button to go back to home menu

2)Button to continue playing in the same game mode

3)Button to exit the game

Game Screen:

1)Load all the background images

2)Load all the objects

3)Load the scorebar

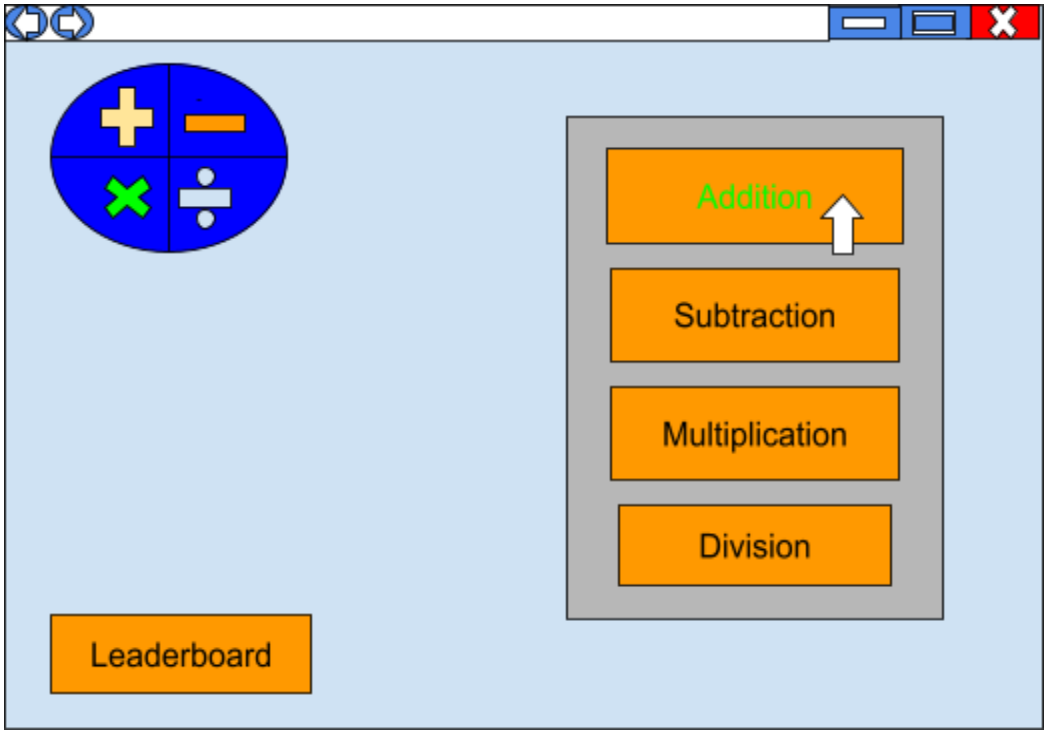
4)Detect collision between objects

5)Load the border

2.2 Structure of the solution

USER INTERFACE DESIGN:

Main game menu:



The main menu will contain:

Aesthetics:

Main menu

1)Title of the game

2)Red Text

3)colourful main menu

4)Basic so it is intuitive for the students

Processing:

Main Menu:

1)When any button is pressed it will take you to that screen

Input:

Main Menu:

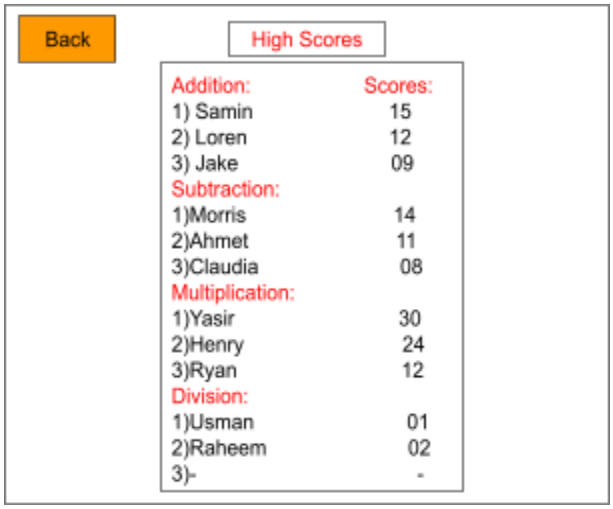
- Option to select to learn/practice: - addition

-multiplication

-subtraction

-division

Leaderboard:



The main menu will contain:

Aesthetics:

Leaderboard menu

1)High score title at the top to help remember student what page their on

2)names on the left and their corresponding scores on the right

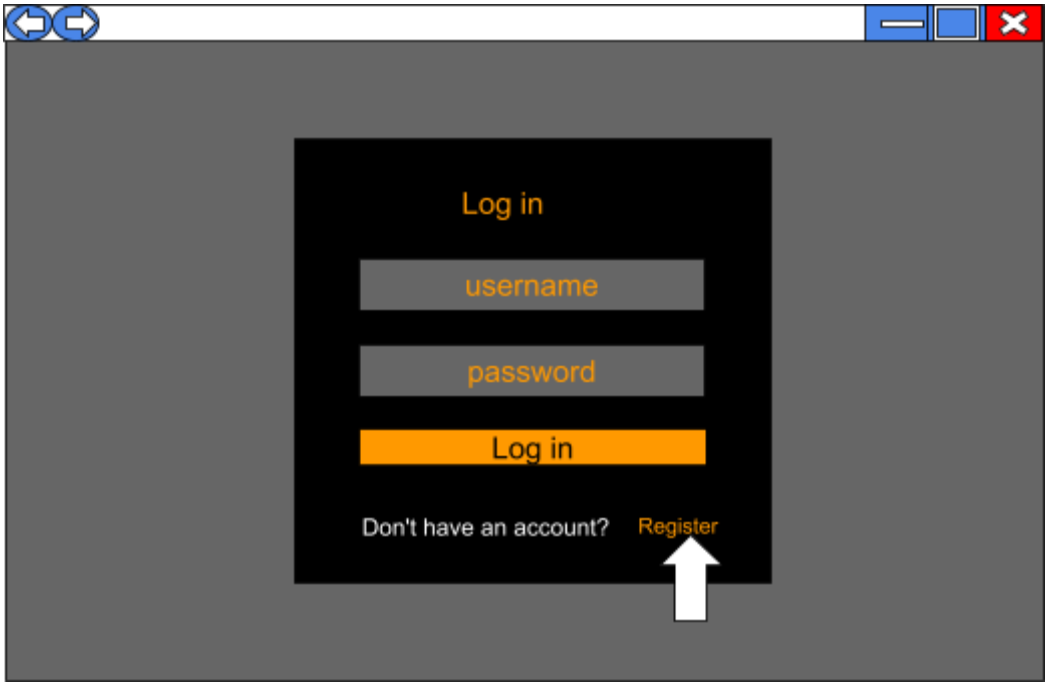
3)Scores that have not been inputted yet are market with "-" to let everyone know no one had for now reached that ranking e.g in this instance we can see that only 2 people played the division mode has been played only twice explaining why the 3rd spot is left empty

Processing:

Back Menu:

1)When button pressed will take you back to the main menu

Login system:



The login menu will contain:

Aesthetics:

Login menu

1)Log in title

2)Orange Text

3)colourful main menu

4)Basic so it is intuitive for the students

Processing:

Login Menu:

1)When the entry field is pressed writing on it will disappear allowing only the users writing

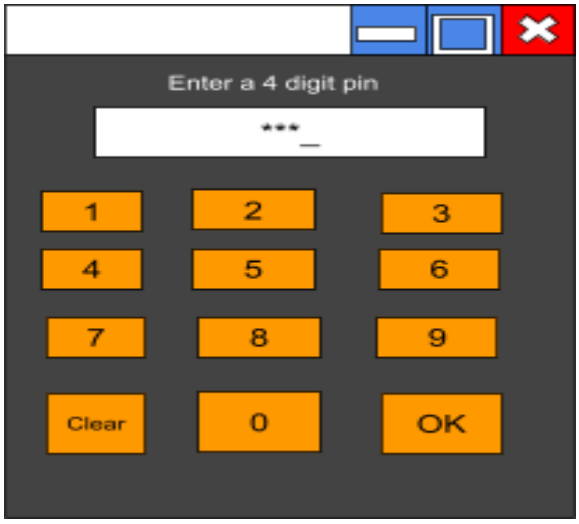
2) When Login is pressed the game menu will appear and the user will be signed in

3)If the user does not own a login they can press register which will lead them to the registration page

I have designed to make the login screen lightweight but also slick, inspiring me to choose the colour combo of black grey and white which go very well together. The cursor currently is pointing towards the registration link which will open another page.

Registration System:

PIN:



Aesthetics:

Pin menu

1)Number buttons displayed

2)Orange Buttons

3)colourful main menu

4)Basic so it is intuitive for the students

5)contains validation checks such as length check

6)Clear button to clear previous numbers

7) Asterisks added for further encryption

Processing:

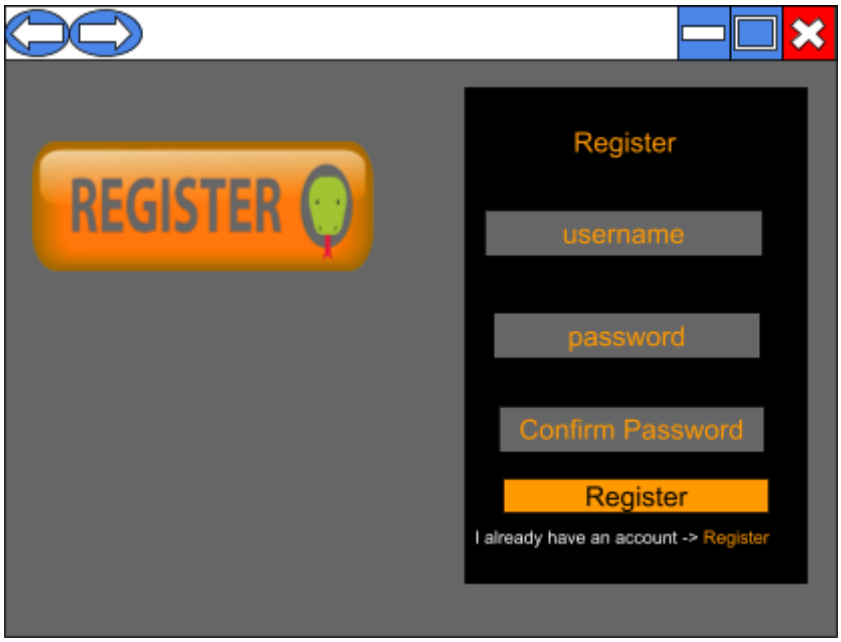
Pin Menu:

1)When Ok is pressed and the pin is correct the registration page opens

To first access the registration system you have to enter the pin in order to register your students. This is so only admins have access to functionality and prevents students from creating multiple accounts that

are not going to overload the database.

Registering page:



The registration menu will contain:

Aesthetics:

Login menu

1)Registration title

2)Orange Text

3)colourful main menu

4)Basic so it is intuitive for the students

Processing:

Login Menu:

1)When the entry field is pressed writing on it will disappear allowing only the users writing

2) When register is pressed a message box will appear showing how you registered

3)Registration contains validation such as minimum characters and for the passwords to match

3) Login page in the bottom allowing the user to login once registered

This Page has a design similar to the login page, the only difference being that the user has to confirm the password once.

2.3 Algorithm design

The snake game will contain many complex algorithms. The main ones being creating the snake, implementing logic for movement, collision and object generation. There are those additional algorithms needed for the login and registration alongside finding a way of saving them inside a leaderboard.

Why am I going to be using pseudocode over flowcharts to display my algorithms?

Flowcharts get unwieldy very quickly when algorithms become more complex and are time-consuming to produce and amend.

The following pseudocode for a basic snake game without any maths functionality involved. This is because my goal is to first build a basic snake game and than iterate the code to add all the maths functionality needed.

For readability I am going to make all the comments red , functions purple, strings green and classes orange :

Function	Pseudocode	Justification

Loading Background image and drawing a grid	`# Load background image and draw grid`
	`FUNCTION draw_grid_over_image(image_path, grid_size)`
	`    background_image = load_image(image_path)`
	`    cell_size = background_image.width DIV grid_size`
	
	`    # Draw horizontal lines`
	`    FOR row = 0 TO grid_size`
	`        draw_horizontal_line(row * cell_size)`
	`    ENDFOR`
	
	`    # Draw vertical lines`
	`    FOR col = 0 TO grid_size`
		By using minimal lines, it ensures that the main steps are easily understandable. The placeholder functions are strategically employed to represent actions dependent on the specific programming language or graphics library, allowing for flexibility in implementation. The variable names, such as `image_path`, `grid_size`, and `background_image`, are chosen to be straightforward and descriptive, contributing to the overall clarity of the pseudocode. This simplicity and logical flow provide a foundation for loading an

	draw_vertical_line(col * cell_size) ENDFOR show_image(background_image) ENDFUNCTION # Placeholder functions for simplicity FUNCTION load_image(image_path) # Placeholder logic for loading image ENDFUNCTION FUNCTION draw_horizontal_line(y_position) # Placeholder logic for drawing horizontal line ENDFUNCTION FUNCTION draw_vertical_line(x_position) # Placeholder logic for drawing vertical line ENDFUNCTION FUNCTION show_image(image) # Placeholder logic for displaying image ENDFUNCTION	image and overlaying a grid without unnecessary complexity.
Making the snake and its methods.	CLASS Snake: PRIVATE body PRIVATE direction	The `move_snake` method allows for a straightforward update of the snake's direction based on user input, providing a concise and readable way to handle

	PUBLIC PROCEDURE NEW(initial_length, start_position, initial_direction):
	body = list containing start_position subtracted by i times initial_direction for each i in range(initial_length)
	direction = initial_direction
	ENDPROCEDURE
	
	PUBLIC PROCEDURE move_snake(user_input):
	IF user_input == "UP" THEN
	direction = (0, -1) # Move upward
	ELSEIF user_input == "DOWN" THEN
	direction = (0, 1) # Move downward
	ELSEIF user_input == "LEFT" THEN
	direction = (-1, 0) # Move left
	ELSEIF user_input == "RIGHT" THEN
	direction = (1, 0) # Move right
	ENDIF
	
	CALL move()
	
	ENDPROCEDURE
	
	PRIVATE PROCEDURE move():
	new_head = (body[0][0] + direction[0], body[0][1] + direction[1])
	body = list containing new_head followed by all elements of body excluding the last one
	ENDPROCEDURE
	
	PUBLIC PROCEDURE add_segment():
	tail = last element of body
		movement. The `move` method efficiently updates the snake's position by adjusting the positions of its body segments, and the `add_segment` method simplifies the addition of a new segment when the snake grows. The `render` method encapsulates the logic for rendering the snake on the game grid. By utilising tuples to represent positions and keeping the code concise, the pseudocode maintains readability and serves as a foundation for translating the logic into actual Python code within a game development context.

	new_segment = (tail[0] + direction[0], tail[1] + direction[1])
	append new_segment to body
	ENDPROCEDURE
	
	PUBLIC PROCEDURE render():
	FOR EACH segment IN body DO
	CALL render_snake_segment(segment)
	ENDFOR
	ENDPROCEDURE	

Movement	IF Key_input == "W": user_input = "UP" ELSEIF Key_input == "S": user_input = "DOWN" ELSEIF Key_input == "A": user_input = "LEFT" ELSEIF key_input == "D": user_input = "RIGHT"	This restricts the snake's movement to only 4 directions and only one direction at a time. This is to prevent the snake from moving sideways and only makes it so it can move vertically or horizontally at once
Rendering	FUNCTION render_game(): render_game_grid() render_snake() render_food() ENDFUNCTION	The `render_game` function strategically calls `render_game_grid` before `render_snake` to ensure the game grid is displayed first. This intentional order prevents the snake from

		spawning before the grid, maintaining a logical sequence in rendering. Subsequently, the call to `render_food` adds the visual representation of food on the grid. This modular approach enhances code organisation, readability, and ensures a visually coherent rendering of game elements.

Apple spawning	`FUNCTION generate_new_apple():`
	
	`    # Generate random coordinates for the new apple within the game grid`
	
	`    apple_x = random_number_between(0, grid_width - 1)`
	
	`    apple_y = random_number_between(0, grid_height - 1)`
	
	`    # Ensure the apple does not spawn on the snake's body`
	
	`    while (apple_x, apple_y) in snake_body:`
	
	`        apple_x = random_number_between(0, grid_width - 1)`
	
	`        apple_y = random_number_between(0, grid_height - 1)`
	
	`    # Set the apple coordinates`
	
	`    apple_position = (apple_x, apple_y)`
	
	`    RETURN apple_position`
	
	`ENDFUNCTION`
		This code ensures the apple grows in random positions within the grid by generating a random x and a random y coordinate. The code keeps generating these until the does not spawn on the snake's body to ensure game logic does not get corrupted.

Checking if apple has been eaten	`FUNCTION check_apple_eaten():`
	`    if snake_head == apple_position:`
	`        increase_score()`
	`        grow_snake()`
	`        apple_position = generate_new_apple()`
	
	`        # Spawn a new bomb when the score goes up`
	`        if score % bomb_spawn_threshold == 0:`
	`            bomb = generate_new_bomb()`
	`ENDFUNCTION`
		This function, `check_apple_eaten`, efficiently detects if the snake's head collides with the current apple position. Upon collision, the player's score increases, and the snake grows. This adds an engaging gameplay element, rewarding the player's success. The subsequent regeneration of the apple position with `generate_new_apple()` ensures a continuous challenge.

Bomb spawning and explosion	`CLASS Bomb:`
	`    PRIVATE position = None`
	
	`    PUBLIC PROCEDURE NEW():`
	`        position = generate_new_bomb_position()`
	
	`    PRIVATE FUNCTION generate_new_bomb_position():`
	`        #Generate random coordinates for the new bomb within the game grid`
	`        bomb_x = random_number_between(0, grid_width - 1)`
	`        bomb_y = random_number_between(0, grid_height - 1)`
	
	`        # Ensure the bomb does not spawn on the snake's body or the apple`
	`        WHILE (bomb_x, bomb_y) in snake_body OR (bomb_x, bomb_y) == apple_position:`
	`            bomb_x = random_number_between(0, grid_width - 1)`
		This pseudocode efficiently manages bomb placement in the game. In the `generate_new_bomb` function, bombs are spawned randomly within the game grid while ensuring they do not overlap with the snake or the apple. The algorithm achieves this by repeatedly generating random coordinates until finding a suitable location that does not coincide with the snake's body or the apple's position. This meticulous process guarantees a fair and challenging gameplay experience, where bombs pose a threat without unfairly obstructing the player's movement. Additionally, the `check_bomb_collision` function promptly detects

	bomb_y = random_number_between(0, grid_height - 1) RETURN (bomb_x, bomb_y) PUBLIC FUNCTION check_bomb_collision(snake_head): IF snake_head == position: game_over() PUBLIC FUNCTION spawn_new_bomb(): IF score MOD bomb_spawn_threshold == 0: position = generate_new_bomb_position()	collisions between the snake's head and a bomb, triggering a game over scenario. Moreover, in the `check_apple_eaten` function, bombs are spawned when the player's score reaches a predefined threshold, introducing strategic elements and enhancing the game's difficulty as players progress.

Registration system	# List to store registered usernames
	registered_usernames = []
	
	# File path for storing student information
	file_path = "students_data.txt"
	
	#register a new student
	FUNCTION register_student(username, password):
	IF is_username_available(username):
	# Create a new student account
	create_student_account(username, password)
	display_message("Registration successful. Welcome, " + username + "!")
	save_student_data_to_file(username, password)
	ELSE:
	display_message("Username already taken. Please choose another.")
		The `register_student` function verifies username availability, updates the list of registered users, and saves student data to a file using the `save_student_data_to_file` function. This file handling approach ensures a straightforward yet effective means of persistent data storage for the system.

	ENDFUNCTION
	
	#check if a username is available
	FUNCTION is_username_available(username):
	    IF username not in registered_usernames:
	        RETURN True
	    ELSE:
	        RETURN False
	ENDFUNCTION
	
	#create a new student account
	FUNCTION create_student_account(username, password):
	    # Store the username in the list of registered usernames
	    registered_usernames.append(username)
	ENDFUNCTION
	
	#save student data to a file
	FUNCTION save_student_data_to_file(username, password):
	    # Open the file in append mode
	    file = open(file_path, "a")
	ENDFUNCTION
	
	    # Write the student data to the file
	    file.write(username + "," + password + "\n")
	
	    # Close the file
	    file.close()
	
	#display a message to the user
	FUNCTION display_message(message):
	    PRINT(message)
	ENDFUNCTION	

Log in system	# List to store registered usernames
	registered_usernames = []
	
	# File path for storing student information
	file_path = "students_data.txt"
	
	# Function to authenticate a student
	FUNCTION authenticate_student(username, password):
	    student_data = read_student_data_from_file()
		This pseudocode outlines an efficient login system that retrieves student information from a file (`students_data.txt`) for authentication. The `authenticate_student` function checks entered credentials against stored

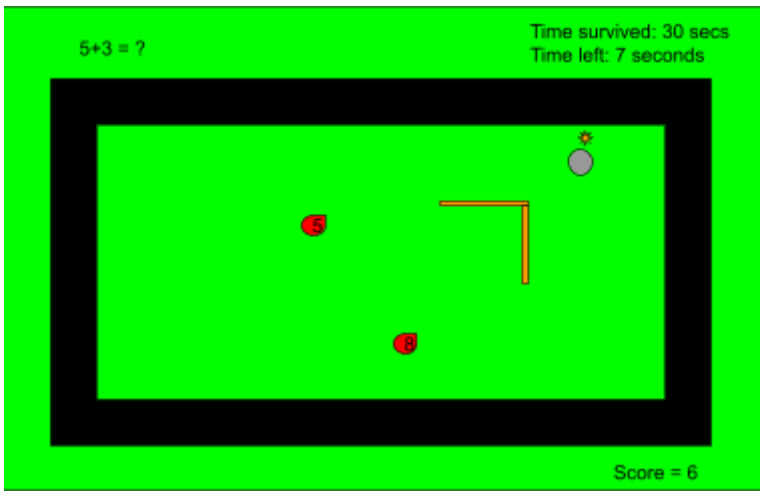
	FOR student in student_data:
	if student["username"] == username and student["password"] == password:
	display_message("Login successful. Welcome, " + username + "!")
	RETURN
	
	display_message("Invalid credentials. Please check your username and password.")
	ENDFUNCTION
	
	#reads student data from the file
	FUNCTION read_student_data_from_file():
	try:
	with open(file_path, "r") as file:
	RETURN [line.strip().split(",") for line in file.readlines()]
	except FileNotFoundError:
	RETURN []
	ENDFUNCTION
	
	#display a message to the user
	function display_message(message):
	PRINT(message)
		records, displaying a welcome message upon success or prompting the user to check their username and password.

Leaderboard	# List to store leaderboard entries
	leaderboard = []
	
	#add a new entry to the leaderboard
	function add_leaderboard_entry(username, score):
	leaderboard.append({"username": username, "score": score})
	
	#display the leaderboard
	function display_leaderboard():
	sorted_leaderboard = sorted(leaderboard, key=lambda x: x["score"], reverse=True)
	display_message("Leaderboard:")
	
	FOR index, entry in enumerate(sorted_leaderboard[:3]):
	display_message(f"{index + 1}. {entry['username']}: {entry['score']}")
	
	#display a message to the user
	FUNCTION display_message(message):
		This concise pseudocode presents a leaderboard system with minimalistic yet effective functionalities. The `add_leaderboard_entry` function appends new entries to the leaderboard list, which consists of dictionaries containing usernames and scores. The `display_leaderboard` function sorts the leaderboard by score, displays the top three entries, and leverages list comprehensions for brevity

	PRINT(message) ENDFUNCTION	

2.4 Usability features

How the game would look like:



This is a simplification of what the game would look like when playing it. There are lots of components in this game, the main one being objects. There is also the aesthetics aspect.

Objects:

Snake:

- The primary player-controlled entity in the game is the snake.
- It represents the main character and is integral to gameplay.
- The snake's movement and interactions with other objects form the core mechanics.

Apples:

- Apples serve as a fundamental gameplay element and a positive reinforcement for the player.
- They represent a target for the snake to "eat" or collect, contributing to score and growth.
- The presence of apples encourages strategic movement and decision-making for the player.

Bomb:

- Introducing a bomb adds an element of challenge and risk to the gameplay.
- Bombs act as obstacles that the player must avoid to prevent game-ending consequences.

By including these objects, you create a dynamic and engaging game environment. The snake's pursuit of apples drives positive gameplay, while the presence of bombs introduces a challenging aspect, enhancing overall user experience and providing a balanced and entertaining gaming scenario.

Aesthetics:

Math Questions (Top Left Corner):

- Purpose: Introducing maths questions adds an educational aspect to the game, aligning with the goal of creating a maths-focused experience for primary school kids.
- Engagement: Players are prompted to apply mathematical skills, promoting learning while playing.
- Integration: By incorporating questions directly into the gameplay, the game seamlessly combines entertainment with educational content.

Time Display (Top Right Corner):

- Survival Time: Displays the time the player has survived, emphasising the endurance aspect of the game.
- Time Left for Answering: Creates urgency and adds a time management component, encouraging quick thinking and decision-making.
- Enhanced Challenge: The time elements contribute to the game's difficulty, making it more engaging and challenging for players.

Score Display (Bottom Right Corner):

- Motivation: Constantly visible score encourages players to strive for improvement and beat their own records.
- Progress Tracking: Allows players to gauge their performance and see the direct impact of correct answers on their score.

- Positive Reinforcement: The score serves as positive reinforcement for successful interactions with maths questions, motivating players to continue playing.

Difficulty increase

-To increase the difficulty we could decrease the “time left “. This is so that the player will have to perform faster calculations due to there being less time to select the correct answer.

-Increase the snake's length in order to decrease the area for the snake to manoeuvre. This is only up to a certain extent as in this case there is more than one apple so we could for example increase it so that it adds a max of 10 segments on its body.

-Bombs objects increasing the chance of collision.

When a game ends:

Final Score Display:

- Feedback: Showing the final score provides immediate feedback to the player, summarising their performance in the completed game session.
- Motivation: The displayed score serves as motivation for improvement in subsequent rounds, encouraging continued engagement.

"Better Luck Next Time" Message:

- Encouragement: The message contributes to a positive player experience, softening the impact of game-over scenarios.

- Resilience: It acknowledges the possibility of not achieving the desired outcome while instilling a sense of optimism and encouragement to try again.

Navigation Buttons (Return to Main Menu, Play Again, Mute Music):

- User Control: Offering multiple buttons provides users with control over their post-game experience, allowing them to choose the next action based on preference.
- Accessibility: The "Return to Main Menu" option allows users to navigate to different sections of the game, while "Play Again" enables a quick restart for those eager to try again.
- User-Friendly: Including a mute option demonstrates consideration for player preferences, enhancing the overall user experience.mjk,

Permanent storage

In this project, a CSV (Comma-Separated Values) file will be utilised as a temporary storage solution to store username and password data for registered students.

Usage of CSV File:

Temporary Storage:

- The CSV file acts as a temporary storage medium to hold the registration data for students.
- It facilitates easy access and manipulation of user information without the need for a full-fledged database.

Read Operation:

- The program reads data from the CSV file to check and authenticate user logins during the login process.
- The CSV file is read to retrieve stored usernames and passwords for comparison with user inputs.

Write Operation:

- During the registration process, the program writes new user data (username and password) to the CSV file.

- This allows for the dynamic addition of new users without the need for complex database management.

Why use CSV files?

-Simplicity: Using a CSV file provides a straightforward and lightweight solution for storing and retrieving user data.

hThe CSV approach allows for easy modification and inspection of user data without the need for a dedicated database system.

-Ease of Integration: CSV operations are seamlessly integrated into the Python programming environment, ensuring a smooth implementation process.

-Temporary Nature: The CSV file serves as a temporary storage solution, suitable for small-scale applications or projects with simplicity as a priority.

2.5 Variables and validations

In this section I will outline the key data types and data structures for both the game and log in scripts that I aim to have when development starts.

Game Inputs Validation:

Buttons and Textboxes Validation:

For the game interface, Pygame's robust capabilities will be instrumental in implementing comprehensive input validation. Pygame, known for its resilience and versatility, offers a solid foundation for handling user inputs with precision. By harnessing Pygame's inherent robustness, I will enforce strict validation checks to ensure accurate processing of user interactions within the game environment. This proactive approach not only enhances the gameplay experience but also reinforces the integrity and fairness of user inputs, contributing to an immersive and enjoyable gaming experience.

Login Inputs Validation:

Buttons and Textboxes Validation:

In the login system, I will rely on Tkinter, renowned for its robust nature, to handle input validation seamlessly. Tkinter's inherent robustness ensures that buttons and textboxes are efficiently validated, enhancing user interaction and data integrity. By utilising Tkinter's extensive features, I aim to implement stringent validation checks, guaranteeing secure and error-free user

inputs. This approach not only ensures the reliability of the login interface but also contributes to a smoother and more intuitive user experience.

Game:

Data types:

Variable	Data Type	Explanation	Justification of data type
score	Integer	Tracks users overall performance	Using an integer is suitable for tracking whole numbers, which represent the user's cumulative score throughout the game. Integers are efficient for arithmetic operations and memory usage, making them ideal for this purpose.
survival_time	Float	Measures how long the player has survived in seconds	A float data type is appropriate for representing decimal numbers, such as the elapsed time in seconds. Floats provide precision for time measurements and allow for fractional values, enabling accurate tracking of survival time during gameplay.
time_left_for_answer	Float	Establishes the limit for answering each maths question	Similar to survival_time, a float data type is used to represent time intervals with fractional seconds. Using floats allows for flexibility in setting precise time limits, accommodating varying durations for answering questions.
current_question	string	Stores the current	Strings are suitable for

		maths question	storing textual data, such as question prompts. They provide versatility for handling alphanumeric characters and facilitate easy manipulation and display of text-based information within the game.
is_game_over	Boolean	Indicates whether the game has ended or not	Booleans are logical data types representing true or false values. They are well-suited for representing binary states, such as game over or game ongoing. Booleans simplify conditional checks and decision-making processes within the game logic.
snake_length	Integer	Tracks the current length of the snake	snake_length cannot be a float because the length of the snake is inherently discrete and cannot be represented as a fractional value. Each segment of the snake occupies a fixed unit of space on the grid, making it impossible to have fractional lengths
snake_direction	String	Tracks the current direction of the snake (e.g., "UP", "DOWN", "LEFT", "RIGHT").	Representing the direction as a string simplifies the logic for handling snake movement and makes it easier to understand and implement.
correct_apple_eaten	Boolean	If the correct apple has been eaten score is incremented while time_left is reset	A Boolean data type is appropriate for "correct_apple_eaten" because it only needs to represent two states: True

			(indicating the correct apple has been eaten) or False (indicating otherwise). Using a Boolean simplifies the logic for tracking the state of the game and enables straightforward conditional checks.
bomb_collision	Boolean	This variable indicates whether the snake has collided with a bomb in the current game cycle. When set to True, it triggers actions such as ending the game or reducing the player's score.	It simplifies the logic and allows for straightforward conditional checks to handle the game's behaviour based on this event.
rock_collision	Boolean	This variable indicates whether the snake has collided with a rock in the current game cycle. When set to True, it triggers actions such as ending the game or reducing the player's score.	It simplifies the logic and allows for straightforward conditional checks to handle the game's behaviour based on this event.

Data Structures:

Variable	Data structure	Explanation	Justification
snake_body	Tuple	Represents the segments of a snake	Using a list of arrays allows for a dynamic addition of new segments
game_options	Dictionary	Stores the available	Using a dictionary

		game options (e.g., modes like addition, subtraction, multiplication, division) mapped to their respective settings.	allows for efficient lookup of game options based on the selected mode. It provides a convenient way to organise and manage different game settings.
game_state	Dictionary	Holds the current state of the game, including variables such as the player's score, the snake's position, and the positions of fruits and obstacles.	Storing the game state in a dictionary enables easy access and modification of game variables during gameplay. It provides a structured approach to manage the game's dynamic elements.
leaderboard	Tuple	Stores the available game options (e.g., modes like addition, subtraction, multiplication, division) mapped to their respective settings	Using a dictionary allows for efficient lookup of game options based on the selected mode. It provides a convenient way to organise and manage different game settings
Bomb_positions	List of Tuples	Stores the positions of bombs on the game grid.	Tuples are immutable and can represent fixed data points like coordinates, making them suitable for storing positions on the grid. This data structure enables

			easy detection of collisions between the snake and bombs, triggering appropriate actions such as ending the game or reducing the player's score.

Class diagram for game objects:

Class	Snake
Attributes	`body_length , direction`
Methods	`move_snake, grow_snake, render`

Summary for Snake:

Attributes:

body_length: Represents the current length of the snake.

direction: Stores the current direction of movement for the snake.

Methods:

move_snake: Responsible for moving the snake in its current direction.

grow_snake: Handles the logic for growing the snake when it eats an apple.

render: Renders the snake on the game screen.

Explanation:

The Snake class encapsulates the behaviour of the player-controlled snake in the game. The body_length attribute keeps track of the number of segments in the snake's body, while the direction attribute determines the snake's movement direction. The

move_snake method updates the position of each segment according to the current direction, while grow_snake manages the logic for adding new segments when the snake eats an apple. The render method handles the visualisation of the snake on the game screen.

Justification:

-Using body_length as an attribute allows for easy tracking of the snake's length, which is essential for collision detection and rendering.

-The direction attribute simplifies the control of the snake's movement, as it only needs to store one direction at a time.

-The move_snake method encapsulates the core functionality of the snake's movement, promoting modular code design.

-The grow_snake method provides a clean interface for extending the snake's length, separating concerns and improving code readability.

-The render method ensures separation of concerns by handling the visualisation of the snake, adhering to the principle of encapsulation

Class	Bomb
Attributes	`position`
Methods	`explode, render,spawn`

Summary for Bomb:

Attributes:

position: Represents the current position of the bomb on the game grid.

Methods:

explode: Triggers the explosion animation and effects when the bomb detonates.

render: Draws the bomb on the game screen during gameplay.

spawn: Generates a random position for the bomb when it needs to appear in the game.

Explanation:

The Bomb class encapsulates the functionality related to bombs in the game. The position attribute stores the coordinates of the bomb's current location on the game grid. The explode method handles the detonation of the bomb, initiating the explosion animation and any associated effects. The render method is responsible for displaying the bomb on the game screen, ensuring it is visually represented during gameplay. Lastly, the spawn method generates a random position for the bomb to appear in the game, adding an element of unpredictability to gameplay.

Justification:

-Using the position attribute allows for easy tracking and manipulation of the bomb's location within the game environment.

-The explode method abstracts the explosion logic, promoting code organisation and readability by encapsulating related functionality.

-Separating the rendering logic into the render method enhances code modularity and maintainability, facilitating easier updates or modifications to the visual representation of the bomb.

-The spawn method adds variability to gameplay by randomly determining the bomb's initial position, contributing to the overall challenge and excitement of the game experience.

Class	Apple
Attributes	`position`
Methods	`render,spawn`

Summary for Apple:

Attributes:

position: Represents the current position of the apple on the game grid.

Methods:

render: Draws the apple on the game screen during gameplay.

spawn: Generates a random position for the apple when it needs to appear in the game.

Explanation:

The Apple class manages the functionality associated with apples in the game. The position attribute stores the coordinates of the apple's current location on the game grid. The render method is responsible for visually displaying the apple on the game screen, ensuring it is visible during gameplay. The spawn method generates a random position for the apple to appear in the game, adding an element of randomness and unpredictability to the game environment.

Justification:

-The position attribute enables easy tracking and manipulation of the apple's location within the game environment, facilitating interaction with other game elements.

-Separating the rendering logic into the render method enhances code modularity and maintainability, allowing for easier updates or modifications to the visual representation of the apple.

-The spawn method introduces variability to gameplay by randomly determining the initial position of the apple, contributing to the challenge and excitement of the game experience.

Login and registration system:

Here I have outlined the key data types and data structures for the account functionality system.

Variable	Data Type	Explanation	Justification
user	String	Represents the	Usernames are

		username input by the user	typically strings of characters.
code	String	Represents the password input by the user	Passwords are often stored as strings for security.
confirm_code	String	Represents the confirmed password input by the user	Similar to the password, it's stored as a string.
data	String	Represents the data read from the file	Data read from files is often stored as strings.
user_data	Dictionary	Represents user data stored in the file	Dictionaries are useful for storing key-value pairs

Additional Variables:

These variables are assumed to be used in conjunction with the Tkinter module. This is a python module that is used for creating GUIs.

Variable	Data Type	Explanation	Justification
Icon_img	N/A	Represents an	Used to load and

		image object for icon display	manipulate image data.
window	N/A	Represents the main application window	Tkinter's `Tk` class represents the application window.
window2	N/A	Represents a secondary window for PIN entry	Another instance of the Tkinter window.
pin_entry	N/A	Represents an entry widget for PIN input	Tkinter's `Entry` widget is used for text input.
result_label	N/A	Represents a label widget for displaying PIN validation	Tkinter's `Label` widget is used for displaying text.
button_frame	N/A	Represents a frame widget for number buttons	Tkinter's `Frame` widget is used for layout purposes.
img	N/A	Represents an image object for registration image	Tkinter's `PhotoImage` class is used for image display.

frame	N/A	Represents a frame widget for layout organisation	Tkinter's `Frame` widget is used for layout purposes.
heading	N/A	Represents a label widget for headings	Tkinter's `Label` widget is used for displaying text.
user	N/A	Represents an entry widget for username input	Tkinter's `Entry` widget is used for text input.
code	N/A	Represents an entry widget for password input	Tkinter's `Entry` widget is used for text input.
confirm_code	N/A	Represents an entry widget for confirming password	Tkinter's `Entry` widget is used for text input.
register	N/A	Represents a button widget for registration	Tkinter's `Button` widget is used for user interaction.
label	N/A	Represents a label widget for displaying text	Tkinter's `Label` widget is used for displaying text.

Variable	Data Structure	Explanation	Justification
user_data	Dictionary	Stores registered usernames and passwords	Using a dictionary allows for efficient lookup of passwords based on usernames during login. It provides a simple and effective way to associate usernames with their corresponding passwords, facilitating authentication.
user_input	String	Stores the username entered by the user	A string data type is appropriate for storing textual input such as usernames. It allows for easy manipulation and validation of user-provided data.
pass_input	String	Stores the password entered by the user	Similar to the username, a string data type is suitable for storing passwords as textual input. It enables the program to handle password entry and validation securely.
PIN	String	Stores the PIN for accessing certain functionalities	Using a string to represent the PIN allows for straightforward comparison and validation. It ensures that the PIN entered by the user can be easily compared with the correct PIN for

			authorization purposes.

Classes Diagrams for login and register:

Authentication class

Class	**Authentication**
Attribute	users: List[User]
Methods	__init__() register(username,password) login(username,password)

Attributes:

1)users: List[User]

-**Explanation**: This attribute holds a list of User objects, representing all registered users in the system.

-**Justification**: Storing user objects in a list facilitates easy access to user information and enables efficient user management operations like registration and login.

Methods:

1)_init_()

Explanation: The constructor method initialises a new instance of the Authentication class.

Justification: The constructor sets up the initial state of the Authentication class, ensuring proper initialization before performing any authentication operations.

2)register(username, password)

Explanation: This method registers a new user by creating a User object with the provided username and password and adding it to the list of users.

Justification: By providing a user registration functionality, the system allows new users to create accounts and access the application securely.

login(username, password)

Explanation: This method authenticates a user by checking the provided username and password against the stored user data.

Justification: User authentication is a fundamental security measure, ensuring that only authorised users can access the system's functionalities and data.

User Class:

Class	user
Attributes	username: string password: string

Methods	`__init__(username, password)`
	`get_username(): str` `set_username(new_username: str)`
	`get_password(): str` `set_password(new_password: str)`
	|

Attributes:

1)username: string

Explanation: This attribute stores the username of a user.

Justification: The username uniquely identifies each user and is essential for user authentication and identification within the system.

2)password: string

Explanation: This attribute stores the password of a user.

Justification: Storing user passwords securely is crucial for protecting user accounts from unauthorised access and ensuring data confidentiality.

Methods:

1)_init_(username, password)

Explanation: The constructor method initialises a new instance of the User class with the provided username and password.

Justification: Initializing user objects with username and password parameters allows for the creation of user instances with specific credentials.

2)get_username(): str

Explanation: This method retrieves the username of the user.

Justification: Providing a way to access the username allows other parts of the system to identify and interact with users as needed.

3)set_username(new_username: str)

Explanation: This method updates the username of the user with a new value.

Justification: Offering the ability to update the username enables users to modify their account information, improving user experience and flexibility.

4)get_password(): str

Explanation: This method retrieves the password of the user.

Justification: Although retrieving passwords is generally discouraged for security reasons, this method might be necessary for specific use cases within the system.

5)set_password(new_password: str)

Explanation: This method updates the password of the user with a new value.

Justification: Allowing users to change their passwords regularly enhances account security and mitigates the risk of unauthorised access due to compromised credentials.

Validation

Validation ensures that the data input into a system meets certain criteria or standards, thereby maintaining the integrity and reliability of the system. Here's an explanation and justification for the validations.

1)Ensuring non-negative values for critical game-related variables:

Explanation: This validation ensures that critical variables such as score, survival time, and time left for answer are always non-negative. By preventing these variables from having negative values, we avoid potential errors or unexpected behaviours in the game logic that could arise from such values.

Justification: Implementing this validation adds resilience to the game by handling potential negative values. It ensures that the game remains stable and functional even if unexpected conditions occur during gameplay, enhancing the overall user experience.

Example: Ensuring non-negative values for score, survival time, and time left for answer

```
if score < 0:
    score = 0

if survival_time < 0:
    survival_time = 0

if time_left_for_answer < 0:
    time_left_for_answer = 0
```

2)Password Strength Validation:

Explanation: Password strength validation ensures that user passwords meet certain security standards, such as minimum length, inclusion of alphanumeric characters, and use of special characters. By enforcing strong password requirements, we reduce the risk of unauthorised access to user accounts through password guessing or brute-force attacks.

Justification: Implementing password strength validation enhances the overall security of user accounts. By requiring users to create strong passwords, we mitigate the risk of account compromise and protect sensitive user data from unauthorised access, ensuring a secure user experience.

```
# Example: Implementing password strength validation

import re


def is_strong_password(password):

    # Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit

    return len(password) >= 8 and re.search(r'[A-Z]', password) and re.search(r'[a-z]', password) and re.search(r'[0-9]', password)


# Usage:

password = input("Enter your password: ")

if not is_strong_password(password):

    print("Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit")
```

3)Error Handling:

Explanation: Error handling mechanisms are implemented to detect, report, and handle errors or exceptions that may occur during the registration or login process. This includes providing informative error messages to users and gracefully handling unexpected situations to prevent application crashes or data corruption.

Justification: Robust error handling mechanisms are essential for providing a smooth and user-friendly experience. By providing meaningful feedback to users in case of registration or login failures, we improve user satisfaction and reduce frustration.

Additionally, proper error handling contributes to the overall stability and reliability of the application, ensuring uninterrupted operation even in the presence of errors.

Example: Handling errors during user registration

```
try:
    register_user(username, password)
    print("Registration successful!")
except RegistrationError as e:
    print(f"Registration failed: {e}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

GUI Validation:

Outside the main program functionality we must ensure that the GUI is robust so that overall all the inputs in the program are validated this is to ensure users are inputting the correct data and all user inputs can be validated

1)Buttons:

Explanation: I'll use buttons to create interactive elements in the GUI, allowing users to trigger actions by clicking on them. In the game, I'll implement buttons using Pygame, while buttons for the login and registration system will be part of Tkinter or another GUI library. When clicked, each button will be linked to a specific function. Although clicking buttons doesn't directly involve input validation, ensuring that each button is correctly linked to its corresponding function is essential for the overall functionality of the application.

Justification: Proper button functionality ensures smooth navigation and interaction within the application. While buttons themselves don't accept user input, validating their behaviour helps prevent potential issues such as broken links or unresponsive buttons, enhancing the user experience.

```
#Example python script
import pygame

def button_click():
    # Functionality to execute when the button is clicked
    pass

pygame.init()
screen = pygame.display.set_mode((800, 600))
button = pygame.Rect(300, 200, 200, 50)
pygame.draw.rect(screen, (255, 0, 0), button)
pygame.display.flip()

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.MOUSEBUTTONDOWN:
            if button.collidepoint(event.pos):
                button_click()
```

```
pygame.quit()
```

2)Text Boxes:

Explanation: Text boxes will allow users to input text, making them crucial for collecting user data such as usernames and passwords. I'll validate text input to ensure that the entered data meets the required format or criteria, preventing errors and ensuring data integrity.

Justification: Proper validation of text boxes helps maintain the correctness and consistency of user-provided information. In the game settings, text boxes may limit input to single characters valid for keyboard keys and enforce correct case usage to prevent compatibility issues.

#Example python script

```
import pygame


def validate_input(input_text):
    # Check if input_text is a valid single character
    return len(input_text) == 1 and input_text.isalpha()


pygame.init()
screen = pygame.display.set_mode((800, 600))
font = pygame.font.Font(None, 36)
input_text = ''
```

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RETURN:
                if validate_input(input_text):
                    print("Valid input:", input_text)
                else:
                    print("Invalid input!")
                input_text = ''  # Reset textbox after validation
            elif event.key == pygame.K_BACKSPACE:
                input_text = input_text[:-1]
            else:
                input_text += event.unicode

    screen.fill((255, 255, 255))
    text_surface = font.render(input_text, True, (0, 0, 0))
    screen.blit(text_surface, (300, 200))
```

```
    pygame.display.flip()


pygame.quit()
```

Testing Method

For testing the snake game application, a comprehensive approach that encompasses both unit testing and user acceptance testing is recommended.

Unit Testing:

Start with unit testing to verify the functionality of individual components in isolation. Break down the game logic into modular components such as the Snake class, Apple class, and grid rendering functions. Implement unit tests for each component, validating that they perform their designated tasks correctly.

For example, you can create unit tests to:

-Verify that the Snake class moves correctly in response to user input.

-Ensure that the Apple class spawns in valid locations on the game grid.

-Test the rendering functions to confirm that they draw the game elements accurately on the screen.

User Acceptance Testing (UAT):

Following unit testing, proceed to User Acceptance Testing to evaluate the game from an end-user perspective. Engage actual users or testers to play the game and provide feedback.

During UAT, focus on aspects such as:

-User interface responsiveness: Test how the game interface behaves on different devices and screen sizes.

-Game mechanics: Assess the gameplay experience to ensure it aligns with user expectations and is enjoyable.

-Overall user experience: Evaluate factors like game controls, graphics, and sound effects to determine if they enhance the gaming experience.

Identify and address any usability issues, bugs, or unexpected behaviours that may arise during UAT. Document and analyse user feedback to refine the game further, ensuring a polished and enjoyable gaming experience.

For example, during UAT, you could ask testers to play through different levels of the game and provide feedback on their experience. You might receive comments on the responsiveness of the controls, the clarity of on-screen instructions, or the difficulty level of the game.

2.6 Iterative test data (test plan)

I plan on now making 6 milestones I aim to reach while developing my game. The test description, test data alongside the expected outcome and the actual outcome that happens after developing the game with cross reference with the iterative development section

Milestone 1:

In this milestone I aim to make a prototype of the actual game. Here I am just going to make a normal working snake game without any maths functionalities

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome
1	Load Background image	Typical	Background image currently being loaded and updated while the game is being run
2	Load snake icon	Typical	Snake icon being correctly loaded
3	Load Game title	Typical	Title of the game being displayed

			next to the icon
4	Load snake	Typical	Snake correctly being loaded
5	Load apple	Typical	Apple currently being loaded
6	Make boundary That player cant cross	Typical boundary:ensures that the boundary of the game area is correctly defined to prevent the snake from crossing it	Ensuring the snake won't cross the boundary
7	Check movement	Typical Boundary: verifies that the snake can move both horizontally and vertically within the defined boundary.	Ensuring the snake can move horizontally and vertically
8	Have a score function	Typical	Ensuring the score is correctly incremented as you eat the apple
9	Adding sound effects when eating apple	Typical	Sounds should be heard when eating the apple
10	Adding music	Typical	Music should be playing while the game is running

Milestone 2:

In this milestone I aim to iterate upon the prototype. Here I am just going to add all the maths functionality and check everything works correctly.

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome
1	Maths question correctly displayed	Typical	The maths question is visible and correctly displayed to the player.
2	Two apple fruit objects created	Typical	Two apple objects are visible and correctly displayed on the game grid.
3	Time running function displayed	Typical	The timer is visible and continuously updates to reflect the elapsed time
4	Time left function displayed	Typical	The countdown timer is visible and accurately reflects the time remaining
5	Score incremented if correct answer chosen	Typical	The player's score is incremented by the specified amount.
6	Two apple objects containing different answers	Typical Boundary: This test ensures that the game correctly handles the	Each apple object displays a different answer, allowing the player to select the correct one

		scenario where two apple objects contain different answers, allowing the player to select the correct one. It tests the boundary condition of apple objects having distinct values for the maths questions	

Milestone 3:

In this milestone I aim to iterate upon the milestone 2 and just add the drop functionalities to make the game harder.

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome
1	Bomb loaded onto the screen	Typical	Bomb correctly loaded
2	Bomb spawns random locations	Typical	Bomb correctly spawning different places
3	Bomb explodes when touched	Typical	Bomb correctly exploding once touched
4	Rock correctly loaded	Typical	Rock correctly being loaded onto the screen

5	Rock spawns different locations	Typical	Rock correctly spawning different locations
6	Rock collision with snake ends the game	Typical	Rock correctly ending the game if snake collides with it
7	Length increasing to a certain point	Typical Boundary: This test ensures that the snake's length visibly increases on the game grid until it reaches the predetermined length threshold. It tests the boundary condition of the snake's length reaching a specific point without exceeding it	The snake's length visibly increases on the game grid until it reaches the predetermined length threshold

Milestone 4:

In this milestone I aim to iterate upon the milestone 3 and just add different menu options such as going back to the main menu/ muting the music / playing again

Test No	**Test Description/Purpose and justification**	**Test Data (Erroneous, boundary, typical)**	**Expecting outcome**
1	Load mute image	Typical	Correctly image has been loaded

2	Load play again image	Typical	Correctly image has been loaded
3	Load home button	Typical	Correctly image has been loaded
4	Ensure mute button works	Typical	Button correctly works
5	Ensure play again button works	Typical	Button correctly works
6	Ensure home button works	Typical	Button correctly works
7	Test GUI buttons	Typical	All the buttons for all the modes should work

Milestone 5:

In this milestone I aim to create a database that will correctly update the leaderboard.

Test No	**Test Description/Purpose and justification**	**Test Data (Erroneous, boundary, typical)**	**Expecting outcome**
1	Create a leaderboard	Typical	Leaderboard correctly created
2	Update leaderboard	Typical	Leaderboard successfully updated

Milestone 6:

In this milestone I aim to create the account functionality. This involves creating the register log in scripts and ensuring they do what they are supposed to do.

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome
1	Create register window	Typical	Window successfully created
2	Create buttons for register	Typical	Button successfully created
3	Add icon for the register page	Typical	Icon successfully added
4	Ensure the buttons function correctly	Typical	Button function works
5	Ensures user types in an appropriate username and password	Typical	Ensures username entered meets a certain criteria
6	Ensure passwords have to match for account to be created	Typical	Ensure access is granted only if the passwords match
7	Ensured the details are saved to an external file	Typical	File are correctly saved to an external file
8	Ensures you can access login page from the register page	Typical	Correctly allows access to register page from login page
9	Ensure only admin can register	Typical	Only allows admin to register
10	Ensure can register only if correct pin is entered	Typical	Only allows registration to occur if the correct pin is entered.
11	Create login window	Typical	Log window successfully

			created
12	Create buttons for login	Typical	Buttons successfully created.
13	Add icon for the login page	Typical	Icon successfully added
14	Ensure the login button works	Typical	Login button works accordingly
15	Ensure username entered is valid	Typical	Only appropriate usernames are accepted
16	Ensure password entered is valid	Typical	Only appropriate password gives access
17	Ensure a messagebox is displayed if the details entered is correct	Typical	Displays message box if wrong data is being given.
18	Ensure you can enter the register page from the login page	Typical	You can access the register page from the login page.

2.7 Post development test data (test plan)

Test Number no	Test description/purpose and justification	Test Data	Expecting outcome	Actual Outcome
1	Evaluate overall game functionality	Typical: user navigates the snake throughout the grid, answers questions by colliding with	Snake moves appropriately, and the questions are displayed accurately, and game element	

		apples and on the same time avoiding bombs	interact correctly	
2	Check robustness of login and registration system	Erroneous: entering an incorrect password during login Boundary: inputting the maximum amount of characters for registering Typical: successfully registering an user	Proper error handling during login, appropriate response to boundary conditions and successful registration	
3	Validate effectiveness of the educational component	Typical: answering questions while it gets more difficult to get the correct answer due to the increase in the snakes length and the decrease in time left to answer	Questions should not be too hard or too easy, just hard enough for a kid to calculate the maths mentally. However, this process gets harder as time left decreases.	
4	Test the game's response to rapid and simultaneous user inputs.	Typical	Rapidly press multiple control keys or buttons simultaneously to assess the game's responsiveness and handling of concurrent	

			inputs.	
5	Evaluate the game's performance under different levels of hardware resources.	Typical	Run the game on machines with varying hardware specifications (e.g., CPU, GPU, RAM) to assess performance and optimise resource usage.	
6	Test the game's behaviour during system interruptions (e.g., notifications, updates).	Erroneous	Simulate system interruptions such as notifications or updates while playing the game to ensure it handles interruptions gracefully.	
7	Evaluate the game's performance and behaviour under varying levels of game complexity (e.g., increasing snake speed, additional obstacles).	Typical	Gradually increase the game's difficulty settings or add more challenging elements (e.g., faster snake speed, more obstacles) to assess performance and behaviour under different gameplay conditions.	
8	Verify the game's	Typical	Run the game	

	behaviour when running concurrently with other applications or processes.		alongside other applications or processes to assess its performance, resource usage, and behaviour in multitasking environments.	
9	Test the game's audio feedback system (e.g., sound effects, background music).	Typical	Enable audio feedback and assess the quality, clarity, and appropriatenes s of sound effects and background music during different gameplay scenarios.	
10	Verify the game's behaviour during transition phases .	Typical	Start the game or transition between different game scenes to ensure smooth transitions, proper initialization, and correct display of relevant information or prompts.	

3.0 Development

In order to make development easier I aim to make a working prototype(milestone 1) which will be a normal snake game and then add in the maths functionality later before moving onto the account and leaderboard system.

3.1 Milestone 1 start:

To create the game we will need to depend on lots of external resources. These will be mainly self built in python modules that will make development easier as we don't need to write functions as they are already built in. For the development of the game the main module will be pygame, a module that is used to build 2D games.

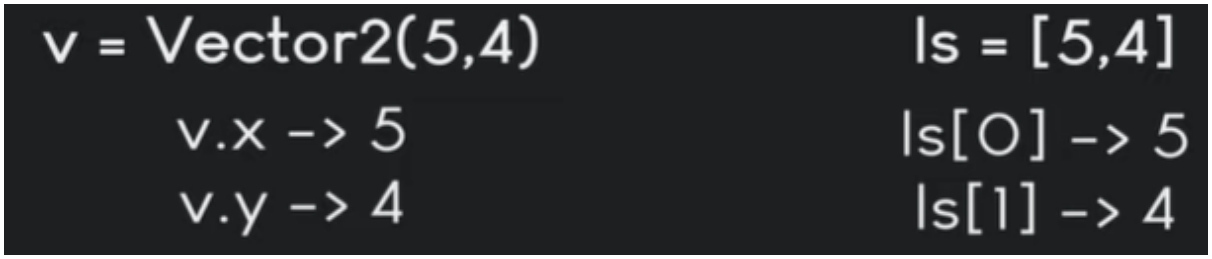
```
import pygame, sys
import random
import time
import os
from pygame.math import Vector2
```

```
#Initialization
pygame.init()
pygame.font.init()
```

- Here we imported the os module for us so that we can use the images inside my pc for the game
- The time module is responsible for adding delays to the program incase need be
- `The import sys` in Python brings in the `sys` module, providing access to variables and functions that interact with the Python interpreter. It's commonly used for system-specific functionality, including command-line arguments (`sys.argv`) and exiting the program (`sys.exit`).
- The pygame.init() initialises everything inside pygame whereas pygame.font.init() initialises everything we might need for text

However the most important use of an external library is the Vector2 module we have imported. The main use of this module is to make calculations easier which will make it easier to work with other functionalities later on the game decreasing development time and complexity.

Here is the summary of how it works and its concept:



Now it's time to actually create the main game loop which I have done below:

```
#Game Loop
FPS = 60
clock = pygame.time.Clock()
Running = True
if __name__ == '__main__':
    while Running == True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                Running = False
```

-We declared FPS at 60 so that it is compatible with most devices as it sets the frame rate not too high or low.

- We have also included the line **" if __name__== '__main__': "** as it ensures no errors occur when working with future modules.

Our next step would be to create a background which we done in the code below:

```
#Background and images
Cell_size = 40
Cell_number = 20
SCREEN = pygame.display.set_mode((Cell_number * Cell_size, Cell_number * Cell_size))
```

After creating the background I have decided to add an icon and a title of the game. The code for all of this is shown below:

```
#Initializiation
import pygame
import random
import os
import sys
from pygame.math import Vector2
pygame.init()
pygame.font.init()
#Background, Screen, Icons  and images
Cell_size = 30
Cell_number = 20
SCREEN = pygame.display.set_mode((Cell_number * Cell_size, Cell_number * Cell_size))
background_image = pygame.image.load("background.jpg")
background_image = pygame.transform.scale(background_image, (Cell_number * Cell_size, Cell_number * Cell_size))
ICON = pygame.image.load("snake_icon.png")
pygame.display.set_icon(ICON)
```

-The background image has been named “background_image” as its a naming convention and makes it easier to remember and work with for the future.

-I have also named the images and icon to what they actually are for example the icon image is “snake_icon.png” which not only makes it easier to look for and import but also look back and find easily. This also increases the readability of the code for an outsider who might not have these image files but can still guess what they are just by their name

-The icons name is named “ICON” as its a constant and wont be changed later and in python anything that wont be changed later is capitalised for convention

- I have also separated the imports with the setting up screen functionality by comments. This is so it makes it easier for you to come back and make any changes to the code.

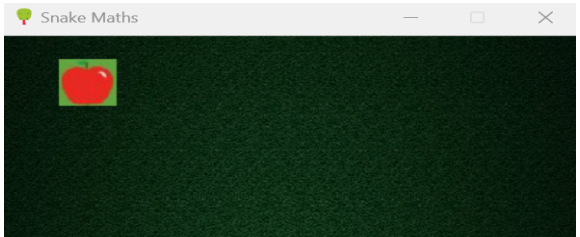
The result:

We are now going to create the classes. The main two classes in the game will be the FRUIT class and the snake class

```
#RGB values
ORANGE = (251,153,2)

#Classes
fruit_image = pygame.image.load("apple.jpg")
class FRUIT:
    def __init__(self):
        self.x = random.randint(0, Cell_number - 1)
        self.y = random.randint(0, Cell_number - 1)
        self.position = Vector2(self.x,self.y)

    def draw(self):
        fruit_rect = pygame.Rect(self.position.x * Cell_size, self.position.y * Cell_size, Cell_size, Cell_size)
        SCREEN.blit(fruit_image, fruit_rect)

class SNAKE:
    def __init__(self):
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]

    def draw(self):
        for part in self.body:
            parts_rect = (part.x * Cell_size, part.y * Cell_size, Cell_size, Cell_size)
            pygame.draw.rect(SCREEN, ORANGE, parts_rect)



#Calling classes
fruit = FRUIT()
snake  = SNAKE()
```

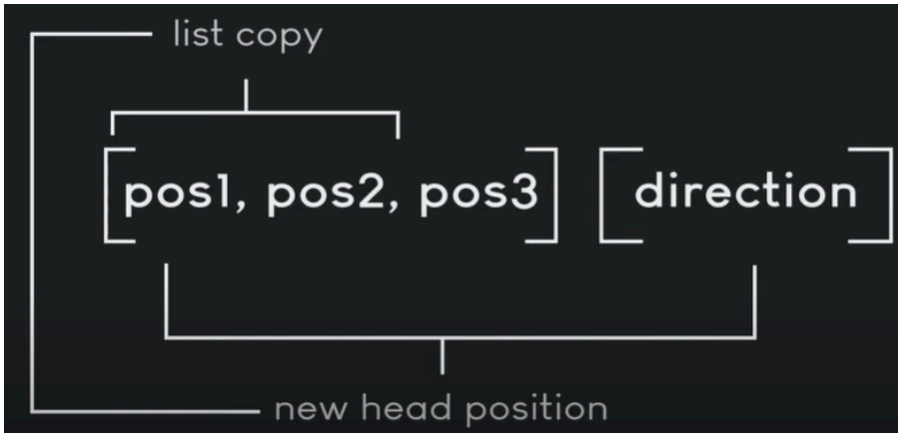
-This functionality worked perfectly but it was not pleasing to the eye. This is because although I mentioned graphics wasn't a huge part of the overall code I still wanted to enhance the user experience. After a bit of research the solution was pretty simple I had to change **`pygame.draw.rect(SCREEN, ORANGE, parts_rect, 0, 7)`. The change has been written in red**

The result:



However, I ran into a problem which was to simulate the movement of the snake. After a bit of research I came up with the cunning idea. Assuming that the head moves right, the rest of the body parts of the snake will do the same. This implies that I can just move the head of the snake and let the next body part of the snake take the head's position. This continues occurring with each block where the next block takes each block's previous position (this deletes the last block).

The concept is:



The coded solution to the problem:

```
def movement(self):
    body_copy = self.body[:-1]
    body_copy.insert(0, body_copy[0] + self.direction)
    self.body = body_copy[:]
```

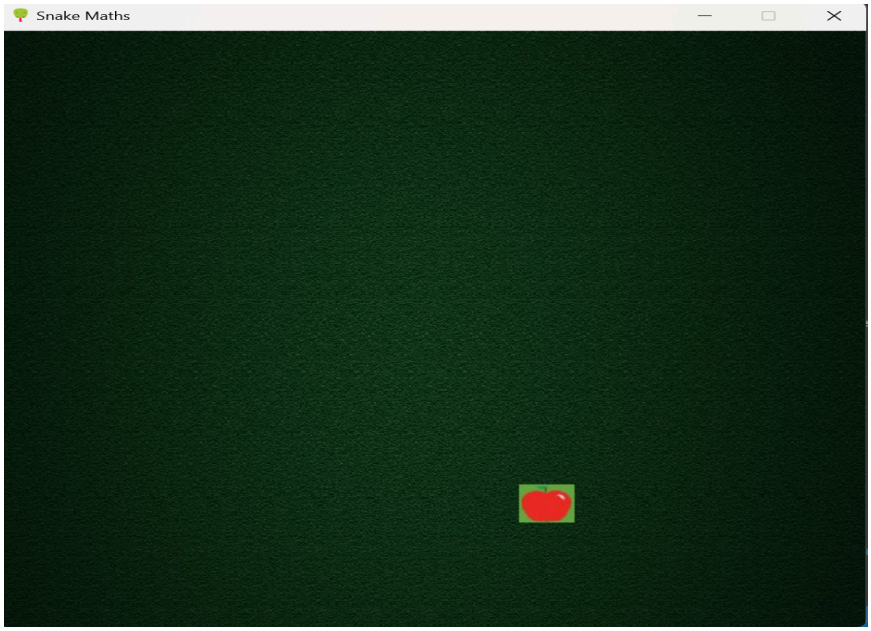
What this code does, it creates a copy of the whole snake body including each part except the last. We then add one element right at the front which is going to be the first element of the previous list and specify the direction we are going to be going in which four our case is

```
self.direction = Vector2(1,0)
```

we than set self.body equal to the body_copy so that we are returning the entire list to our original body as this is the one we are going to draw

PROBLEM 1

-This part of the code however created an issue as it was moving way too fast. This is due to the FPS which was set at 60 frames per second this means the game loop is updated 60 times per second. This can be seen in the screenshot below where the snake has seemed to have disappeared but in reality it just escaped the screen.



CODED SOLUTION 1

We could declare a constant and set it as “pygame.USEREVENT” which is a special event in pygame that allows programmers to create custom events. We could implement a timer allowing the event to be triggered once every 150 milliseconds.

```
SNAKE_UPDATE = pygame.USEREVENT

pygame.time.set_timer(SNAKE_UPDATE, 150)
```

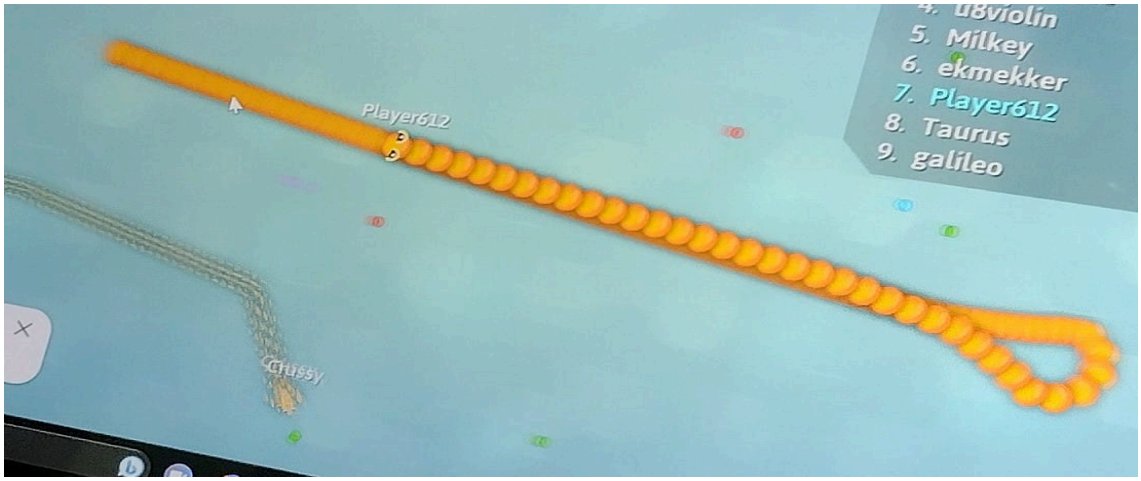
Movement functionality:

Movement in pygame is pretty simple as it works by looping through the events in the game loop using for events in pygame.event.get(). You check the event type to determine if it's related to user input such as a key press. Most pygame programmers use arrows but in my program I will allow the usage of “WASD” keys if for some reason the computer that the student was on had some arrow keys missing he/she could still play the game using other keys.

Problem 2:

The user however is now able to move the snake one direction e.g right and then instantly move it to the opposite direction e.g left essentially making the snake move on top of itself.

In the picture below I have represented the problem visually. Here we can see a snake moving on top of its own body.



source: Snake.io

Thankfully the solution was not too hard to find as it just meant not allowing the snake to move to the opposite direction. For example if the snake was moving left it means that it could only move left as long as it was not initially right.

The coded solution 2:

```
if event.type == SNAKE_UPDATE:
    snake.movement()
if event.type == pygame.KEYDOWN:
    if (event.key == pygame.K_UP or event.key == pygame.K_w) and snake.direction != Vector2(0,1):
        snake.direction = Vector2(0,-1)
    if (event.key == pygame.K_DOWN or event.key == pygame.K_s) and snake.direction != Vector2(0,-1):
        snake.direction = Vector2(0,1)
    if (event.key == pygame.K_LEFT or event.key == pygame.K_a) and snake.direction != Vector2(1,0):
       snake.direction = Vector2(-1, 0)
    if (event.key == pygame.K_RIGHT or event.key == pygame.K_d) and snake.direction != Vector2(-1,0):
       snake.direction = Vector2(1,0)
```

Now I could move onto creating a GAME class that would contain all the functionality of the code and would not only make it much easier to debug as we would have less redundant code

but it would also make it easier for a new programmer to understand. Here I have pasted the new code with the changes implemented.

```
class GAME:
    def __init__(self):
        self.fruit = FRUIT()
        self.snake = SNAKE()

    def draw(self):
        self.snake.draw()
        self.fruit.draw()

    def update(self):
        self.snake.movement()
```

```
#Calling classes
game = GAME()
```

```
if event.type == SNAKE_UPDATE:
    game.update()
if event.type == pygame.KEYDOWN:
    if (event.key == pygame.K_UP or event.key == pygame.K_w) and game.snake.direction != Vector2(0,1):
        game.snake.direction = Vector2(0,-1)
    if (event.key == pygame.K_DOWN or event.key == pygame.K_s) and game.snake.direction != Vector2(0,-1):
        game.snake.direction = Vector2(0,1)
    if (event.key == pygame.K_LEFT or event.key == pygame.K_a) and game.snake.direction != Vector2(1,0):
        game.snake.direction = Vector2(-1, 0)
    if (event.key == pygame.K_RIGHT or event.key == pygame.K_d) and game.snake.direction != Vector2(-1,0):
        game.snake.direction = Vector2(1,0)
```

Problem 3

I have added this new code to detect collision between the snake and the fruit. If a collision occurs the fruit is generated to a new random position. However, when the snake touches the fruit it just goes under it.

```
def collision_with_fruit(self):

        if self.snake.body[0] == self.fruit.position:

            self.fruit.random()
```

```
class FRUIT:

    def __init__(self):

        self.position = self.random()



    def random(self):

        self.x = random.randint(0, Cell_number - 1)

        self.y = random.randint(0, Cell_number - 1)

        return Vector2(self.x,self.y)
```

Coded Solution 3 :

-The previous code for the `FRUIT` class didn't work because it didn't properly set the `self.position` attribute with the `Vector2` instance that holds the fruit's position.

-In the __init__ function I called the self.random() to generate new random x and y coordinates. In the random method I generated random x and y coordinates, but did not assign them to 'self.position' within the __init__ method .

-The critical issue was that the 'self.position' attribute was not being set in the '__init__' method , so the fruits position was not initialised properly. However, the corrected code ensures that 'self.position' is set with the random coordinates when a 'FRUIT' object is created. This in turn allows the fruit to have a valid initial position, which is essential for collision detection.

Making the snake grow:

To make the snake grow I created a function called grow. This function had only a single line of code inside it being

```
def grow(self):
```

```
        self.add_part = True
```

self.add part was set to False stated below the __init__ snake function:

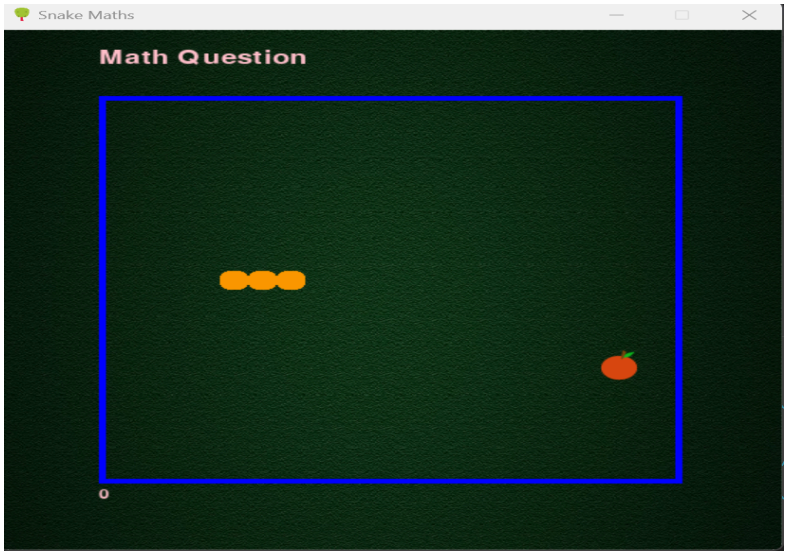
```
class SNAKE:
```

```
    def __init__(self):
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
        self.direction = Vector2(1,0)
        self.add_part = False
```

I then changed the movement function which allowed the snake to grow:

```
def movement(self):
        if self.add_part == True:
            body_copy = self.body[:]
            body_copy.insert(0, body_copy[0] + self.direction)
            self.body = body_copy[:]
            self.add_part = False
        else:
           body_copy = self.body[:-1]
           body_copy.insert(0, body_copy[0] + self.direction)
           self.body = body_copy[:]
```

I have then decided to add a border functionality to act as a river circling the snake to make it more aesthetically pleasing to the eye and improve the user experience.



After this step I added sounds for when the player hits the apple and when it causes a mistake which causes the game to end. I also added background music. With all this my Milestone 1 has been completed and the code is shown below:

```
#Initialization
import pygame
import random
import sys
from pygame.math import Vector2
from pygame import mixer
pygame.init()
pygame.font.init()


#Background, Screen, Icons  and images
Cell_size = 20
Cell_number = 20
OFFSET = 75
SCREEN = pygame.display.set_mode(((OFFSET + OFFSET) + Cell_number *
Cell_size, (OFFSET + OFFSET) + Cell_number * Cell_size))
background_image = pygame.image.load("background.jpg")
background_image = pygame.transform.scale(background_image,
(SCREEN.get_width(), SCREEN.get_height()))
ICON = pygame.image.load("snake_icon.png")
pygame.display.set_icon(ICON)
pygame.display.set_caption("Snake Maths")



#RGB values and other constants
ORANGE = (251,153,2)
BLUE = (0,0,255)
```

```
SNAKE_UPDATE = pygame.USEREVENT

pygame.time.set_timer(SNAKE_UPDATE, 150)


QUESTION_FONT = pygame.font.Font(None, 30)

SCORE_FONT = pygame.font.Font(None, 20)




#Classes

fruit_image = pygame.image.load("apple.png")

class FRUIT:

    def __init__(self):

        self.random()







    def draw(self):

        fruit_rect = pygame.Rect(OFFSET + self.position.x * Cell_size,
OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)

        SCREEN.blit(fruit_image, fruit_rect)


    def random(self):

        self.x = random.randint(0, Cell_number - 1)

        self.y = random.randint(0, Cell_number - 1)

        self.position = Vector2(self.x,self.y)


class SNAKE:

    def __init__(self):
```

```
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
        self.direction = Vector2(1,0)
        self.add_part = False
        self.chewing_sound = pygame.mixer.Sound("Sound/eat.mp3")
        self.mistake_sound = pygame.mixer.Sound("Sound/wall.mp3")

    def draw(self):
        for part in self.body:
            parts_rect = (OFFSET + part.x * Cell_size, OFFSET + part.y *
Cell_size, Cell_size, Cell_size)
            pygame.draw.rect(SCREEN, ORANGE, parts_rect, 0, 7)

    def movement(self):
        if self.add_part == True:
            body_copy = self.body[:]
            body_copy.insert(0, body_copy[0] + self.direction)
            self.body = body_copy[:]
            self.add_part = False
        else:
           body_copy = self.body[:-1]
           body_copy.insert(0, body_copy[0] + self.direction)
           self.body = body_copy[:]

    def grow(self):
        self.add_part = True

    def reset_position(self):
```

```
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
        self.direction = Vector2(1,0)



class GAME:
    def __init__(self):
        self.fruit = FRUIT()
        self.snake = SNAKE()
        self.state = "ready"
        self.score = 0
        self.playing_music = False
        pygame.mixer.music.load("Sound/bg_music_1.mp3")



    def draw(self):
        self.snake.draw()
        self.fruit.draw()


    def update(self):
        if self.state == "ready":
            self.snake.movement()
            self.collision_with_fruit()
            self.boundary_check()
            self.collision_with_body()
            self.play_background_music()
```

```
    def collision_with_fruit(self):
        if self.snake.body[0] == self.fruit.position:
            self.fruit.random()
            self.snake.grow()
            self.score = self.score + 1
            self.snake.chewing_sound.play()

    def boundary_check(self):
        if self.snake.body[0].x == Cell_number or self.snake.body[0].x ==
-1:
            self.game_over()

    def collision_with_body(self):
        for part in self.snake.body[1:]:
            if self.snake.body[0] == part:
                self.game_over()

    def play_background_music(self):
        if self.score > 0 and not self.playing_music:
            pygame.mixer.music.play(-1)  # Start playing if score is
greater than zero
            self.playing_music = True
        elif self.score == 0 and self.playing_music:
            pygame.mixer.music.stop()  # Stop playing if score goes back
to zero
            self.playing_music = False
```

```
    def game_over(self):
        self.snake.reset_position()
        self.fruit.random()
        self.state = "not ready"
        self.score = 0
        self.snake.mistake_sound.play()




#Calling classes
game = GAME()


#Game Loop
FPS = 60
RUN = True
if __name__ == '__main__':
    while RUN:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                sys.exit()
            if event.type == SNAKE_UPDATE:
                game.update()
```

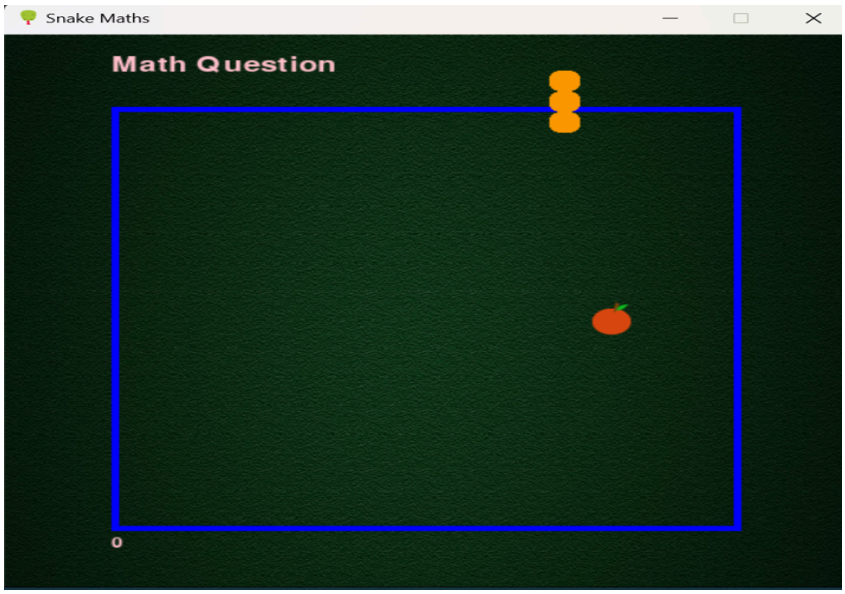
```
            if event.type == pygame.KEYDOWN:
                if game.state == "not ready":
                    game.state = "ready"
                if (event.key == pygame.K_UP or event.key == pygame.K_w)
and game.snake.direction != Vector2(0,1):
                    game.snake.direction = Vector2(0,-1)
                if (event.key == pygame.K_DOWN or event.key == pygame.K_s)
and game.snake.direction != Vector2(0,-1):
                    game.snake.direction = Vector2(0,1)
                if (event.key == pygame.K_LEFT or event.key == pygame.K_a)
and game.snake.direction != Vector2(1,0):
                   game.snake.direction = Vector2(-1, 0)
                if (event.key == pygame.K_RIGHT or event.key ==
pygame.K_d) and game.snake.direction != Vector2(-1,0):
                   game.snake.direction = Vector2(1,0)


        #Drawing
        SCREEN.blit(background_image, (0, 0))
        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5,
Cell_size*Cell_number+10, Cell_size*Cell_number + 10), 5)
        game.draw()
        Question_surface = QUESTION_FONT.render("Math Question", True,
(255,192,203))
        Score_surface = SCORE_FONT.render(str(game.score), True,
(255,192,203))
        SCREEN.blit(Score_surface, (OFFSET-5 , OFFSET +
Cell_size*Cell_number + 10))
        SCREEN.blit(Question_surface, (OFFSET - 5, 20))
        pygame.display.update()
```

```
pygame.quit()
```

3.2 Milestone 1 further review:

Problem 4:

Although my final prototype was working it still had some issues that needed to be fixed.The first minor issue was that there was small possibility chance of the fruit spawning on top of the snake which would corrupt the whole game logic .The second minor issue was that the sound for some reason started playing 3 seconds after hitting the apple.The third issue was that for some reason the boundary check for the Y axis was not working and the issue is shown visually below:



Solution 4:

First problem solution:

```
for part in self.snake.body[0]:
    if part == self.fruit.position:
        self.fruit.random()
```

This code ensured that the fruit would never spawn on top of the body by making the program generate another random fruit position that was not on top of the body.

Second issue could not be solved: The background music only played for the first time till the snake died so I decided to leave it for now and come back to it later

The third solution:

```
def boundary_check(self):

        if self.snake.body[0].x == Cell_number or self.snake.body[0].x ==
-1:

            self.game_over()

        if self.snake.body[0].y == Cell_number or self.snake.body[0].y ==
-1:

            self.game_over()
```

The issue for this one is that I forgot to check if the body hit any of the y axis so I simply had to copy the code and change x to y.

Problem 5 :

While playing the game and about to move to milestone 2 I realised that the apple was sometimes being hit by the head but did not register. I spent countless hours trying to figure out the issue. I first tried calculating the distance between them and if they were very close just count it as if the head hit the apple. However, this created more issues and I decided to scrap the idea and made a new solution.

Coded solution 5:

I made a grid that circled the blue perimeter which is meant to represent a river. I then scaled my apple to fit the square inside the grid perfectly. Below I have pasted the code I used.

```
def draw_grid(screen, cell_size, cell_number, offset):
    # Define the colors
    WHITE = (255, 255, 255)

    # Draw the horizontal lines
    for y in range(offset, offset + cell_size * cell_number + 1, cell_size):
        pygame.draw.line(screen, WHITE, (offset, y), (offset + cell_size * cell_number, y), 1)

    # Draw the vertical lines
    for x in range(offset, offset + cell_size * cell_number + 1, cell_size):
        pygame.draw.line(screen, WHITE, (x, offset), (x, offset + cell_size * cell_number), 1)

    # Draw the blue perimeter on top of the grid
    pygame.draw.rect(screen, BLUE, (offset - 1, offset - 1, cell_size * cell_number + 2, cell_size * cell_number
```

```
#Drawing

        SCREEN.blit(background_image, (0, 0))

        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5,
Cell_size*Cell_number+10, Cell_size*Cell_number + 10), 5)

         # Clear the screen

        SCREEN.blit(background_image, (0, 0))


        # Draw the blue perimeter

        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5, Cell_size
* Cell_number + 10, Cell_size * Cell_number + 10), 5)


       # Draw the grid

        draw_grid(SCREEN, Cell_size, Cell_number, OFFSET)

  # Draw the blue perimeter on top of the white grid

        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5, Cell_size
* Cell_number + 10, Cell_size * Cell_number + 10), 5)
```

3.3 Milestone 1 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Test Met?
1	Load Background image	Typical	Background image currently being loaded and updated while the game is being run	Yes Reference: Pages 82,84
2	Load snake icon	Typical	Snake icon being correctly loaded	Yes Reference:

				Page 82,84
3	Load Game title	Typical	Title of the game being displayed next to the icon	Yes Reference: Page 82,84
4	Load snake	Typical	Snake correctly being loaded	Yes Reference: Page 88
5	Load apple	Typical	Apple currently being loaded	Yes Reference: Page 82,88
6	Make boundary	Typical Boundary	Ensuring the snake won't cross the boundary	Yes Reference: Page 88,96
7	Check movement	Typical Boundary	Ensuring the snake can move horizontally and vertically	Yes Reference: Pages 84,86,96 In these pages I have explained the code. As you can see in the images the snake's position is changing due to its movement functionality working . I have also inserted a

				video link showing the movement working in test number 9
8	Have a score function	Typical	Ensuring the score is correctly incremented as you eat the apple	Yes Reference: Page 88 In the images bottom left corner where score at that current moment was 0
9	Adding sound effects when eating apple	Typical	Sounds should be heard when eating the apple	Yes Reference: Page 88  Video reference: Music/so... Youtube Link:https://youtu.be/IWwVlfyHqgY
10	Adding music	Typical	Music should be playing while the game	Yes Reference: Page 93

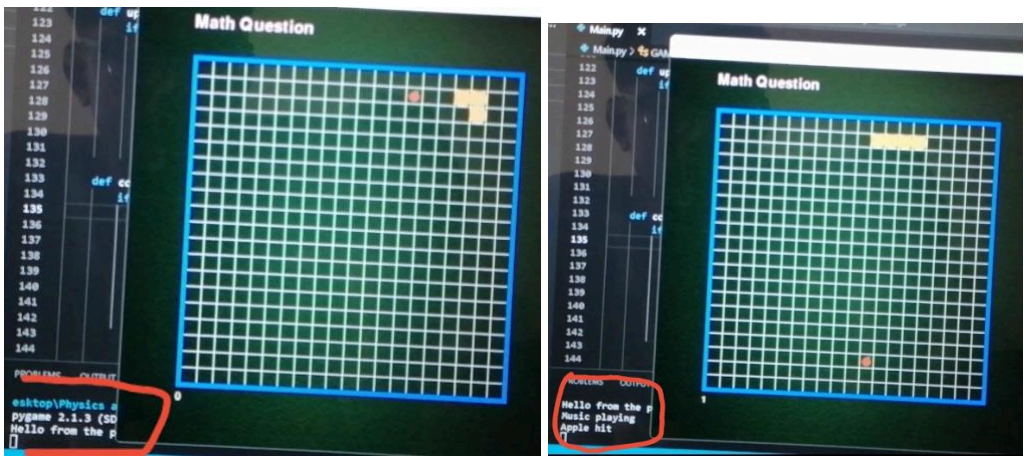
			is running	👇 Youtube Link:https://youtu.be/IWwVlfyHqgY Video Reference: Music/so…

For Test number 9 and 10 I have decided to add print statements that the terminator will print if the music or the sound is playing.

The code👇:

```
def collision_with_fruit(self):
    if self.snake.body[0] == self.fruit.position:
        self.fruit.random()
        self.snake.grow()
        self.score = self.score + 1
        self.snake.chewing_sound.play()
        self.increase_speed()
        if pygame.mixer.music.get_busy():
            print("Music playing")
        print("Apple hit")
```

Affect:



:

Initially when the score was zero the terminator didn't display anything but after hitting the apple, the score incremented by one and the terminator displayed the apple being eaten and the music's playing

All of milestone 1 was shown in a video format. Just follow this link to see it:
https://youtu.be/IWwVlfyHqgY

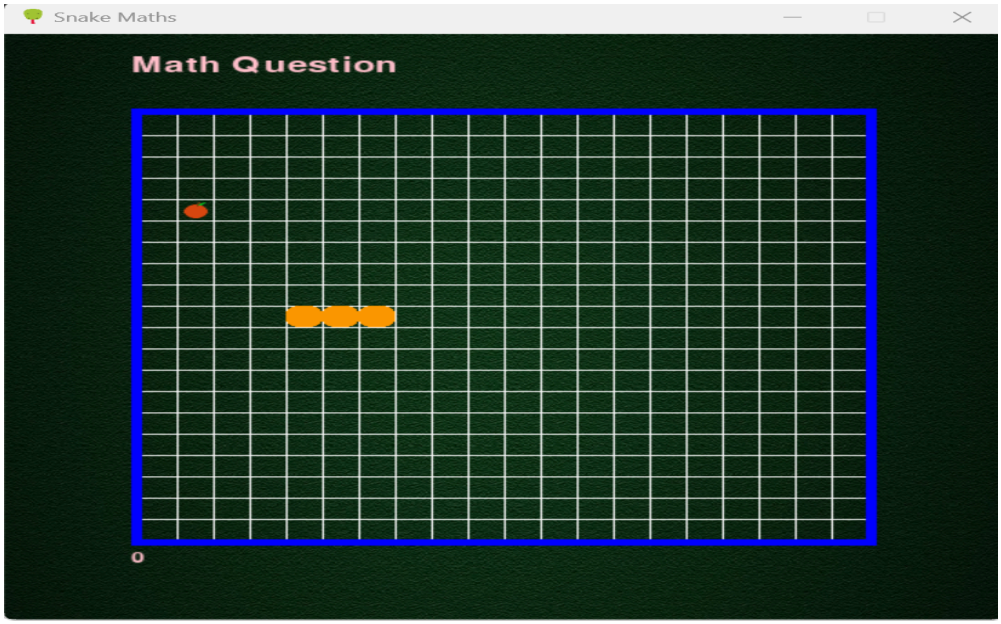
User feedback 1:

My user thought the first prototype was perfect with the only objection being about the score. They suggested that the score was not intuitive meaning that it was hard to understand what the number was representing so they suggested adding a variable such as “score:” to make it clear what the number was representing

Reflection 1:

I completely agree with the user and will implement the change in my next milestone as soon as possible

3.4 Milestone 2 start:



After finishing milestone 1 I thought about increasing the functionality of the game. I noticed that the game did not return the highest score the user received. For this I decided to create a green filled screen that displays the final score of the user and starts a new game only if the user presses the spacebar

Steps involved:

1)created a new function to reset the game .

```
def reset_game(self):
    # Reset game attributes to their initial state
    self.fruit = FRUIT()
    self.snake = Snake()
    self.get_name = False
    self.state = "ready"
    self.play_background_music()
```

2)Modified the game over function:

```
def game_over(self):

        pygame.mixer.music.stop()  # Stop background music

        self.snake.reset_position()

        self.fruit.random()

        self.state = "not ready"

        final_score = self.score

        self.play_background_music()  # Start playing background music
again

        self.snake.mistake_sound.play()

        print("Final Score:", self.score)


        font = pygame.font.Font(None, 48)

        text = font.render(f"Final Score: {self.score}", True, WHITE)

        text_rect = text.get_rect(centre=(self.SCREEN.get_width() // 2,
self.SCREEN.get_height() // 2))

        self.SCREEN.fill(GREEN)

        self.SCREEN.blit(text, text_rect)

        pygame.display.update()


        waiting_for_input = True

        while waiting_for_input:
```

```
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_SPACE:  # Restart the game on
spacebar press
                        self.reset_game()  # Reset the game state
                        waiting_for_input = False
```

This modified game over function allows for the program to display the final score on a green screen and wait for the user to press space bar before resetting the game.

3) We also changed the game loop

```
 if event.type == pygame.KEYDOWN:
                if game.state == "not ready" and event.key ==
pygame.K_SPACE:  # Restart the game only on spacebar press
                    game.reset_game()
                    waiting_for_input = False
```

Here we added a check to restart the game when the user presses the space bar only if the game state is “not ready” . These changes allows for the program to display the final score on a green screen and wait for the user to press space bar before resetting the game.

Problem 6:

You know how when the snake moves right before we would not let it move left if the snake was already moving right or we would not let the snake move up if it was already moving down. Well this functionality doesn't seem to work anymore and now if the snake was going left the user can now press right and the snake would start going right which would end the game due to collision with the snake's body.

Solution 6:

```
if event.key == pygame.K_UP or event.key == pygame.K_w:
                        if game.snake.direction != Vector2(0, 1):
                            game.snake.direction = Vector2(0, -1)
                    elif event.key == pygame.K_DOWN or event.key ==
pygame.K_s:
                        if game.snake.direction != Vector2(0, -1):
                            game.snake.direction = Vector2(0, 1)
                    elif event.key == pygame.K_LEFT or event.key ==
pygame.K_a:
                        if game.snake.direction != Vector2(1, 0):
                            game.snake.direction = Vector2(-1, 0)
                    elif event.key == pygame.K_RIGHT or event.key ==
pygame.K_d:
                        if game.snake.direction != Vector2(-1, 0):
                            game.snake.direction = Vector2(1, 0)
```

Problem 7:

While documenting this whole process I realised I declared a variable called FPS but have not used it. My initial reasoning for having an FPS set up with a value of 60 as this is a solid frame rate that should run on most computers without any issues.

Coded Solution 7

For this all I had to do was import the time module from pygame and set the amount of times it was going to refresh to a variable named Clock. For these changes the code has been provided below:

```
import pygame.time
```

```
clock = pygame.time.Clock()
FPS = 60
```

```
# Control frame rate
clock.tick(FPS)
```

Timer function:

I have decided to now move to the timer function to keep count in seconds for how long the game has been running.

```
self.play_background_music()
self.start_time = pygame.time.get_ticks()
```

```
def draw_timer(self):
    current_time = pygame.time.get_ticks()
    elapsed_time_seconds = (current_time - self.start_time) // 1000  # Convert to seconds
    timer_text = f"Time: {elapsed_time_seconds} s"
    timer_surface = SCORE_FONT.render(timer_text, True, (255, 192, 203))
    timer_rect = timer_surface.get_rect(topright=(self.SCREEN.get_width() - 10, 20))
    self.SCREEN.blit(timer_surface, timer_rect)
```

-timer_text = f"Time left: {time_remaining // 1000} s": Constructs a text string indicating the time left by converting time_remaining (presumably in milliseconds) to seconds.

-SCORE_FONT.render(timer_text, True, (255, 0, 0)): Renders the timer text onto a surface using a predefined font (SCORE_FONT). The (255, 0, 0) represents the text colour (red).

-timer_rect = timer_surface.get_rect(topright=(self.SCREEN.get_width() - 10, 20)): Creates a rectangle for the timer surface, positioning it at the top-right corner of the screen with a 10-pixel margin from the right and a 20-pixel margin from the top.

-self.SCREEN.blit(timer_surface, timer_rect): Draws (blitz) the timer surface onto the game screen at the specified rectangle position.

```
game.draw_timer()
pygame.display.update()
```

Maths functionality:

From here on I focused on the maths functionality of the game. My objectives were to create a function that would generate questions on the screen e.g “8+9”. I then had to

create a function for the common operations. These would check if the answer is correct or not. Lastly I created another fruit object and edited the fruit class.

```
self.correct_answer = 0  # Initialize correct_answer attribute
self.correct_answer = 0  # Initialize correct_answer attribute
self.incorrect_answer = 0
self.current_question = ""
self.selected_operation = "addition"
#self.selected_operation = "subtraction"
#self.selected_operation = "multiplication"
#self.selected_operation = "division"
self.generate_question()
```

For now the default operation was set as addition as we could just implement the same logic and functionalities of the addition operation to other operations but just change the signs.

```
def generate_question(self):
    if self.selected_operation == "addition":
        num1 = random.randint(1, 10)
        num2 = random.randint(1, 10)
        self.correct_answer = num1 + num2
        self.incorrect_answer = num1 + num2 + 1
        self.current_question = f"{num1} + {num2}"
```

The idea was to create a simple function that would produce a correct answer and one incorrect answer. In this function we generate two random numbers between one to ten and then add them together before saving it inside the variable called “self.correct_answer”. We then create another variable that uses the same two numbers but just adds for the time being to make an incorrect answer and stores it inside a variable called “self.incorrect answer”. This adding 1 functionality is temporary as we are just making a prototype for an incorrect answer and later on can change the difficulty.

```
# Generate unique correct and incorrect answers for self.fruit
correct_answer_fruit = self.correct_answer
incorrect_answer_fruit = self.incorrect_answer
self.fruit = FRUIT(correct_answer_fruit, incorrect_answer_fruit)

# Generate unique correct and incorrect answers for self.fruit2
correct_answer_fruit2 = self.correct_answer
incorrect_answer_fruit2 = self.incorrect_answer
self.fruit2 = FRUIT(correct_answer_fruit2, incorrect_answer_fruit2)
```

```
# Display the current question using an f-string
question_text = f"Question: {self.current_question}"
question_surface = QUESTION_FONT.render(question_text, True, (255, 192, 203))
question_rect = question_surface.get_rect(topleft=(OFFSET + 10, 20))
self.SCREEN.blit(question_surface, question_rect)
```

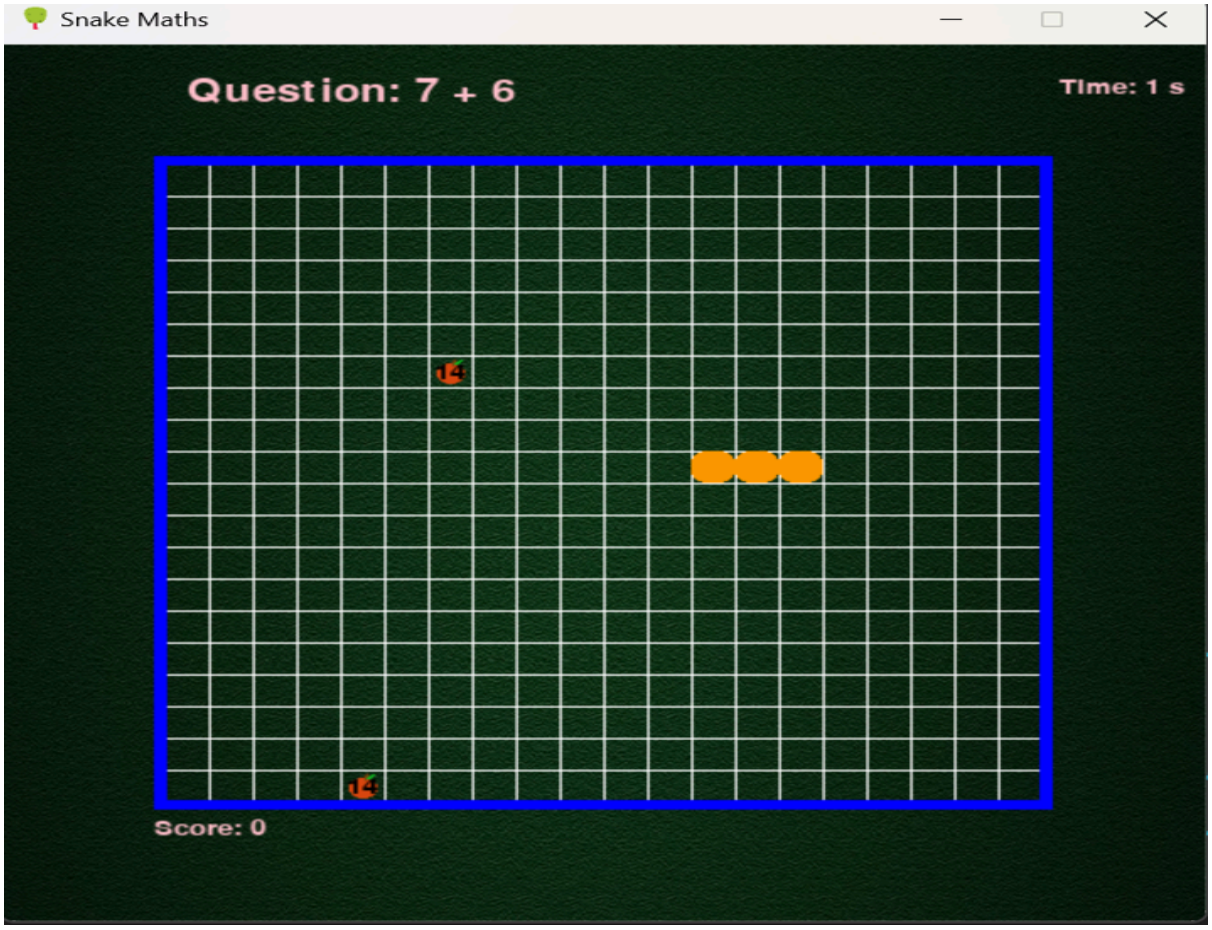
```
class GAME:
    def __init__(self):
        self.fruit = FRUIT()
        self.fruit2 = FRUIT()
        self.snake = SNAKE()
        self.SCREEN = SCREEN
        self.state = "ready"
        self.score = 0
        self.playing_music = False
```

In this block of code I initialised the other apple “self.fruit2 = FRUIT()” This is so that it has all the same characteristics of the original fruit class with the only difference being that sometimes it may hold the incorrect answer

Problem 8:

However this had some issues because sometimes questions pop up for example "7+6" and the 2 fruits hold different values with one holding the right one and one holding the wrong one. But other times the answers are not just wrong but they are both holding the wrong value. Other times one of the fruits says none

Issue visually defined:



Solution 8 :

This involved getting rid of any other functionality involving any operators such as addition multiplication and division.I then edited the generate random question and created another function called generate_incorrect_answer to fix the error and the result is shown below:

```
# Maths Logic and functionality

    def generate_question(self):

        if self.selected_operation == "addition":

            num1 = random.randint(1, 10)

            num2 = random.randint(1, 10)

            self.correct_answer = num1 + num2
```

```
            self.incorrect_answer =
self.generate_incorrect_answer(self.correct_answer)


            # Generate the first incorrect answer
            while True:
                self.incorrect_answer =
self.generate_incorrect_answer(self.correct_answer)
                if self.incorrect_answer is not None and
self.incorrect_answer != self.correct_answer:
                    break


            # Generate the second correct and incorrect answers
            while True:
                num1_2 = random.randint(1, 10)
                num2_2 = random.randint(1, 10)
                self.correct_answer2 = num1_2 + num2_2
                self.incorrect_answer2 =
self.generate_incorrect_answer(self.correct_answer2)


                if self.correct_answer2 != self.correct_answer and
self.incorrect_answer2 != self.correct_answer:
                    break


            # Choose randomly which fruit will have the correct answer
            correct_fruit = random.choice([self.fruit, self.fruit2])
            correct_fruit.correct_answer = self.correct_answer
            correct_fruit.incorrect_answer = self.incorrect_answer
            correct_fruit.is_correct = True
```

```
            incorrect_fruit = self.fruit2 if correct_fruit is self.fruit
else self.fruit
            incorrect_fruit.correct_answer = self.correct_answer2
            incorrect_fruit.incorrect_answer = self.incorrect_answer2
            incorrect_fruit.is_correct = False


            # Generate the current question string
            self.current_question = f"{num1} + {num2}"



    def generate_incorrect_answer(self, correct_answer,
other_incorrect_answer=None):
    # Generate an incorrect answer that is not equal to the correct answer
        while True:
            incorrect_answer = random.randint(1, 20)  # Adjust the range
as needed
            if incorrect_answer != correct_answer and incorrect_answer !=
other_incorrect_answer:
                return incorrect_answer
```

The result:

Adding a time left function to increase difficulty:

This involved giving the user specific time frame to select the right answer. This would enable the user to develop critical thinking skills in time conditions as there was a time restriction to selecting the correct answer.

We had to modify the game class and add new attributes:

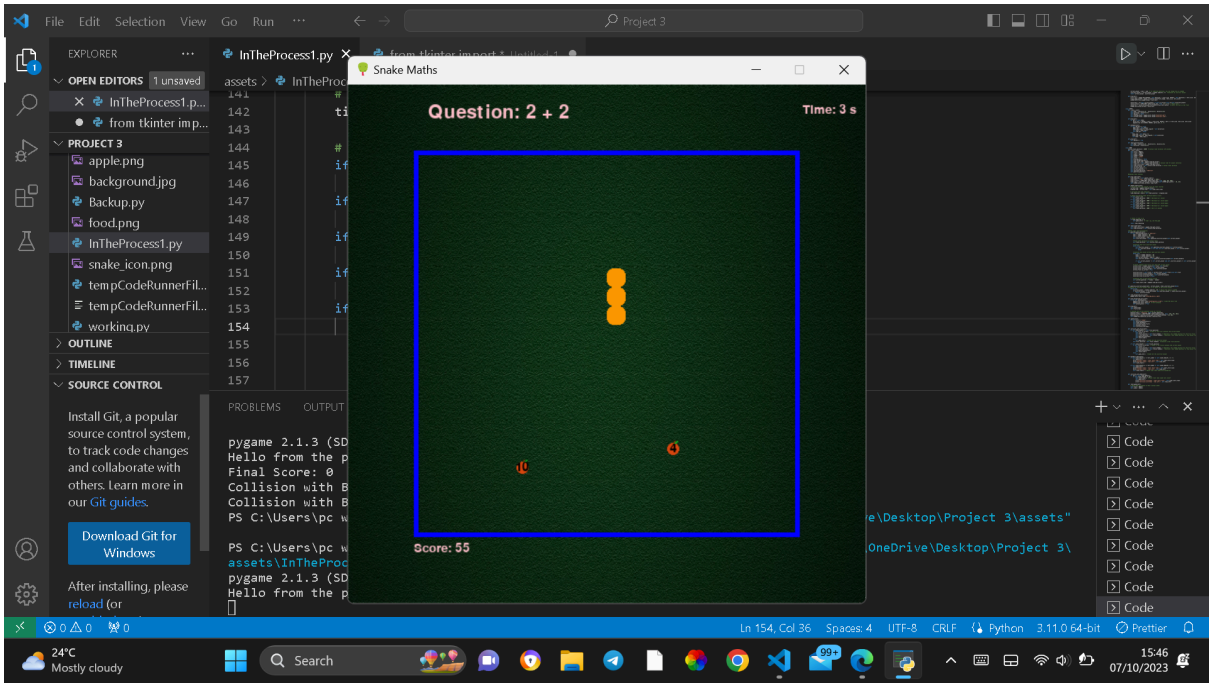
```
# Timer attributes
self.initial_timer_duration = 11000  # Initial timer duration (10 seconds)
self.time_left = self.initial_timer_duration  # Initial time for answer selection
self.timer_start_time = pygame.time.get_ticks()
self.timer_duration = self.initial_timer_duration  # Actual timer duration
self.start_time = pygame.time.get_ticks()  # Record the start time when the game starts
```

```
def update_timers(self):
    # Calculate the time elapsed since the timer started
    current_time = pygame.time.get_ticks()
    elapsed_time = current_time - self.timer_start_time

    # Calculate the time remaining
    time_remaining = max(0, self.timer_duration - elapsed_time)
```

```
# Handle decreasing timer duration based on score
if self.score >= 10:
    self.timer_duration = 9000  # Decrease by 1 second
elif self.score >= 20:
    self.timer_duration = 8000
elif self.score >= 230:
    self.timer_duration = 7000
elif self.score >= 40:
    self.timer_duration = 6000
elif self.score >= 50:
    self.timer_duration = 5000  # Set to 5 seconds

# Check if time is up
if time_remaining == 0:
    self.game_over()  # Time's up, end the game
```

```
return time_remaining
```

Result:



Displaying the running time:

I decided to display the running time on the screen as we can use this information later on for analytics showing how long each player practised for.

What I did was:

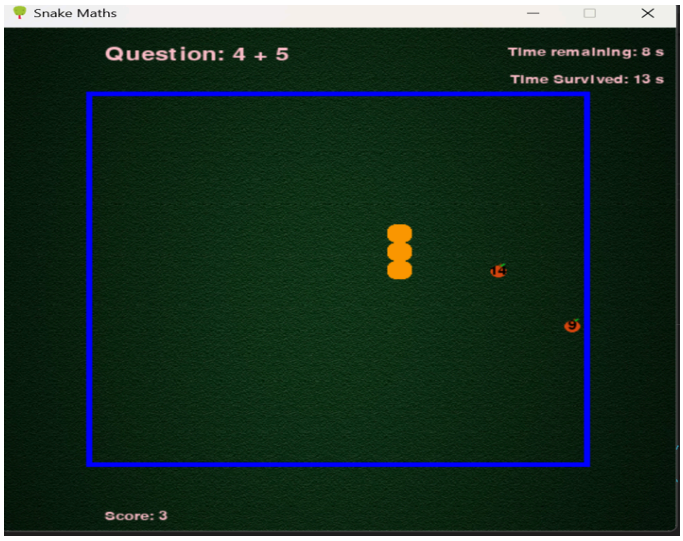
```
# Timer attributes
self.initial_timer_duration = 10000  # Initial timer duration (10 seconds)
self.time_left = self.initial_timer_duration  # Initial time for answer selection
self.timer_start_time = pygame.time.get_ticks()
self.timer_duration = self.initial_timer_duration  # Actual timer duration
self.start_time = pygame.time.get_ticks()  # Record the start time when the game starts
```

```
def calculate_time_survived(self):
    current_time = pygame.time.get_ticks()  # Get the current time
    time_survived = current_time - self.start_time  # Calculate time survived
    return time_survived

def draw_time_survived(self):
    time_survived = self.calculate_time_survived()
    time_text = f"Time Survived: {time_survived // 1000} s"
    time_surface = SCORE_FONT.render(time_text, True, (255, 192, 203))
    time_rect = time_surface.get_rect(topright=(self.SCREEN.get_width() - 10, 50))
    self.SCREEN.blit(time_surface, time_rect)
```

```
def update(self):
    if self.state == "ready":
        self.play_background_music()
        self.snake.movement()
        self.collision_with_fruit()
        self.boundary_check()
        self.collision_with_body()
        self.draw_time_survived()  # Draw the "Time Survived" on the screen
```

```
# Draw game-related information
game.draw()  # Draw the current question
game.draw_score()  # Draw the player's score
game.draw_timer()  # Draw the timer
game.draw_time_survived()  # Draw the time survived
```

The result:



I have removed the grid from now on to make it easier to see the changes

3.5 Milestone 2 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Test Met? Yes/No
1	Maths question correctly displayed	Typical	The maths question is visible and correctly displayed to the player.	Yes Reference: Page 109
2	Two apple fruit objects created	Typical	Two apple objects are visible and correctly displayed on the game grid.	Yes Reference: Page 109
3	Time running function displayed	Typical	The timer is visible and continuously updates to	Yes Reference: Page 109

			reflect the elapsed time	
4	Time left function displayed	Typical	The countdown timer is visible and accurately reflects the time remaining	Yes Reference: Page 115
5	Score incremented if correct answer chosen	Typical	The player's score is incremented by the specified amount.	Yes Reference: Page 115
6	Two apple objects containing different answers	Typical	Each apple object displays a different answer, allowing the player to select the correct one	Yes Reference: Page 115

User feedback 2:

-The user found the second prototype to be impressive, particularly praising the addition of maths questions to the gameplay.
-They appreciated the challenge it added and mentioned enjoying the mental stimulation it provided during gameplay.

Reflection 2:

I'm glad to hear the positive feedback on the maths questions. It validates the decision to integrate educational components into the game. Moving forward, I'll continue to refine this aspect to ensure it remains engaging and beneficial for users.

3.6 Milestone 3 start:

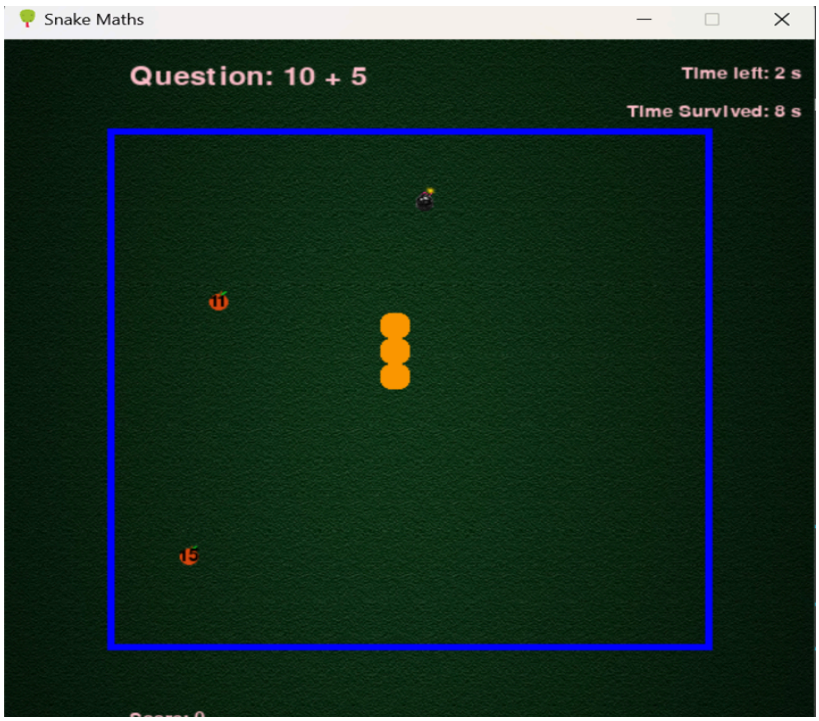
Adding drops:

To make the game more interesting I decided to add power ups. The first powerup being a bomb that would explode and end the game once the snake's head touches it.

Here is the relevant code for it.

```
class Bomb:
    def __init__(self):
        self.position = self.random()
        self.explode = False

    def random(self):
        x = random.randint(0, Cell_number - 1)
        y = random.randint(0, Cell_number - 1)
        return Vector2(x, y)

    def draw(self):
        if self.explode:
            Bomb_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
            Bomb_image_scaled = pygame.transform.scale(explode_image, (Cell_size, Cell_size))
        else:
            Bomb_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
            Bomb_image_scaled = pygame.transform.scale(bomb_image, (Cell_size, Cell_size))
        SCREEN.blit(Bomb_image_scaled, Bomb_rect)

    def check_collision(self, snake_head):
        if snake_head == self.position:
            self.explode = True


        return snake_head == self.position
```

The code works essentially like the fruit class with the only difference being that when it has any collisions with the snake the game ends.



Increase difficulty

I have then decided to increase the difficulty of the game. The two ways I could do that was increasing the snake part to a maximum of length 10. I chose a maximum of length 10 as explained in the design section. We would have 2 apples in the game and increasing the length any further would make movement too

difficult and the game unplayable. We could then add another bomb to the game to make collisions more likely but I decided to add a stone just to spice things up

For increasing the length:

```
class SNAKE:
    def __init__(self):
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
        self.direction = Vector2(1,0)
        self.add_part = False
        self.chewing_sound = pygame.mixer.Sound("Sound/eat.mp3")
        self.mistake_sound = pygame.mixer.Sound("Sound/wall.mp3")
        self.growth_count = 0  # Keep track of how many parts the snake has grown
```

```
    def movement(self):
        self.body.insert(0,self.body[0] + self.direction)
        if self.add_part and self.growth_count < 10:
            self.growth_count += 1
        else:
            # If the snake has reached the maximum growth, remove the last part
            self.body = self.body[:-1]
        self.add_part = False


    def grow(self):
        # Check if the snake can grow (up to a maximum of 10 parts)
        if self.growth_count < 10:
            self.add_part = True
            self.growth_count += 1
```

```
def reset_position(self):
    self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
    self.direction = Vector2(1,0)
    self.growth_count = 0
```

```
def collision_with_fruit(self):
    if self.snake.body[0] == self.fruit.position:
        if self.fruit.is_correct:  # Check if the fruit contains the correct answer
            self.snake.grow()
            self.score += 1
            self.fruit.position = self.fruit.random()  # Generate a new random position for the first fruit
            self.fruit2.position = self.fruit2.random()  # Generate a new random position for the second fru
            self.generate_question()
            self.reset_timer()
```

```
elif self.snake.body[0] == self.fruit2.position:
    if self.fruit2.is_correct:  # Check if the fruit2 contains the correct answer
        self.snake.grow()
        self.score += 1
        self.fruit.position = self.fruit.random()  # Generate a new random position for the first fruit
        self.fruit2.position = self.fruit2.random()  # Generate a new random position for the second fru
        self.generate_question()
        self.reset_timer()
        # Respawn the stone randomly when the score goes up
        self.stone.position = self.stone.random()
```

The result;

Problem 9:

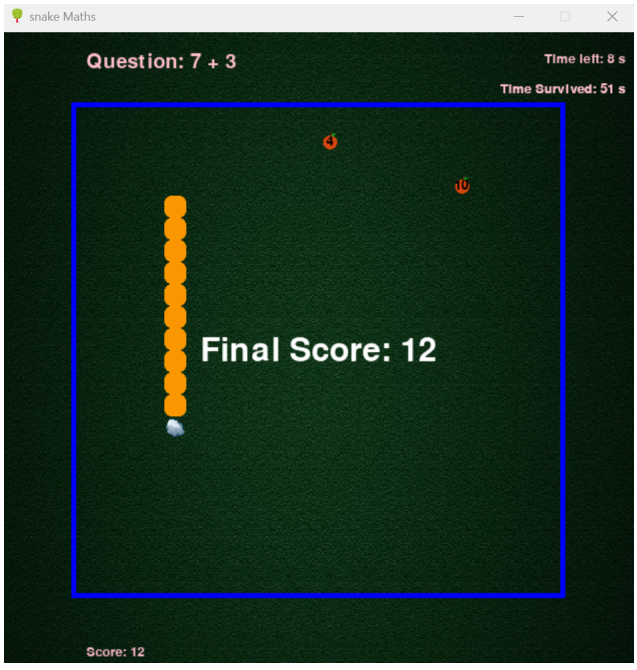
For some reason when I set the growth counter to 10, the snake grew to 8 blocks.

Coded solution 9:

So I set it to 12 and the snake grew only up to 9 blocks. That's why I decided to set the growth count to 14 to ensure a max length of 10 is achieved.

```
    def grow(self):
        # Check if the snake can grow (up to a maximum of 10 parts)
        if self.growth_count < 14: #(For some reason when you set the growth count to 10 it only grows 8 parts)
            self.add_part = True
            self.growth_count += 1
```

The result:

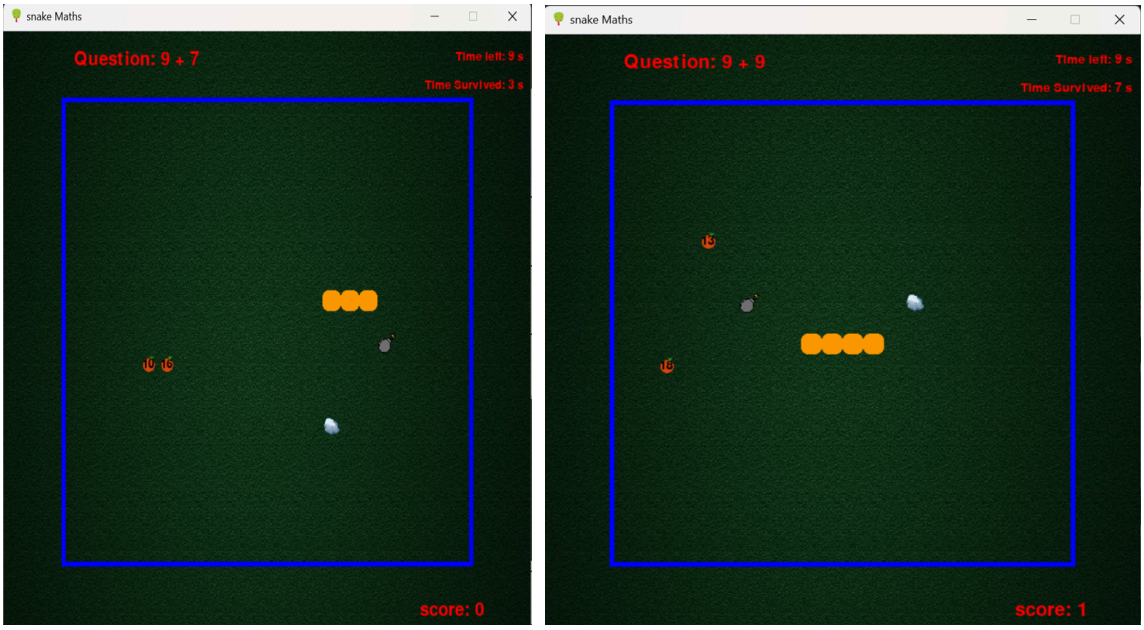


3.7 Milestone 3 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Test Met? Yes/No
1	Bomb loaded onto the screen	Typical	Bomb correctly loaded	Yes Reference:117 Extra reference: 
2	Bomb spawns random locations	Typical	Bomb correctly spawning different places	Yes Reference: The two screenshots below the table 
3	Bomb explodes when touched	Typical	Bomb correctly exploding once touched	Yes Reference:

				
4	Rock correctly loaded	Typical	Rock correctly being loaded onto the screen	Yes Reference:120 Extra reference: 
5	Rock spawns different locations	Typical	Rock correctly spawning different locations	Yes Reference: Pages 119,120 In these 2 images it is seen the rock spawns in different locations Extra reference:

				
6	Rock collision with snake ends the game	Typical	Rock correctly ending the game if snake collides with it	Yes Reference: Page 120
7	Length increasing to a certain point	Typical	The snake's length visibly increases on the game grid until it reaches the predetermined length threshold	Yes Reference Page 120 and image below where size increased by one block



Here I am showing how the bomb changes position as the score is incremented. I have also changed the colour of text from pink to red for better visibility. Initially all the objects were in the position shown in the image on the left but as soon as the snake selected the right answer to the question. In this case the answer is 16. All the objects position changed, the time survived increased and the time left function reset

User feedback 3:

My user was happy with the functionality however had a suggestion about the aesthetic. They said it was too boring and dull and suggested I find a way to improve it.

Reflection 3:

I decided to make the background an exercise books page as it establishes a scholarly theme, enhancing the game's educational focus, while also adding visual interest and depth to the environment. This choice creates a more engaging and thematically coherent experience for players.

3.8 MileStone 4 start:

```
Home = pygame.image.load("home.png")
Home = pygame.transform.scale(Home, (50, 50))


Reset = pygame.image.load("reset.png")
Reset = pygame.transform.scale(Reset, (50, 50))
```

The code loads four images (home.png, reset.png, speaker1.png, and speaker2.png) using the Pygame library and then resizes each image to a uniform size of 50x50 pixels. The images are likely intended for use in a Pygame-based graphical user interface or game interface.

```
class GAME:
    initial_timer_duration = 10000  # Initial timer duration (10 seconds)
    def __init__(self,mode):
        self.SCREEN = SCREEN
        self.fruit = FRUIT()
        self.fruit2 = FRUIT()
        self.snake = Snake()
        self.stone = STONE()
        self.mode = mode
        self.bomb = Bomb()
        self.name = username
        self.get_name = False
        self.home_rect = pygame.Rect(180,400,50,50)
        self.play_again_rect = pygame.Rect(280,400,50,50)
        self.mute_rect = pygame.Rect(380,400,50,50)
```

Three rectangular areas are defined using points:

•Home Rectangle: Top-left corner at (180, 400), with a width and height of 50 pixels.

•Play Again Rectangle: Top-left corner at (280, 400), with a width and height of 50 pixels.

•Mute Rectangle: Top-left corner at (380, 400), with a width and height of 50 pixels.

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()

    if event.type == pygame.MOUSEBUTTONDOWN:
        if self.mute_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]):
            if self.playing_music:
                self.playing_music = False

            else:
                self.playing_music = True

        elif self.home_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]):
            menu()

        elif self.play_again_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]):
            if self.get_name:
                if self.mode == 'addition':
```

If the mouse button is pressed (pygame.MOUSEBUTTONDOWN event):

•Check if the mouse is within the mute button area:

•If the music is currently playing (self.playing_music is True), set it to False; otherwise, set it to True.

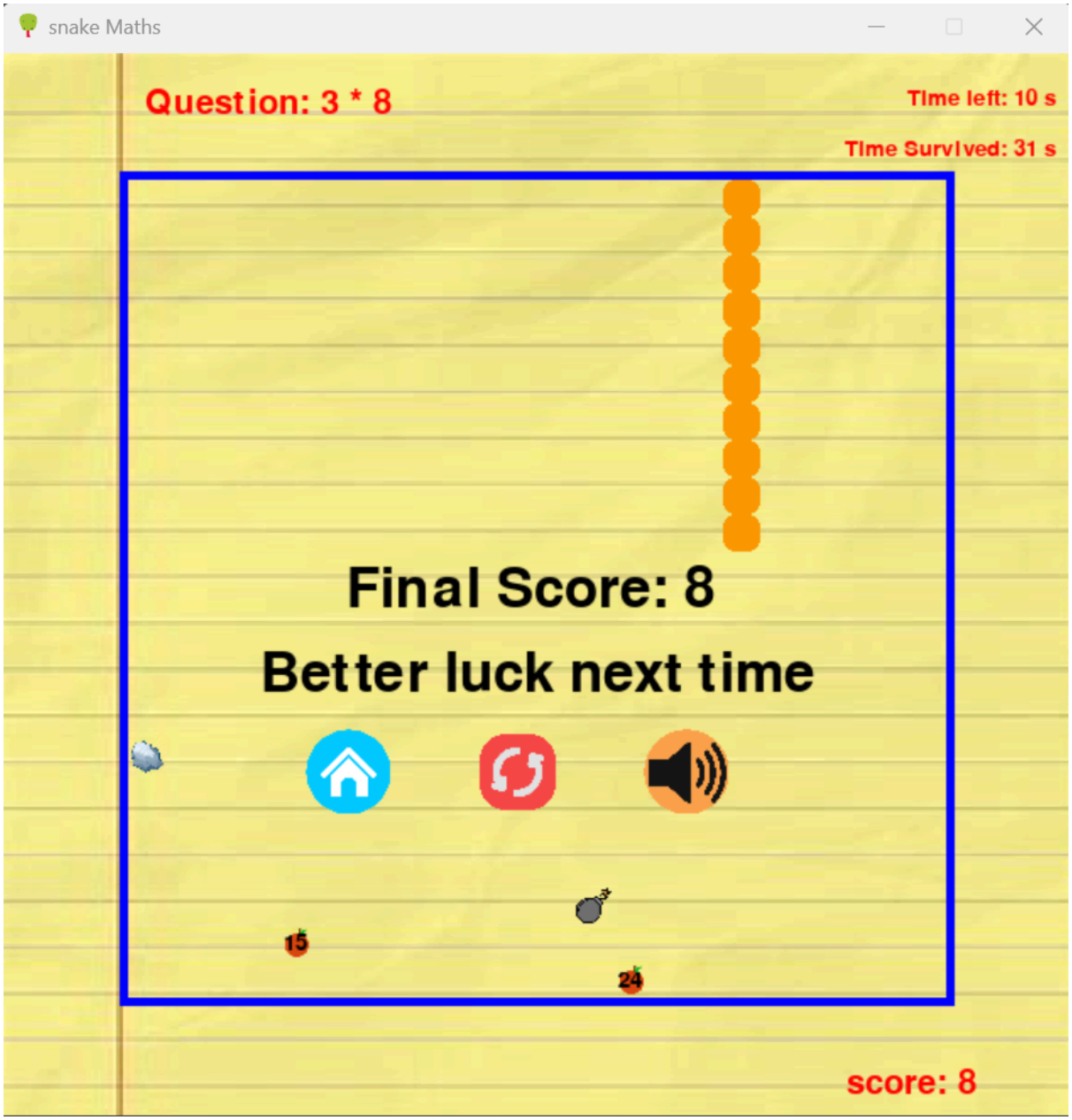
•Else, check if the mouse is within the home button area:

•Call the menu() function.

•Else, check if the mouse is within the play again button area:

•If certain conditions are met (self.get_name, self.mode, self.score), prepare SQL queries to update the "Addition_table" with the current score and name.

The result:



Making the button for the GUI:

After making the second prototype of the game with all the maths functionality it was time to make the GUI. I had the option of using Tkinter or I could just use pygame. I chose to use pygame as my GUI was not going to be anything crazy. To make the GUI

however I had to first make another script to make a class called button. This class could then be used throughout the program.

The code for it:

```
# Class for buttons
class Button():
    # Parameters are image of rectangle,position of it,text of the button,font,and the color shown when mouse is
    def __init__(self, image, pos, text_input, font, base_color, hovering_color):

        # Each parameter is put in a self variable
        self.image = image
        self.x_pos = pos[0]
        self.y_pos = pos[1]
        self.font = font
        self.base_color, self.hovering_color = base_color, hovering_color
        self.text_input = text_input
        self.text = self.font.render(self.text_input, True, self.base_color)
        self.change = base_color
```

All of its methods:

```
        if self.image is None:
            self.image = self.text
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))
        self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))

    # function to update
    def update(self, screen):
        # If there's image it is placed in such a way that text can be in the middle
        if self.image is not None:
            screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)

    # The fuction to check for input(Mouse button)
    def checkForInput(self, position):
        # Checks position of mouse and text and if the mouse has been clicked
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.re
            return True
        return False

    # function for hovering color
    def changeColor(self, position):

        # Checks mouse position and text position if they are same, the color of text changes
        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.re
            self.text = self.font.render(self.text_input, True, self.hovering_color)
            self.change = self.hovering_color
        else:
            self.text = self.font.render(self.text_input, True, self.base_color)
            self.change = self.base_color
```

This code creates a button that you can click on in a computer program. The button can have text on it and change colour when you move your mouse over it. This is the break down of what the code does:

Initialization (__init__):

- When you make a button, you give it a few things: an image (or it can just be based on text), a position on the screen, the text you want on the button, the font

style for the text, and two colours – one for when your mouse is away, and one for when your mouse is on the button.
- It sets up the button's initial appearance and position.

Update Function (update):
- This function updates how the button looks on the screen.
- If you gave it an image, it shows the image on the screen at the specified position. It also displays the button's text at the same spot so that it appears in the middle of the button.

Checking for Clicks (checkForInput):
- This function checks if you've clicked the button with your mouse.
- It looks at where your mouse is, and if your mouse is over the button, it says "yes, the button was clicked." If not, it says "no, the button was not clicked."

Changing Color (changeColor):
- This function is about the button's colour when you hover your mouse over it.
- When your mouse is over the button, the text on the button changes colour to the "hovering" colour. When you move your mouse away, it goes back to the normal colour.

In a nutshell, this code creates a clickable button with text that changes colour when you hover over it. It's like a button you'd find in an app or a game, and you can customise how it looks and behaves.

Creating the menu and using buttons:

```
def menu():
    img = pygame.transform.scale(pygame.image.load('math_symbol.png'),(120,120))
    while True:
        SCREEN.fill('white')
        SCREEN.blit(img,(50,30))
        pygame.draw.rect(SCREEN,'black',(0,600,1000,5))
        pygame.draw.rect(SCREEN,'grey',(SCREEN.get_width()/2+10,190,280,335))
        MOUSE_POS = pygame.mouse.get_pos()
        Addition = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCR
                          text_input="Addition", font=SCORE_FONT, base_color="black", hovering_color="green")

        Subtraction = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(
                          text_input="Subtraction", font=SCORE_FONT, base_color="black", hovering_color="green")

        Multiply = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCR
                          text_input="Multiplication", font=SCORE_FONT, base_color="black", hovering_color="green")

        Divide = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCREE
                          text_input="Division", font=SCORE_FONT, base_color="black", hovering_color="green")

        Leaderboard = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(
                          text_input="Leaderboard", font=SCORE_FONT, base_color="black", hovering_color="green")
```

```
    # Showing buttons
    for button in [Addition,Subtraction,Multiply,Divide,Leaderboard]:
        button.changeColor(MOUSE_POS)
        button.update(SCREEN)

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if Addition.checkForInput(MOUSE_POS):
                run_game('addition')

            if Subtraction.checkForInput(MOUSE_POS):
                run_game('subtract')

            if Multiply.checkForInput(MOUSE_POS):
                run_game('multiply')

            if Divide.checkForInput(MOUSE_POS):
                run_game('divide')

            if Leaderboard.checkForInput(MOUSE_POS):
                leader_score()


    pygame.display.update()
```

This code creates a menu screen for a game or application. On this menu, you have a selection of options, and you can click on these options to perform different actions. Here is the break down what's happening in the code:

Loading Images and Setting Up the Screen:

- The code loads an image, typically representing a maths symbol, and scales it to a specific size.
- The screen is filled with a white background, and the image is displayed at a certain position.

Drawing Lines and Rectangles:

- It draws black and grey rectangles on the screen to create a visual separation and background for the buttons.

Button Creation:

- The code creates several buttons, each corresponding to a different action or option.
- These buttons are customised with specific text, positions, colours, and hover colours.
- The buttons include "Addition," "Subtraction," "Multiplication," "Division," and "Leaderboard."

Displaying Buttons:

- It loops through the list of buttons and manages their appearance.
- The button text changes colour when the mouse hovers over it (a visual indication of interactivity).

Displaying the Menu:

- The game menu is continuously displayed on the screen, and the user can interact with it.

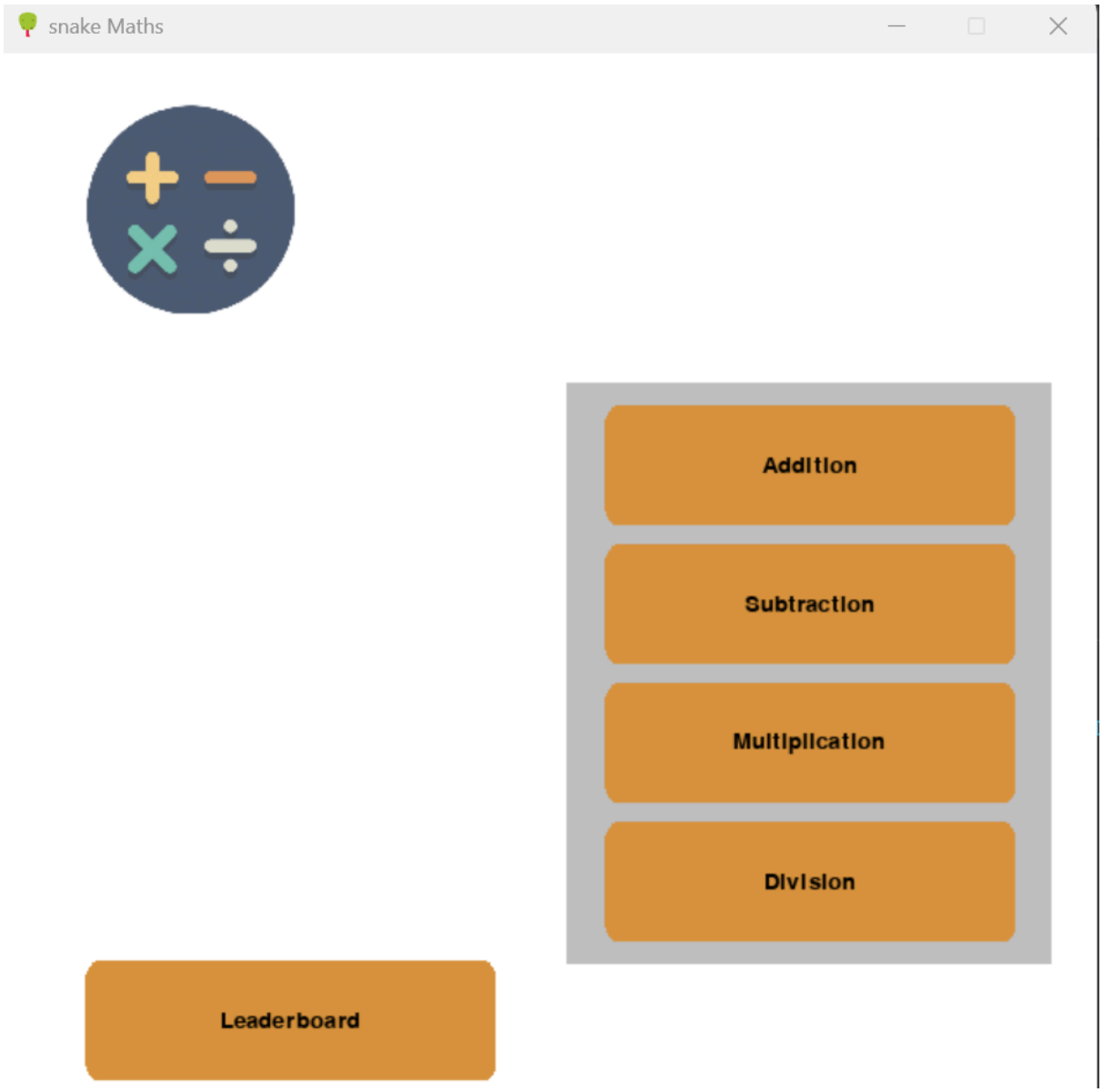
In summary, this code creates a menu screen with buttons for different options. When you click on a button, it initiates various actions, such as starting different math games or showing a leaderboard. The "Leaderboard," "Division," "Multiplication," and "Subtraction" buttons are additional options you can select from the menu.

```
# Showing buttons
for button in [Addition,Subtraction,Multiply,Divide,Leaderboard]:
    button.changeColor(MOUSE_POS)
    button.update(SCREEN)

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    if event.type == pygame.MOUSEBUTTONDOWN:
        if Addition.checkForInput(MOUSE_POS):
            run_game('addition')

        if Subtraction.checkForInput(MOUSE_POS):
            run_game('subtract')

        if Multiply.checkForInput(MOUSE_POS):
            run_game('multiply')

        if Divide.checkForInput(MOUSE_POS):
            run_game('divide')

        if Leaderboard.checkForInput(MOUSE_POS):
            leader_score()


pygame.display.update()
```

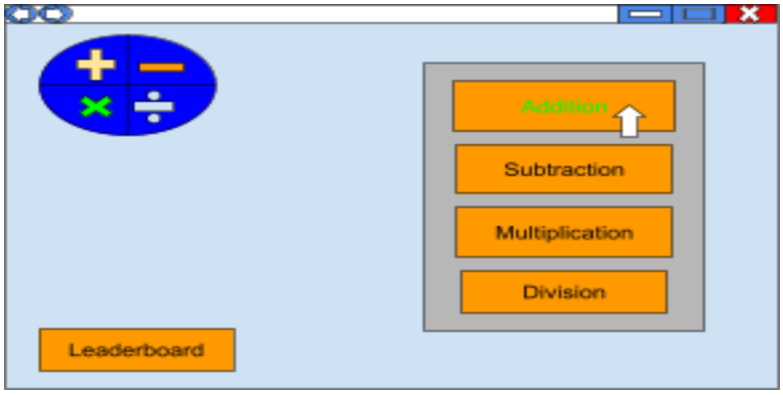
Handling User Input:

- The code continuously checks for user input and responds to events.
- If the user clicks on a button with their mouse, it triggers a specific action:
    - If the "Addition" button is clicked, it runs a game related to addition.
    - If the "Subtraction" button is clicked, it runs a subtraction game.
    - If the "Multiplication" button is clicked, it runs a multiplication game.
    - If the "Division" button is clicked, it runs a division game.
    - If the "Leaderboard" button is clicked, it displays the leaderboard.
      Note: currently the leaderboard has no functionality as we are just making the button and will add all the functionality later on.

The result:



As you can see this is practically identical to what we were hoping to have in the design section. Below I am printing the image of the design for a side by side comparison:



If you hover on it with the cursor the result is:

The text turns green as planned in the design section

INITIALIZING CLASS

```
class GAME:
    initial_timer_duration = 10000  # Initial timer duration (10 seconds)
    def __init__(self,mode):
        self.SCREEN = SCREEN
        self.fruit = FRUIT()
        self.fruit2 = FRUIT()
        self.snake = Snake()
        self.stone = STONE()
        self.mode = mode
```

This code defines a **GAME** class, which is to be part of a game or simulation. It manages essential game elements like the player character (a snake), fruit, and stones. The class also has a mode attribute to determine the game's mode, and it starts with an initial timer duration of 10 seconds.

Problem 10 :GENERATING QUESTION

```
# Maths Logic and functionality
def generate_question(self):

    num1 = random.randint(1, 10)
    num2 = random.randint(1, 10)

    self.incorrect_answer = self.generate_incorrect_answer(self.correct_answer)
    if self.mode != "":
        if self.mode =='addition':
            self.correct_answer = num1 + num2
        if self.mode =='subtract':
            self.correct_answer = num1 - num2

        if self.mode =='multiply':
            self.correct_answer = num1 * num2

        if self.mode =='divide':
            self.correct_answer = num1 // num2
```

In this code, there's a function called generate_question that creates maths questions for the game. It randomly selects two numbers, num1 and num2, within the range of 1 to 10. However, there's a challenge encountered when generating questions for the "divide" mode. In the "**divide**" mode, dividing these random numbers might produce float (decimal) numbers, which may not be suitable for a basic maths quiz. The correct answer to the question is calculated based on the selected game mode, which can be "**addition**," "**subtract**," "**multiply**," or "**divide**." The code also generates an incorrect answer, but it appears that the incorrect answer is generated before the correct answer is calculated, which may lead to issues when selecting the

incorrect answer. It might be more appropriate to generate the incorrect answer based on the correct answer.

Coded Solution 10:

```
if self.mode =='divide':
    while num1 < num2 or num1 == 0 or num2 == 0 or num1 % num2 != 0:
        num1 = random.randint(1, 40)
        num2 = random.randint(1, 10)
    self.correct_answer = num1 // num2
```

This code snippet specifically handles the "**divide**" mode. It ensures that the generated **num1** and **num2** values meet certain conditions: **num1** should be greater than or equal to **num2**, and both numbers should not be equal to zero. Additionally, it ensures that the division of **num1** by **num2** results in a whole number (no remainder). By implementing this code, the challenge of generating suitable division questions with integer answers has been successfully solved.

3.9 Milestone 4 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Tes met? Yes/No
1	Load mute image	Typical	Correctly image has been loaded	Yes Reference: Page 126
2	Load play again image	Typical	Correctly image has been loaded	Yes Reference: Page 126
3	Load home button	Typical	Correctly image has been loaded	Yes Reference: Page 126
4	Ensure mute button works	Typical	Button correctly works	Yes reference: Testing G...

				Youtube link: https://youtu.be/KtKol7uSn8Y
5	Ensure play again button works	Typical	Button correctly works	Yes reference: Testing G... Youtube Link: https://youtu.be/KtKol7uSn8Y
6	Ensure home button works	Typical	Button correctly works	Yes reference: Testing G... Youtube Link: https://youtu.be/KtKol7uSn8Y
7	Test GUI buttons	Typical	All the buttons for all the modes should work	Yes reference: Testing G... Youtube Link: https://youtu.be/KtKol7uSn8Y

All these tests are shown in the link: https://youtu.be/KtKol7uSn8Y

User feedback 4:

-In response to the fourth prototype, the user commended the inclusion of additional menu options like "Play Again" and "Mute."
-They found these features convenient and user-friendly, enhancing the overall gaming experience.

Reflections 4:

The positive reception of the new menu options validates the effort put into refining the user interface. I'll continue to prioritise user convenience and accessibility in future updates

3.10 Milestone 5 start:

Time to create the leaderboard

```
# Main loop
def leader_score():
    while True:
        MOUSE_POS = pygame.mouse.get_pos()

        SCREEN.fill((255, 255, 255))  # Use a tuple for the color

        # Draw rows
        draw_rows()

        # Create a "Back" button
        Back = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-190,50)), pos=(80,40
                            text_input="Back", font=SCORE_FONT, base_color=(0, 0, 0), hovering_color=(0, 255, 0))

        for button in [Back]:
            button.changeColor(MOUSE_POS)
            button.update(SCREEN)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if Back.checkForInput(MOUSE_POS):
                    menu()

        pygame.display.update()
```

This code defines a Pygame function, leader_score(), that displays a leader score screen. The screen consists of rows, and it includes a "Back" button. The button changes colour when hovered over and returns to the main menu if clicked. The function runs in an infinite loop, updating the display based on user input and mouse position.

DATABASE

```
# Setting up the database
import sqlite3 as sq

db = sq.connect("database.db")

try:
    st = """
        CREATE TABLE Addition_table
        (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
        """
    db.execute(st)
    db.commit()

        # Insert data only once
    insert = "INSERT INTO Addition_table(Name1, Score1,Name2, Score2,Name3, Score3) VALUES ('{}','{}','{}','{}','{}','{}')".format("
    db.execute(insert)
    db.commit()

except Exception as e:
    print("Error:", e)

# Getting the info from the database
fetc = """SELECT * FROM Addition_table"""
data = db.execute(fetc)
Addition_elements = []
for row in data:
    # Appending each row as a tuple
    Addition_elements.append((row[0], row[1]))
    Addition_elements.append((row[2], row[3]))
```

Database Connection (sqlite3):

- The code establishes a connection to a SQLite database named "database.db" using the sqlite3 module.

Table Creation (CREATE TABLE):

- It attempts to create a table named "Addition_table" in the database.
- This table is intended to store high scores or data related to an "Addition" game. It has six columns: Name1, Score1, Name2, Score2, Name3, and Score3.

Data Insertion (INSERT INTO):

- Initial data is inserted into the "Addition_table" table only if the table is successfully created.
- The inserted data represents placeholders for high scores. Three rows are added, each containing two pairs: a player's name and their corresponding score.
- These initial values are set to "-" for player names and 0 for scores.

Error Handling (try-except):

- The code includes error handling to catch any exceptions that may occur during table creation and data insertion.
- If an error occurs, it is printed out for debugging.

Data Retrieval (SELECT *):

- The code fetches data from the "Addition_table" using a SQL SELECT statement.
- It retrieves all the rows from the table and stores them in a list called Addition_elements.
- Each row from the table is converted into a tuple containing a player's name and their corresponding score. These tuples are appended to the list.

In summary, this code sets up a SQLite database, creates a table for storing high scores related to an "Addition" game, inserts initial placeholder data, and retrieves the stored data from the table. It is a foundational step for managing and displaying game-related statistics or scores.

Updating database

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_RETURN: # Restart the game on spacebar press
        if self.get_name:
            if self.mode == 'addition':
                if self.score> int(Addition_elements[0][1]):
                    Insert=f'''UPDATE Addition_table SET Score1={self.score}'''
                    Insert2 = f'''UPDATE Addition_table SET Name1 = '{self.name}' '''

                elif self.score> int(Addition_elements[1][1]):
                    Insert=f'''UPDATE Addition_table SET Score2={self.score}'''
                    Insert2 = f'''UPDATE Addition_table SET Name2 = '{self.name}' '''

                elif self.score> int(Addition_elements[2][1]):
                    Insert=f'''UPDATE Addition_table SET Score3={self.score}'''
                    Insert2 = f'''UPDATE Addition_table SET Name3 = '{self.name}' '''
```

It checks if a player's score should be inserted into the high score table and performs the necessary SQL updates. Here's a description of what this code does:

Condition for Updating Scores (if self.get_name):

- It first checks if the get_name attribute is True, indicating that the player has entered their name for recording high scores.

Updating High Scores (UPDATE Statements):

- Within the condition, it further checks the game mode (e.g., 'addition') and the player's score.
- If the player's score is greater than the first place score (the highest), it prepares SQL UPDATE statements to update the highest score and player name.
- If the player's score is greater than the second place score, it prepares SQL UPDATE statements for the second place.
- If the player's score is greater than the third place score, it prepares SQL UPDATE statements for the third place.

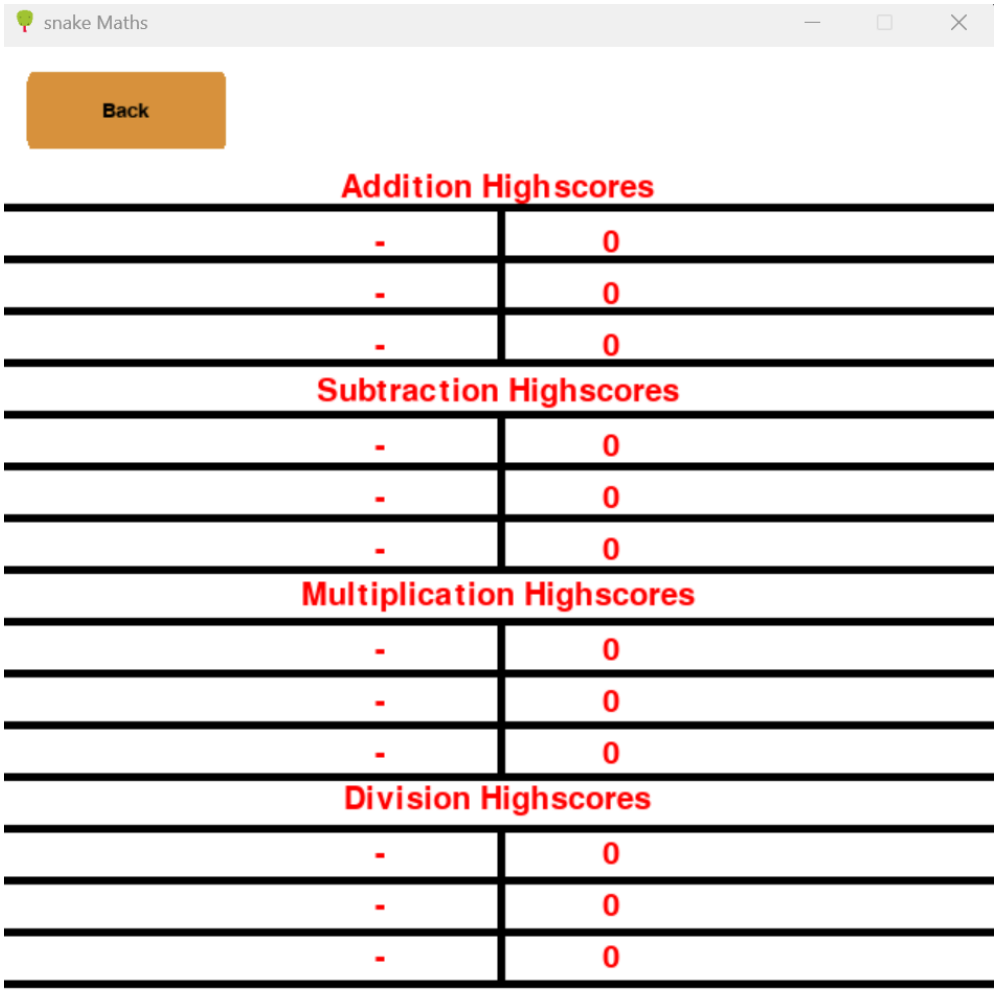
Repetition for Different Game Modes:

- The code snippet mentioned is focused on updating high scores for the 'addition' mode.

- Similar logic is likely repeated for other game modes (e.g., 'subtraction,' 'multiplication,' and 'division').
- For each mode, the code checks if the player's score surpasses the top three scores and updates the database accordingly.

In summary, this code is responsible for updating high scores in a database for different game modes based on player performance. It checks the player's score, compares it to the existing high scores, and updates the database if the player's score qualifies for one of the top three positions. The same logic is applied for multiple game modes, ensuring that high scores are accurately recorded for each mode

The final result:

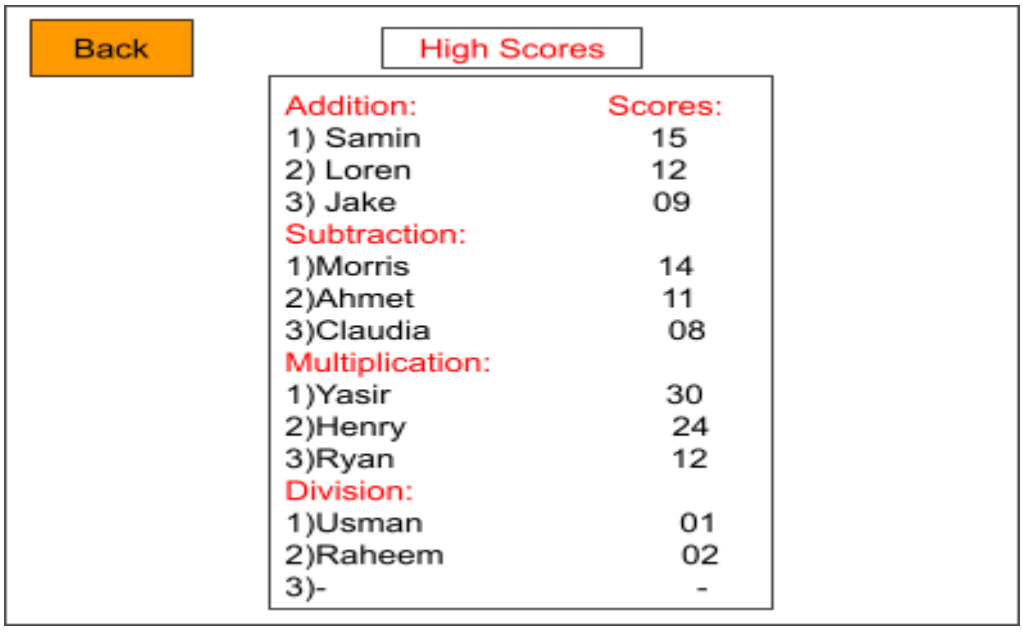


Note: As of now there are no names showing as no one has played the game yet, however this will gradually fill up as people start playing the game and gain highscores.

This is after achieving a highscore with a username named “Testing1”:

Multiplication Highscores	
Testing1	20
-	0
-	0

As you can see this is practically identical to what we were hoping to have in the design section. Below I am printing the image of the design for a side by side comparison:



3.11 Milestone 5 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Test Met? Yes/No
1	Create a leaderboard	Typical	Leaderboard correctly created	Yes Reference: Page 139
2	Update leaderboard	Typical	Leaderboard successfully updated	Yes Reference:

				Page 139 Here you can see testing got added.

User feedback 5:

-After experiencing the fifth prototype, the user appreciated the implementation of a leaderboard feature.
-They found it motivating to compare their scores with others and expressed excitement about the competitive aspect it introduced.

Reflections 5:

The user's positive reaction to the leaderboard underscores its value in fostering engagement and competition. I'll focus on further optimisation of this feature to ensure it remains a compelling aspect of the game.

3.12 MileStone 6 start:

Log in system

Now that we have created all the game functionality and GUI it is time to move onto the login system

To avoid circular imports I have decided to add both my login and registration system under one python script

```
def create_pin_entry_window():
    global window2
    # Create the window
    window2 = Tk()
    window2.title("PIN Entry")

    window2.geometry(f"{200}x{300}+{700}+{400}")
    window2.configure(bg="#2f2f30")

    # Create and place widgets
    pin_label = Label(window2, text="Enter 4-digit PIN:", bg="#2f2f30", fg="white")
    pin_label.pack(pady=10)

    global pin_entry
    pin_entry = Entry(window2, show="*", bg="white", fg="black", justify="center", font=("Helvetica", 16))
    pin_entry.pack(pady=10)

    # Create number buttons
    button_frame = Frame(window2, bg="#2f2f30")
    for i in range(1, 10):
        button = Button(button_frame, text=str(i), command=lambda i=i: enter_digit(i), width=4, height=2,bg='orange')
        button.grid(row=(i-1)//3, column=(i-1)%3, padx=5, pady=5)

    zero_button = Button(button_frame, text="0", command=lambda: enter_digit(0), width=4, height=2,bg='orange')
    zero_button.grid(row=3, column=1, padx=5, pady=5)

    clear_button = Button(button_frame, text="Clear", command=clear_pin, width=4, height=2,bg='orange')
    clear_button.grid(row=3, column=0, padx=5, pady=5)
```

```
submit_button = Button(button_frame, text="OK", bg='orange', command=check_pin,width=4, height=2)
submit_button.grid(row=3, column=2, padx=5, pady=5)

button_frame.pack()
```

Create Window:

- Creates a Tkinter window named `window2` for PIN entry.
- Configures window size, position, and background color.

Create Widgets:

- Creates a label for instructing the user to enter a 4-digit PIN.
- Creates an entry widget for the PIN with obscured characters (using `show="*"`).

Create Number Buttons:

- Creates buttons for digits 1 through 9 and 0.
- Each button is associated with a command (using `lambda`) that calls the `enter_digit` function with the corresponding digit.

Clear and OK Buttons:

- Creates a "Clear" button that calls the `clear_pin` function.
- Creates an "OK" button that calls the `check_pin` function.

Packing:

- Packs the label and PIN entry widgets.
- Packs the button frame containing the numeric and control buttons.

This function essentially creates a simple PIN entry window with numeric buttons, allowing the user to enter a 4-digit PIN. The window includes buttons for each digit, a "Clear" button, and an "OK" button. The entered PIN is obscured with asterisks.

```
def enter_digit(digit):
    current_pin = pin_entry.get()
    if len(current_pin) < 4:
        pin_entry.insert(END, str(digit))

def clear_pin():
    pin_entry.delete(0, END)

def check_pin():
    entered_pin = pin_entry.get()
    PIN = '1234'

    if entered_pin.isdigit() and len(entered_pin) == 4 and entered_pin == PIN:
        result_label.config(text="PIN Accepted")
        window2.destroy()
        set_up()  # Call your set up function
    else:
        result_label.config(text="Invalid PIN")
```

The `enter_digit` function appends a digit to the PIN entry if the current length is less than four. The `clear_pin` function deletes the PIN entry's content. The `check_pin` function compares the entered 4-digit PIN with a correct PIN ('1234'). If the entered PIN is valid, it updates a label to "PIN Accepted," closes the window, and calls a setup function. If the entered PIN is invalid, it updates the label to "Invalid PIN." These functions collectively handle the process of entering and validating a PIN in the Tkinter-based interface, providing clear feedback on PIN acceptance or rejection.

You may wonder where we got all the imports and icons from so I pasted the code here for reference:

The result:

```
from tkinter import *
from PIL import Image
from tkinter import messagebox
import ast
from pygame.locals import *
from subprocess import call



#File conversions:
input_png = 'LogInIcon.png'
output_ico = 'LogInIcon.ico'
input_jpg = 'RegisterImage.jpeg'
output_ico2 = 'RegisterImage.ico'

# Open the PNG file
Icon_img = Image.open(input_png)

# Save the icon as an ICO file
Icon_img.save(output_ico)

input_png = 'LogInIcon.png'
output_ico = 'LogInIcon.ico'

# Open the PNG file
Icon_img = Image.open(input_png)

# Save it as an ICO file
Icon_img.save(output_ico)
```

As you can see this is practically identical to what we were hoping to have in the design section. Below I am printing the image of the design for a side by side comparison:

Mainloop

```
def login():
    username=user.get()
    password=code.get()

    file=open('datasheet.txt','r')
    d=file.read()
    r=ast.literal_eval(d)
    file.close()
    try:
        file = open('datasheet.txt', 'r')
        data = file.read()
        user_data = ast.literal_eval(data)
        file.close()

        if username in user_data and password == user_data[username]:
            # Close the login window
            window.destroy()
        else:
            messagebox.showerror('Invalid', 'Invalid username or password')
    except Exception as e:
        messagebox.showerror('Error', 'An error occurred: ' + str(e))


    if username in r.keys() and password == r[username]:

        try:

            window.destroy()
```

```
        except:
            pass

        try:

            window2.destroy()
        except:
            pass



        from subprocess import call
        call(["python", "main 2.py", "--username", username])

    else:
        messagebox.showerror('Invalid','invalid username or password')
def on_login_button_pressed():
    # window.destroy()
    pass
```

-The function reads data from 'datasheet.txt' and attempts to evaluate it as a dictionary using `ast.literal_eval`.

-It then compares the entered username and password with the stored data. If a match is found, it closes the Tkinter window (`window`).

-If the credentials do not match or an exception occurs during the process, it displays an appropriate error message.

-Additionally, it checks if the entered credentials match a backup data structure (`r`) and closes another Tkinter window (`window2`) if it exists.

The overall design ensures a robust user authentication process, with error handling, backup checks, and proper closure of associated windows, providing a secure and user-friendly login experience.

```
def set_up():
    global window
    window = Toplevel(window)
    window.title("Register")
    window.geometry('925x500+300+200')
    window.configure(bg='#606060')
    window.resizable(False,False)


    def register():
        username=user.get()
        password=code.get().strip()
        conform_password=confirm_code.get().strip()
```

`set_up` Function:

- Creates a new Tkinter window (`window`) for user registration using `Toplevel`.
- Sets window attributes such as title, size, background colour, and makes it non-resizable.

`register` Function:

- Defines an internal function `register` to handle the user registration process.
- Retrieves the entered username, password, and confirmed password from Tkinter entry widgets (`user`, `code`, and `confirm_code`).
- Uses `.strip()` to remove leading and trailing whitespaces from the entered passwords.

By encapsulating this logic within a dedicated function, the code achieves modularity and clarity, allowing for easier maintenance and understanding of the registration functionality. This approach facilitates separation of concerns, with `set_up` focused on window configuration and `register` on user input processing, contributing to a more maintainable and readable codebase.

Validations:

-After I have decided to add some validations for both the username and password

The code for it:

```
# Condition 1: Username must not be empty
if not username:
    messagebox.showerror('Invalid', 'Username cannot be empty')
    return
 # Condition 2: Username must not contain special characters
if not username.isalnum():
    messagebox.showerror('Invalid', 'Username must only contain alphanumeric characters')
    return
 # Condition 3: Password must be at least 6 characters long
if len(password) < 6:
    messagebox.showerror('Invalid', 'Password must be at least 6 characters long')
    return
# Condition 4: Password must contain an uppercase letter
if not any(char.isupper() for char in password):
    messagebox.showerror('Invalid', 'Password must contain at least one uppercase letter')
    return
# Condition 5: Password must contain a number
if not any(char.isdigit() for char in password):
    messagebox.showerror('Invalid', 'Password must contain at least one number')
    return
```

```
if password == conform_password:
    try:
        file=open('datasheet.txt','r+')
        d=file.read()
        r=ast.literal_eval(d)

        dict2={username:password}
        r.update(dict2)
        file.truncate(0)
        file.close()

        file=open('datasheet.txt','w')
        w=file.write(str(r))

        messagebox.showinfo('Register','Successefully registered')

    except:
        file=open('datasheet.txt','w')
        pp=str({'username':'password'})
        file.close()
else:
    messagebox.showerror('Invalid', 'Both Passwords have to match')
```

-The code checks if the entered password matches the confirmed password.

-If true, it tries to update the contents of 'datasheet.txt' with the new user's credentials.

-It reads the existing dictionary from the file, updates it with the new credentials, truncates the file, and writes the updated dictionary back to the file.

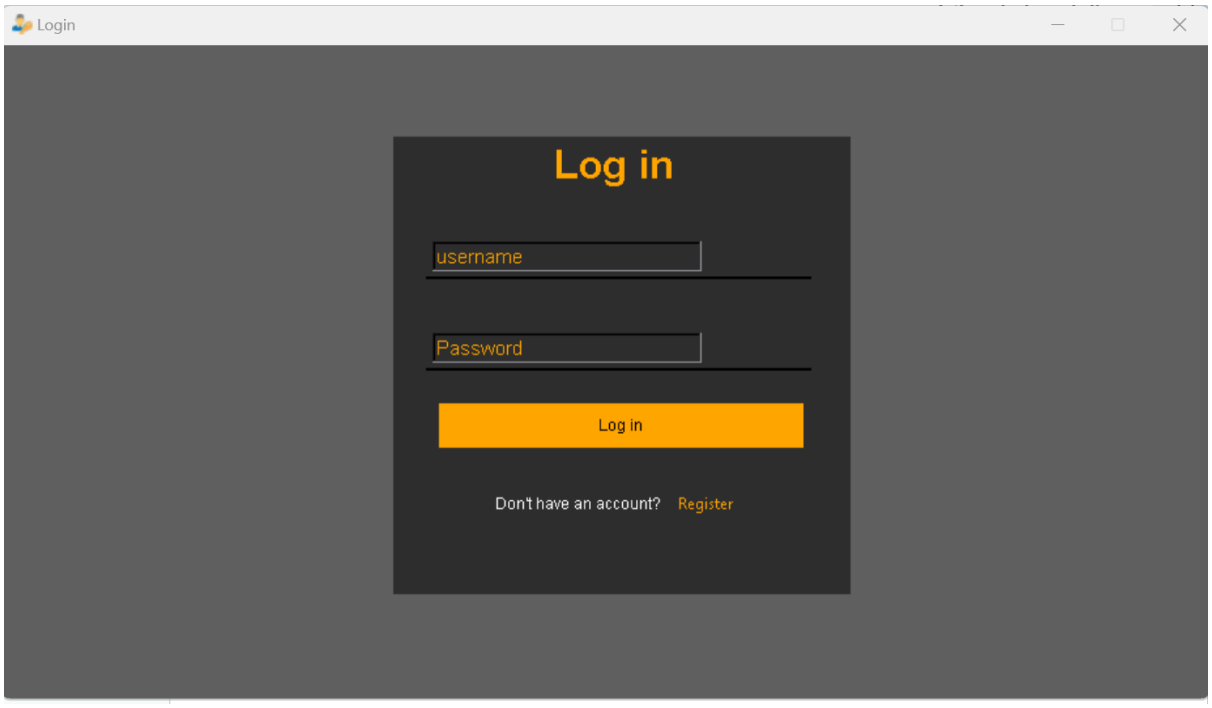
-If an exception occurs during this process, it handles the exception by creating a new file with default credentials.

-If the passwords do not match, it displays an error message using `messagebox.showerror`.

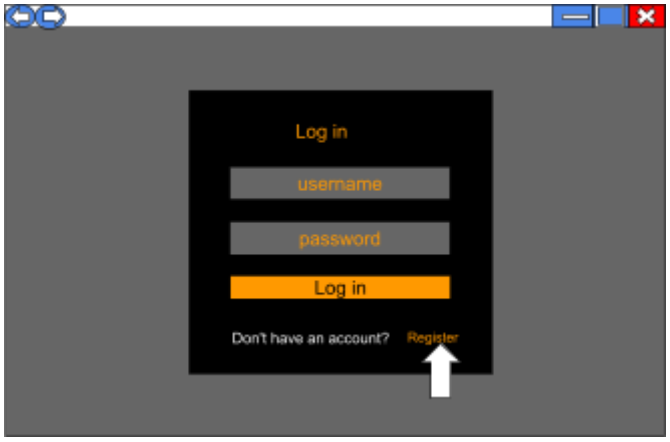
The code ensures the integrity of the registration process by checking if the entered password matches the confirmed password before proceeding. In the case of a match, the code attempts to update an existing dictionary in the 'datasheet.txt' file with the new user's credentials, providing a structured storage format. The use of a try-except block ensures graceful handling of potential exceptions during file manipulation, promoting robustness. If an exception occurs, the code creates a new file with default credentials to prevent data corruption. Additionally, displaying informative messages through `message box` enhances user interaction, offering clear feedback on the success or failure of the registration attempt and contributing to a positive user experience

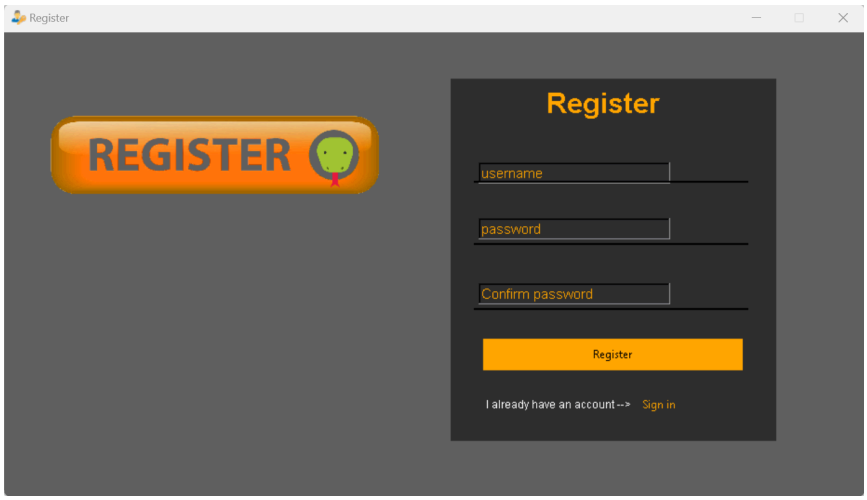
After making the GUI look nice by defining all the frames and entities result was:

For the log in:



As you can see this is practically identical to what we were hoping to have in the design section. Below I am printing the image of the design for a side by side comparison:





For the register:

As you can see this is practically identical to what we were hoping to have in the design section. Below I am printing the image of the design for a side by side comparison:

Getting username:

```
if "--username" in sys.argv:
    username_index = sys.argv.index("--username")
    if username_index + 1 < len(sys.argv):
        username = sys.argv[username_index + 1]
        print("Username:", username)
    else:
        print("No username provided.")
else:
    print("Username not found in command-line arguments.")
```

This Python code snippet is designed to extract a username from the command-line arguments passed to a script. Here's a description of how it works:

if "--username" in sys.argv::

- This condition checks whether the string "--username" is present in the list sys.argv. sys.argv is a list that contains the command-line arguments passed to the script.

username_index = sys.argv.index("--username"):

- If the "--username" string is found in the command-line arguments, this line retrieves the index at which it is located in the sys.argv list.

if username_index + 1 < len(sys.argv)::

- This conditional statement checks if there is an argument following "--username" (i.e., if the index username_index + 1 is within the bounds of the sys.argv list).

username = sys.argv[username_index + 1]:

- If a valid username argument is found, it extracts the username from the list by accessing the element at the index username_index + 1.

print("Username:", username):

- The code then prints the extracted username to the console.

else:

If there is no username argument after "--username," this block of code is executed.

print("No username provided."):

- This line informs the user that no username argument was found after "--username" in the command-line arguments.

else: (outside the initial if block):

- If the "--username" string is not found in the command-line arguments, this block is executed.

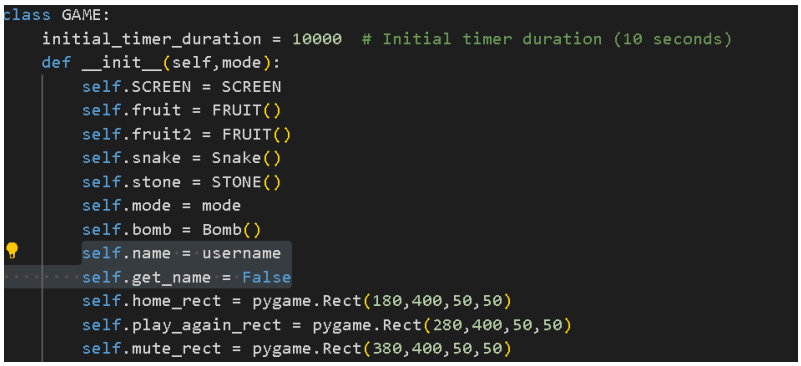
print("Username not found in command-line arguments."):

- This message is printed to indicate that the script did not receive a "--username" argument.

In summary, this code snippet parses the command-line arguments to extract and print a username when "--username" is provided as an argument. It also handles cases where no username is provided or when the "--username" argument is missing.

PUT USERNAME FOR GAME

This code snippet is a part of a Python GAME class and does the following:



```
class GAME:
    initial_timer_duration = 10000  # Initial timer duration (10 seconds)
    def __init__(self,mode):
        self.SCREEN = SCREEN
        self.fruit = FRUIT()
        self.fruit2 = FRUIT()
        self.snake = Snake()
        self.stone = STONE()
        self.mode = mode
        self.bomb = Bomb()
        self.name = username
        self.get_name = False
        self.home_rect = pygame.Rect(180,400,50,50)
        self.play_again_rect = pygame.Rect(280,400,50,50)
        self.mute_rect = pygame.Rect(380,400,50,50)
```

- It assigns the value of username to the instance variable self.name, indicating that the class manages and stores a user's name.

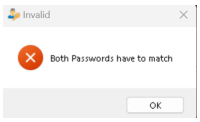
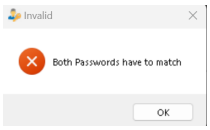
- The code initialises the boolean instance variable self.get_name to `False. It is used to indicate that the name already exists, preventing the need to ask for a name later.

3.13 Milestone 6 test review:

Test No	Test Description/Purpose and justification	Test Data (Erroneous, boundary, typical)	Expecting outcome	Test Met? Yes/No
1	Create register window	Typical	Window successfully created	Yes Reference: 149
2	Create buttons for register	Typical	Button successfully created	Yes Reference: Page 149
3	Add icon for the register page	Typical	Icon successfully added	Yes Reference: Page 149

4	Ensure the register buttons function correctly	Typical	Button function works	Yes reference: Testing R... Youtube Link: https://youtu.be/0D8_MtBKeKo
5	Ensures user types in an appropriate username and password for register	Typical	Ensures username entered meets a certain criteria	Yes reference: Testing R... Youtube Link:https://youtu.be/0D8_MtBKeKo Time stamp: 2:00
6	Ensure passwords have to match for account to be created	Typical	Ensure access is granted only if the passwords match	Yes reference: Testing R... Youtube Link: https://youtu.be/0D8_MtBKeKo Time stamp: 2:00
7	Ensured the details are saved to an external file	Typical	File are correctly saved to an external file	Yes reference: Testing R... Youtube Link: https://youtu.b

				e/0D8_MtBKeKo Time stamp: 2:10
9	Ensure only admin can register	Typical	Only allows admin to register	Yes reference: Pin Testing Youtube: https://www.youtube.com/watch?v=HwLgZHIMm2Y
10	Ensure can register only if correct pin is entered	Typical	Only allows registration to occur if the correct pin is entered.	Yes reference: Pin Testing Youtube: https://www.youtube.com/watch?v=HwLgZHIMm2Y Time stamp: 27 seconds
12	Create login window	Typical	Log window successfully created	Yes Reference: Page 148
13	Create buttons for login	Typical	Buttons successfully created.	Yes Reference: Page 148
14	Add icon for the login page	Typical	Icon successfully added	Yes Reference: Page 148

15	Ensure the login button works	Typical	Login button works accordingly	Yes reference: Testing L... Youtube: https://www.youtube.com/watch?v=HwLgZHIMm2Y Time stamp: 40 seconds
16	Ensure username entered is valid	Typical	Only appropriate usernames are accepted	Yes reference: Testing L... Youtube: https://youtu.be/uG_jvOkNhR0?si=M8GcCGeMtvlw_Q3z Time stamp: 40 seconds
17	Ensure password entered is valid	Typical	Only appropriate password gives access	Yes Reference: 
18	Ensure a messagebox is displayed if the details entered is correct	Typical	Displays message box if wrong data is being given.	Yes Reference: 

19	Ensure you can enter the register page from the login page	Typical	You can access the register page from the login page.	Yes Reference: Pin Testing Youtube: https://www.youtube.com/watch?v=HwLgZHIMm2Y Time stamp: 26 seconds

User feedback 6:

-Following the sixth prototype, the user praised the seamless integration of the registration and login system.
-They found the process straightforward and efficient, contributing to a smoother overall user experience.

Reflections 6:

The positive feedback on the registration and login system affirms the importance of prioritising user convenience and security. I'll continue to refine these systems to maintain their effectiveness and reliability.

4.0 Post development testing:

Test Number no	Test description/purpose and justification	Test Data	Expecting outcome	Actual Outcome
1	Evaluate overall game functionality	Typical: user navigates the snake throughout the grid, answers questions by	Snake moves appropriately, and the questions are displayed accurately, and	Link: https://www.youtube.com/shorts/DggeRCdk2Mo

		colliding with apples and on the same time avoiding bombs	game element interact correctly	
2	Check robustness of login and registration system	Erroneous: entering an incorrect password during login Boundary: inputting the maximum amount of characters for registering Typical: successfully registering an user	Proper error handling during login, appropriate response to boundary conditions and successful registration	Seen already throughout pages: 151 -155
3	Validate effectiveness of the educational component	Typical: answering questions while it gets more difficult to get the correct answer due to the increase in the snakes length and the decrease in time left to answer	Questions should not be too hard or too easy, just hard enough for a kid to calculate the maths mentally. However, this process gets harder as time left decreases.	Link:https://www.youtube.com/watch?v=ahCqN3gxyQE&t=12s
4	Test the game's response to rapid and simultaneous user inputs.	Typical	Rapidly press multiple control keys or buttons simultaneously to assess the game's responsiveness and handling of	Link: Multiple Key str… Youtube; https://youtu.be/5bdpdzLovtk

			concurrent inputs.	
5	Evaluate the game's performance under different levels of hardware resources.	Typical	Run the game on machines with varying hardware specifications (e.g., CPU, GPU, RAM) to assess performance and optimise resource usage.	Link: https://youtu.be/9sEm8Fafl8E
6	Test the game's behaviour during system interruptions (e.g., notifications, updates).	Erroneous	Simulate system interruptions such as notifications or updates while playing the game to ensure it handles interruptions gracefully.	Link: Multiple proces… Youtube:https://youtu.be/enOebfS4hs8
7	Assess the leaderboard functionality	Typical	Check if leaderboard gets updated instantly or takes some time to get updated	Link: Checking leade… Youtube:https://youtu.be/wIQhk7mFN64 Time stamp: 47 seconds
8	Verify the game's behaviour when running concurrently with other applications or processes.	Typical	Run the game alongside other applications or processes to assess its performance, resource usage, and behaviour in multitasking environments.	Link: Multiple proces… Youtube: https://youtu.be/enOebfS4hs8
9	Test the game's audio feedback system (e.g., sound effects,	Typical	Enable audio feedback and assess the	Link: https://www.youtube.com/shorts/DggeRCdk

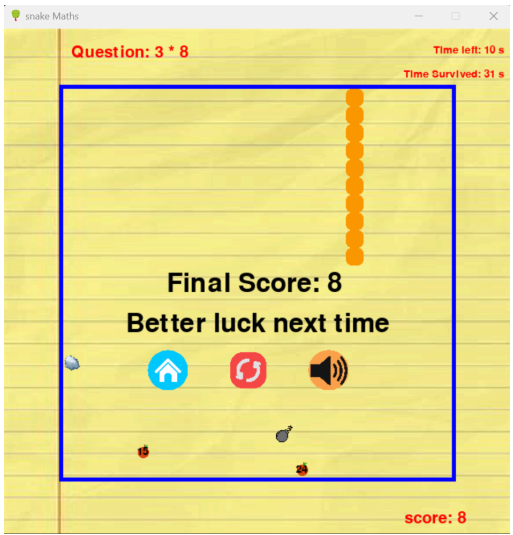
	background music).		quality, clarity, and appropriateness of sound effects and background music during different gameplay scenarios.	2Mo
10	Verify the game's behaviour during transition phases .	Typical	Start the game or transition between different game scenes to ensure smooth transitions, proper initialization, and correct display of relevant information or prompts.	Link:https://youtu.be/3Zone9O4CnM?si=4rQDWUwgIq2bqYYk

Test 1: Evaluate overall game functionality

Outcome: Fully met

Explanation: The game successfully completed all requirements, including boundary handling, smooth movement, accurate object spawning, and correct display of questions and answers. The evidence video demonstrates the game's seamless interaction with its objects and logic, confirming the achievement of all testing criteria.

Image where this is shown:

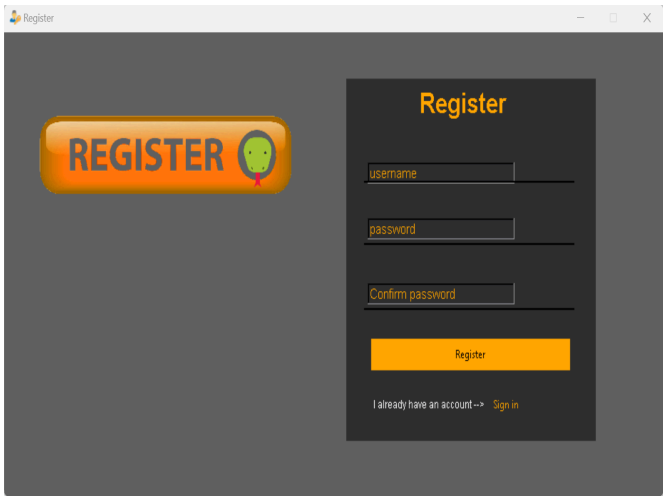


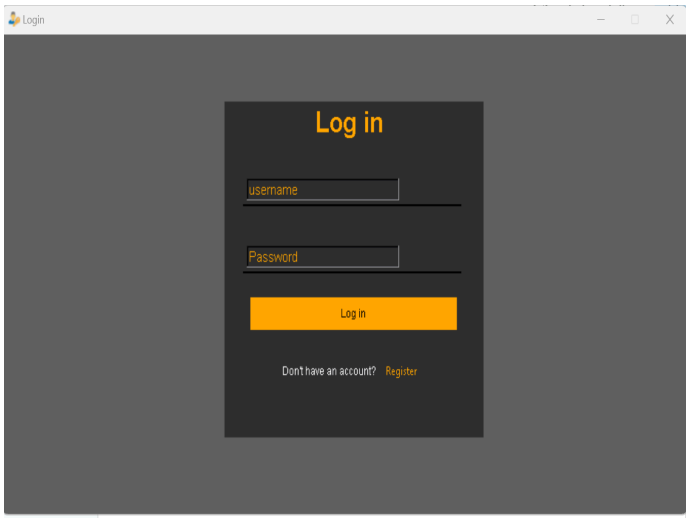
Test 2: Check robustness of login and registration system

Outcome: Fully met

Explanation: The login and registration system exhibited robustness across various scenarios, effectively handling erroneous login attempts, boundary conditions, and typical registration processes. Clear feedback and user-friendly processes were evident, showcasing reliability and scalability.

Shown in the images:





Test 3: Validate effectiveness of the educational component

Outcome: Partially met

Explanation: While the educational component provided questions at an adequate difficulty level and appropriately increased in challenge over time, there were some limitations. The program's effectiveness was hindered by the simplicity and repetitiveness of questions, especially in the division mode. Additionally, issues with displaying certain numbers on apples and potential curriculum limitations impacted the program's overall effectiveness, particularly for older primary years.

Test 4: Assess input handling robustness for snake movement

Outcome: Partially met

Explanation: During testing, I observed that rapidly switching between keys or pressing two keys simultaneously occasionally caused the snake to unexpectedly die. This inconsistency in input handling could be attributed to limitations within the Pygame

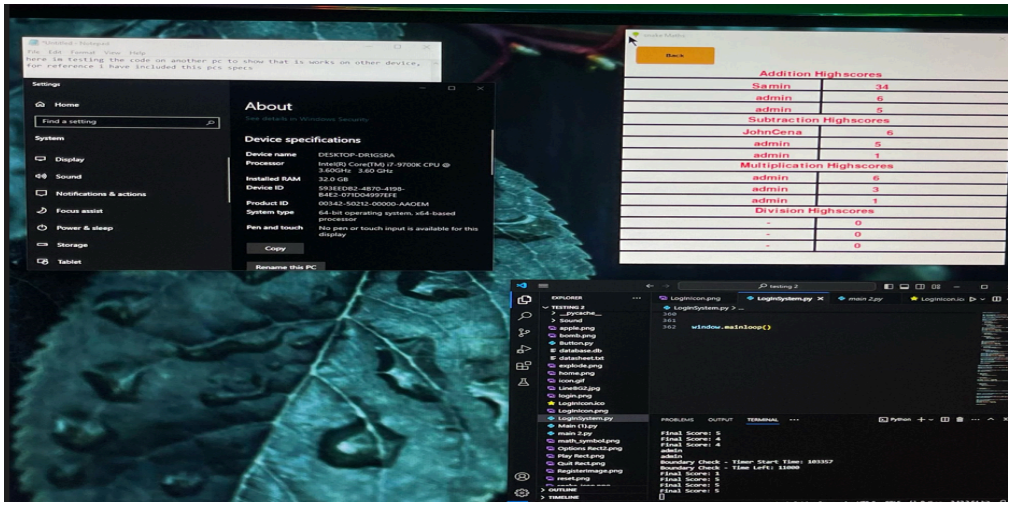
module and the constraints imposed by frame rate processing. Pygame's event handling system, although versatile for game development, may struggle to accurately capture rapid key inputs or manage simultaneous key presses due to its single-threaded nature. Fluctuations in frame rate can exacerbate these issues, leading to unpredictable outcomes during gameplay.

Test 5: Evaluate the game's performance under different levels of hardware resources

Outcome: Fully met

Explanation: The game's performance was assessed across machines with varying hardware specifications, including CPU, GPU, and RAM. It demonstrated consistent performance and optimised resource usage across different setups, ensuring smooth gameplay and responsiveness. While hardware variations may influence performance to some extent, the game maintained satisfactory performance levels across all tested configurations, meeting expectations for adaptability and efficiency.

This is shown in the image below where I took the screenshot of the specs of the computer I was running the game on. The video link can be found in the table.

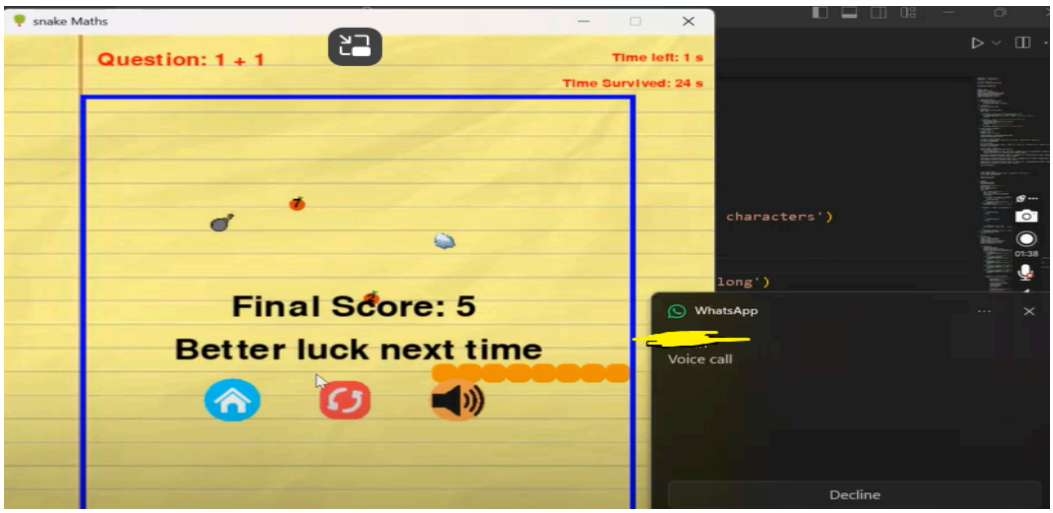


Test 6: Evaluate game behaviour during system interruptions

Outcome: Fully met

Explanation: Throughout the test, I observed the game's response to simulated system interruptions, such as notifications or updates. Despite these interruptions, the game maintained proper functionality, handling them gracefully without any noticeable issues. However, it's important to consider that the computer's hardware capabilities, such as available RAM and processing power, may impact the overall user experience during such interruptions. Nonetheless, the game itself performed as expected, demonstrating resilience in the face of system interruptions.

Image shown below where I had a person call me on whatsapp while playing the game:

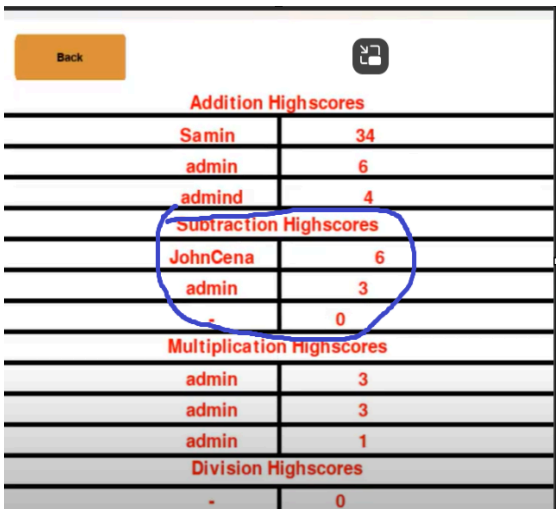


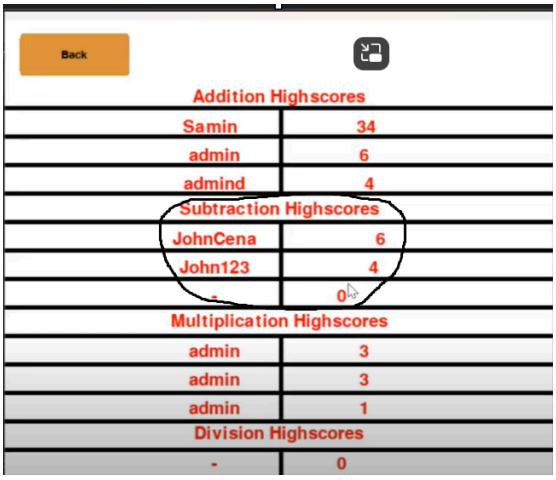
Test 7: Evaluate leaderboard functionality

Outcome: Partially met

Explanation: During testing, I assessed the leaderboard functionality to determine if it updated instantly or required some time to reflect changes. It was observed that the leaderboard did not update instantly within the game session; rather, it updated only once the program was closed and reopened. This behaviour indicates that while the leaderboard functionality is implemented, it does not update in real-time during gameplay. Instead, it requires a restart of the program to reflect the latest changes.

In the video I have logged in as John Cena and after playing the game. I checked the leaderboard. After checking initially it shows the table not updating but only updating after logging in back into the system:





Test 8: Evaluate game performance in multitasking environments

Outcome: Fully met

Explanation: During testing, I ran the game concurrently with other applications and processes to evaluate its performance and behaviour in multitasking environments. The game demonstrated satisfactory performance, maintaining smooth gameplay and responsiveness even when running alongside other applications. However, it is important to note that the behaviour may vary depending on the operating system and hardware components of the system. Overall, the game exhibited robustness and compatibility in multitasking scenarios.

Test 9: Evaluate game audio feedback system

Outcome: Fully met

Explanation: Upon enabling the audio feedback system, I assessed the quality, clarity, and appropriateness of sound effects and background music during various gameplay scenarios. The audio feedback was clear, synchronised with the game events, and enhanced the overall gaming experience. Both sound effects and background music played seamlessly without any distortion or delay, meeting the expected criteria for audio performance.

Test 10: Assess game behaviour during transition phases

Outcome: Fully met

Explanation: Testing the game's behaviour during transition phases, including starting the game and transitioning between different scenes, revealed smooth transitions, proper initialization, and correct display of relevant information or prompts. The game seamlessly navigates between scenes without any glitches or inconsistencies, ensuring a seamless and immersive gaming experience. All elements of the transition process functioned as expected, meeting the predefined criteria for transition behaviour.

4.1Usability features

For this section I will get one of my stakeholders to try my program and rate our usability features. For this investigation our humble stakeholder Ahmet Yalcin decided to volunteer about our usability features and explain how he felt about the program overall, telling us all the positive and all the negative aspects alongside suggestions on how we could improve it.

Interview based around Log in system:

Functionality:

- *Question:* Can you briefly describe the overall functionality of the login system in the game? Does it fulfil the intended purpose of allowing users to log in securely?
- *Response by ahmet:* The login system functions as intended, requiring users to enter a username and password. It checks the entered credentials against stored data and allows access upon successful login. The integration of a PIN entry window also adds an extra layer of security.

Robustness:

- *Question:* How robust is the login system in handling different scenarios, such as erroneous login attempts and typical login processes?
- *Response by ahmet:* The login system demonstrates robustness by effectively handling erroneous login attempts. Error messages are appropriately displayed, and the system reacts accordingly to different inputs. The use of a PIN for enhanced security is a positive aspect. However, further testing and validation could be considered to ensure the system's resilience against potential edge cases.

Usability:

- *Question:* Assessing the usability of the login system, are the user interface and controls intuitive? Is it easy for users to understand how to log in?
- *Response by ahmet:* The user interface is visually appealing, with a clean design and clear labels. The use of entry fields for username and password, along with buttons for actions like login, contributes to usability. The PIN entry window enhances security but might benefit from additional visual cues for user guidance during the entry process. The login process is straightforward, with sensible input validation.

Suggestions for Improvement:

- *Question*Based on the evaluation, are there any specific areas where improvements could be made in terms of functionality or usability?
- *Suggestions:* While the login system appears well-implemented, consider incorporating additional visual feedback or guidance for users during the PIN entry process. This could enhance the overall user experience and make the login system even more user-friendly. Additionally, thorough testing with various inputs and scenarios can further ensure the robustness of the login system.

Interview based around registration system:

Functionality:

- *Question:* As a dad whose son is in primary school, can you share how you and your child find the process of creating accounts and logging in? Does it seem straightforward for a young user?

- *Ahmet:* From a parent's perspective, the registration and login process is fairly straightforward. The system requires a username, password, and even adds an extra layer of security with a PIN, which I appreciate for my child's safety. The password criteria help in creating secure accounts, and the overall process seems manageable for my child.

Robustness:

- *Question:* As a parent, do you feel confident in the system's ability to handle different scenarios, especially when it comes to your child's interactions with passwords and the PIN?
- *Ahmet:* Yes, as a parent, I feel confident in the system's robustness. It effectively enforces password criteria, ensuring that my child creates a strong password. The addition of a PIN adds an extra layer of security, providing peace of mind. However, I'd like to see more visual cues during the PIN entry to make it even more user-friendly for my child.

Usability:

- *Question:* How does your child find the overall usability of the system? Is it easy for them to understand how to login and create accounts?
- *Ahmet:* My child generally finds the system user-friendly. The clear labels and entry fields make it easy for them to understand where to input their information. However, during the PIN entry, we noticed that some additional visual cues would be helpful for my child to better understand its purpose.

Suggestions for Improvement:

- *Question:* As a dad, are there any specific areas you'd like to see improved in terms of functionality or usability in the registration and login system?
- *Ahmet:* While the system is generally good, I would appreciate more visual cues during the PIN entry process. This would help my child understand the significance of the PIN and make the system even more user-friendly for them. Overall, it's a positive experience, but small improvements could enhance the system for young users in primary school.

Interview based about the actual game:

Gameplay Dynamics:

- *Question:* Ahmet, as a stakeholder with a program designed for your son in primary school, how do the different game modes enhance your son's experience in "MathSnake Adventure"? How did the design of gameplay dynamics balance entertainment with educational value, ensuring a seamless and user-friendly interaction?
- *Ahmet's Response:* The game modes aim to make learning maths enjoyable for my son. Gameplay dynamics were designed for functionality, providing a smooth and engaging experience. Usability is key, making it easy for my son to navigate modes while balancing fun and learning. The robust design ensures a reliable and consistent gaming environment.

Objectives and Challenges:

- *Question:* Ahmet, focusing on functionality and robustness, how do the game's objectives and challenges contribute to providing a seamless and engaging experience for your son in "MathSnake Adventure"? Specifically, how do objects like fruits and bombs enhance the gameplay while ensuring the overall robustness of the gaming experience?
- *Ahmet's Response:* The game's objectives and challenges are crafted for functionality, keeping my son actively involved. Fruits and bombs contribute to the game's overall functionality by providing interactive learning experiences. Usability is maintained by ensuring these elements are intuitive, and robustness is achieved by seamlessly integrating them into the gameplay to avoid any disruptions.

Leaderboard and Account System:

- *Question:* The inclusion of a leaderboard is interesting. As a stakeholder, how does this feature enhance the overall robustness of the game for your son? Additionally, could you elaborate on the decision to implement an account system for high scores and its impact on functionality and your son's gaming experience?
- *Ahmet's Response:* The leaderboard enhances the overall robustness by adding a competitive edge and motivating my son to improve scores. Functionality is achieved through the personalised touch of the leaderboard and account system, allowing my son to easily track his progress. Usability is maintained through a user-friendly interface, ensuring these features

5.0 Evaluation

Criteria met:

No.	Criteria	Justification	Met: Yes/No	Evidence
1	Display the score	This is to tell the player how well they are doing, encouraging them to collect more points	Yes	Image1,shown below the table Page: 88
2	Display the time the user has survived	This is to tell the player how well they are doing, encouraging them to survive for longer	Yes	Image1 Page: 117
3	Display the time left to answer question	To let the player know how long they have left	Yes	Image1 Page 117
3	Display two fruit objects	Allows the player to select the correct answer between the 2 options	Yes	Image1 Page 117
4	Randomly display a rock	This makes collision more likely, making the game progressively harder	Yes	Image1,image 2 Page 126
5	Randomly display a bomb	This is to make collision more likely, making the game progressively harder	Yes	Image1,image 2 Page 126
6	Display a boundary the	Restrict movement	Yes	Image1,image

	player cannot cross	beyond the game space, keeping the gameplay within the specified location.		2 Page 88,96(an issue was found in this page)
7	Display questions	This is to let the player formulate the answer in their head before selecting the right answer	Yes	Image1 Page 113
8	Have a leaderboard to show highest scores	To promote healthy competition	Yes	Image3 Page 139
9	Have a login system	To allow only user of that organisation to use the software	Yes	Page 148
10	Having a pin for registering	This is do that only admin can register students	Yes	Page 144
11	Having an explosion image	this is to make the game more aesthetic	Yes	Image4 This is for the initial prototype 

				Image4 shows the first prototype
12	Music playing	To add more life to the game	Yes	Page 101
13	Mute function	To stop the music playing	Yes	Image2,4 Page 134-135 these tests have already been conducted
14	Reload button	To allow the user to start a new game without having to start the program again	Yes	Image2,4 Page 134-135 these tests have already been conducted
15	Home button	To allow the user to access the home menu without having to start the program again	Yes	Image2,4 Page 134-135 these tests have already been conducted

Proof:

Image1:

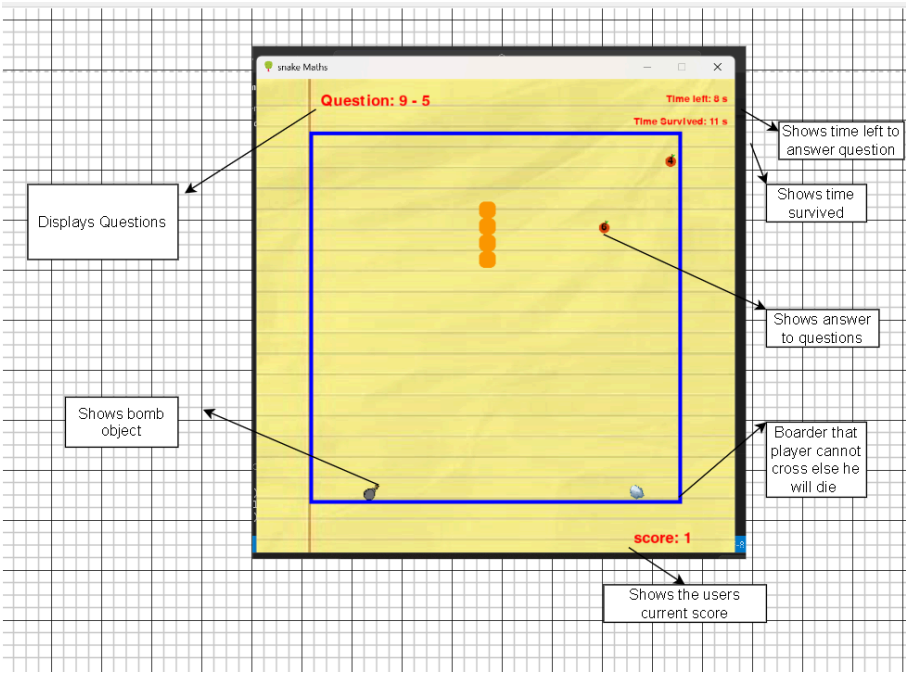
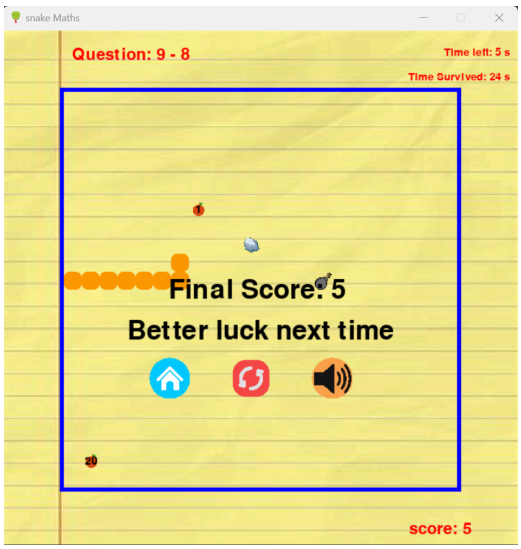


Image2:



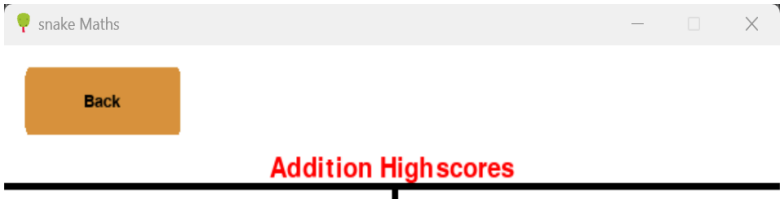
As you can see in these screenshots we proved the program's functionality for the success criterias 4,5,6 as well as further acting as evidence for all the other success criteria.

Success criteria 4: this criteria is filled by comparing it with image1 as you can see the rock indeed spawns in different location

Success criteria 5: this criteria is filled by comparing it with image1 as you can see the bomb indeed spawns in different locations

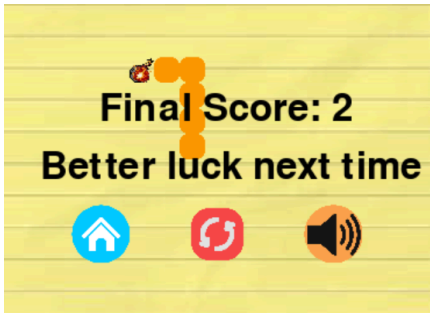
Success criteria 6: As annotated in the image in the first image the blue grid displays the space the player cannot cross and the image2 shows what would happen if the player tried to cross that perimeter.

Image3:



Success criteria 7: in this image it's clearly visible how 2 players so far played the game and the highest score was samin with a score of 34 which indeed is greater than the second player which has a score of 6

Image 4:



5.1 Maintenance and limitations

Scalability:

The game currently has a simple structure, but as we add more features and make the gameplay more complex, scalability becomes crucial. Without proper scalability measures, it may become hard to manage the codebase, which can make it difficult to implement new features and updates. Additionally, using inefficient data structures and algorithms can cause performance problems, especially on weaker devices or when there are many players online.As the game grows, it's essential to prioritise addressing scalability issues to ensure its long-term success.

Security:

While implementing the login system for my game, I have prioritised security through inclusion of a PIN-based authentication mechanism. This strategy provides an additional bar against unauthorised access by requiring users to enter a four-digit PIN before they can log in. I have also introduced input validation checks to ensure that these PINs conform to specified requirements and improve the robustness of the system.Nevertheless, there are limits on this approach:Storing user credentials locally is

exposed to third party infiltration or manipulation.The absence of other authentication factors such as biometric or two-factor authentication exposes the system to password cracking attacks.Although encrypted, it may lack the same level of protection as enterprise-class solutions do.These risks can be mitigated by regular updates and maintenance and enhancing trust from users.

Compatibility:

The compatibility of my game across an array of devices and platforms is vital. This ensures maximal reach and accessibility. Yet, due to variations in hardware specifications, operating systems, and screen resolutions, compatibility problems may emerge. Incompatibility can lead to less than ideal performance. It can also cause graphical glitches, or even inability to start the game on certain devices.Further, differences in input methods might call for more development work. For example, the use of touchscreen controls in contrast to keyboard and mouse setups require different approaches. This is imperative to achieve a consistent and intuitive gameplay experience across platforms. Neglecting compatibility issues could result in isolating a large number of potential players. It could also restrict the game's market reach.Thus, addressing compatibility issues is vital. This is key to providing a seamless gaming experience that all players can enjoy. It is also important regardless of the chosen device or platform.Moreover, the game was formulated explicitly in Python for computers. This renders it unsuitable for mobile gadgets and other platforms. Challenges might surface with fresh Python updates. Changes in operating systems could also trigger complications. Differing hardware specifications and screen resolutions among users might impact performance. Support for various input methods is limited. This could necessitate substantial development work. Addressing these issues is paramount. This is the path for guaranteeing gaming experience that is seamless for all players.

5.2 Further development

Dynamic Difficulty Levels:

Introduce dynamic difficulty adjustments based on player performance to provide a personalised and challenging gaming experience.

Justification: Improved data collection can help make the game better by showing what parts players like the most, what parts are really hard, and what parts need to be fixed. This gives a good idea of how people play and how well they do. With this information,

updates can be made that are more likely to please players and make them want to keep playing. Also, collecting a lot of data lets allows each player a customised experience, which can make them more interested and want to keep playing for longer.

Enhanced Player Data Collection:

Enhance the game's data collection mechanisms to gather comprehensive insights into player behaviours and performance.

Justification: Improved data collection informs iterative enhancements by identifying popular aspects challenging segments and areas requiring refinement this comprehensive understanding of player behaviours and performance allows for targeted updates and refinements ensuring that future iterations of the game align closely with player expectations and preferences additionally thorough data collection facilitates the implementation of personalised gaming experiences enhancing player engagement and retention

Expanded Educational Content:

Incentivized Achievement:To keep players motivated and interested, a reward system could be created where they get game currency for doing well and making progress in the game.They can then use this to buy different skins or power ups from an in game shop. This way, they'll want to keep playing and trying to beat the levels to get more rewards.

Justification:Having a reward system is a great idea, because it gives players something to work towards and makes them want to keep playing. It's like, if you're really good at a game and you keep winning, you feel proud and accomplished. That's what the reward system is for - to make players feel that way and keep them coming back for more. Plus, it just makes the game more fun and satisfying overall.

Ensuring Security Measures:

Implement robust security measures, such as two-step authentication, to safeguard player accounts and sensitive information.

To protect player accounts and sensitive information, stronger security measures should be implemented. For example, two-step authentication could be introduced to enhance account security.

Justification:The implementation of robust security measures is crucial to ensure the safety of player accounts and sensitive data. It provides reassurance to players that their information is safeguarded and that assistance is available in case of password loss or similar issues. Moreover, it underscores our commitment to prioritising player privacy and data protection. Effective security measures not only enhance player trust but also contribute to the overall credibility of the game. By instilling confidence in players, it creates a more positive gaming experience and fosters a sense of security. Additionally, robust security practices demonstrate professionalism and seriousness in game development.Considering these factors and exploring avenues for further improvement, the snake game can distinguish itself and retain player engagement over time. The aim is to strike a balance between educational value, entertainment, and social interaction, ensuring a dynamic and enjoyable gaming experience for all players.

5.3 If I could build this project again:

In this section I briefly talk about what I would like to do if i had the opportunity to build the project again

1)Changes in Design:

- *Reflection:* Looking back, I might refine the user interface to make it even more intuitive. Improving usability by incorporating child-friendly design principles, larger icons, and clearer instructions could enhance the overall user experience. Additionally, I would revisit certain aspects of the educational component, ensuring a more diverse and adaptive range of questions to cater to various learning levels.

2)Technologies:

-*Reflection:* Implementing artificial intelligence, with a focus on Python libraries such as TensorFlow or PyTorch, is pivotal for creating an adaptive learning experience in my game

-The AI-driven approach ensures a personalised and challenging learning journey for each student, aligning with the expectations of schools, tuition, and parents.

-The system continuously adapts to students' responses, offering a tailored educational experience and promoting effective learning outcomes.

3)Add ons:

Reflection: In retrospect, a significant enhancement to my game would involve the incorporation of advanced deep analytics specifically focused on user performance. Leveraging Python's robust data analysis libraries, including Pandas and NumPy, this technological approach aims to process and analyse user-specific metrics efficiently. By tracking individual question response times, accuracy rates, and overall progress, the analytics system provides comprehensive insights. Visualisation tools, potentially implemented with Matplotlib or Seaborn, offer graphical representations of performance trends. The user-friendly reports generated by this analytics dashboard, accessible to educators and parents, aim to facilitate a nuanced understanding of a student's strengths and areas needing improvement. This continuous improvement strategy ensures that the system evolves dynamically, staying aligned with emerging educational methodologies and user feedback.

Source code:

For the Main Game:

```python
#Initializiation
import pygame
import random
import sys
from pygame.math import Vector2
from pygame import mixer
import sqlite3 as sq

import sys

if "--username" in sys.argv:
    username_index = sys.argv.index("--username")
    if username_index + 1 < len(sys.argv):
        username = sys.argv[username_index + 1]
        print("Username:", username)
    else:
        print("No username provided.")
else:
    print("Username not found in command-line arguments.")


pygame.mixer.pre_init(22050, -16, 1, 1024)
pygame.init()
```

```python
pygame.font.init()
from Button import Button


# Setting up the database
import sqlite3 as sq

db = sq.connect("database.db")

try:
    st = """
        CREATE TABLE Addition_table
        (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
        """
    db.execute(st)
    db.commit()

        # Insert data only once
    insert = "INSERT INTO Addition_table(Name1, Score1,Name2, Score2,Name3, Score
    db.execute(insert)
    db.commit()
```

```
except Exception as e:
    print("Error:", e)

# Getting the info from the database
fetc = """SELECT * FROM Addition_table"""
data = db.execute(fetc)
Addition_elements = []
for row in data:
    # Appending each row as a tuple
    Addition_elements.append((row[0], row[1]))
    Addition_elements.append((row[2], row[3]))
    Addition_elements.append((row[4], row[5]))

print(Addition_elements)




try:
    st = """
        CREATE TABLE Subtract_table
        (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
        """
    db.execute(st)
    db.commit()

        # Insert data only once
    insert = "INSERT INTO Subtract_table(Name1, Score1,Name2, Score2,Name3, Score3) VALUES ('{}','{}','{}','{}','{}','{}')".format(
    db.execute(insert)
```

```
69          db.commit()
70
71              # Insert data only once
72          insert = "INSERT INTO Subtract_table(Name1, Score1,Name2, Score2,Name3, Score3) VALUES ('{}','{}','{}','{}','{}','{}')".format("
73          db.execute(insert)
74          db.commit()
75
76      except Exception as e:
77          print("Error:", e)
78
79      # Getting the info from the database
80      fetc = """SELECT * FROM Subtract_table"""
81      data = db.execute(fetc)
82      Subtract_elements = []
83      for row in data:
84          # Appending each row as a tuple
85          Subtract_elements.append((row[0], row[1]))
86          Subtract_elements.append((row[2], row[3]))
87          Subtract_elements.append((row[4], row[5]))
88
89      print(Subtract_elements)
90
91
92
93      try:
94          st = """
95              CREATE TABLE Multiply_table
96              (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
```

```
try:
    st = """
        CREATE TABLE Multiply_table
        (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
        """
    db.execute(st)
    db.commit()

        # Insert data only once
    insert = "INSERT INTO Multiply_table(Name1, Score1,Name2, Score2,Name3, Score3) VALUES ('{}','{}','{}','{}','{}','{}')".format("
    db.execute(insert)
    db.commit()

except Exception as e:
    print("Error:", e)

# Getting the info from the database
fetc = """SELECT * FROM Multiply_table"""
data = db.execute(fetc)
Multiply_elements = []
for row in data:
    # Appending each row as a tuple
    Multiply_elements.append((row[0], row[1]))
    Multiply_elements.append((row[2], row[3]))
    Multiply_elements.append((row[4], row[5]))

print(Multiply_elements)
```

```
try:
    st = """
        CREATE TABLE Divide_table
        (Name1 TEXT, Score1 TEXT,Name2 TEXT, Score2 TEXT,Name3 TEXT, Score3 TEXT)
        """
    db.execute(st)
    db.commit()

        # Insert data only once
    insert = "INSERT INTO Divide_table(Name1, Score1,Name2, Score2,Name3, Score3) VALUES ('{}','{}','{}','{}','{}','{}')".format("-"
    db.execute(insert)
    db.commit()

except Exception as e:
    print("Error:", e)

# Getting the info from the database
fetc = """SELECT * FROM Divide_table"""
data = db.execute(fetc)
Divide_elements = []
for row in data:
    # Appending each row as a tuple
    Divide_elements.append((row[0], row[1]))
    Divide_elements.append((row[2], row[3]))
    Divide_elements.append((row[4], row[5]))

print(Divide_elements)
```

```
#Background, Screen, Icons  and images
Cell_size = 22
Cell_number = 22
OFFSET = 75
SCREEN = pygame.display.set_mode(((OFFSET + OFFSET) + Cell_number * Cell_size, (OFFSET + OFFSET) + Cell_number * Cell_size))
BackGroundImage = pygame.image.load("LineBG2.jpg")
BackGroundImage = pygame.transform.scale(BackGroundImage, (SCREEN.get_width(), SCREEN.get_height()))
ICON = pygame.image.load("snake_icon.png")
pygame.display.set_icon(ICON)
pygame.display.set_caption("snake Maths")
fruit_image = pygame.image.load("apple.png")
stone_image = pygame.image.load("stone.png")
bomb_image = pygame.image.load("bomb.png")
explode_image = pygame.image.load("explode.png")

Home = pygame.image.load("home.png")
Home = pygame.transform.scale(Home, (50, 50))


Reset = pygame.image.load("reset.png")
Reset = pygame.transform.scale(Reset, (50, 50))


speaker = pygame.image.load("speaker1.png")
speaker = pygame.transform.scale(speaker, (50, 50))

speaker2 = pygame.image.load("speaker2.png")
speaker2 = pygame.transform.scale(speaker2, (50, 50))
```

```
#RGB values and other constants
ORANGE = (251,153,2)
BLUE = (0,0,255)
GREEN = (0, 255, 0)
WHITE = (255, 255, 255)
snake_UPDATE = pygame.USEREVENT
pygame.time.set_timer(snake_UPDATE, 150)

QUESTION_FONT = pygame.font.Font(None, 30)
SCORE_FONT = pygame.font.Font(None, 20)

#-----------------------------------------------------------------------------------------
#Function for growing grid
def draw_grid(screen, cell_size, cell_number, offset):
    WHITE = (255, 255, 255)
    for y in range(offset, offset + cell_size * cell_number + 1, cell_size):
        pygame.draw.line(screen, WHITE, (offset, y), (offset + cell_size * cell_number, y), 1)
    for x in range(offset, offset + cell_size * cell_number + 1, cell_size):
        pygame.draw.line(screen, WHITE, (x, offset), (x, offset + cell_size * cell_number), 1)
    pygame.draw.rect(screen, BLUE, (offset - 1, offset - 1, cell_size * cell_number + 2, cell_size * cell_number + 2), 2)

#-----------------------------------------------------------------------------------------------------------------------
```

```
#Classes

class STONE:
    def __init__(self):
        self.position = self.random()

    def random(self):
        x = random.randint(0, Cell_number - 1)
        y = random.randint(0, Cell_number - 1)
        return Vector2(x, y)

    def draw(self):
        stone_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
        stone_image_scaled = pygame.transform.scale(stone_image, (Cell_size, Cell_size))
        SCREEN.blit(stone_image_scaled, stone_rect)

    def check_collision(self, snake_head):
        return snake_head == self.position



class Bomb:
    def __init__(self):
        self.position = self.random()
        self.explode = False

    def random(self):
        x = random.randint(0, Cell_number - 1)
```

```
        y = random.randint(0, Cell_number - 1)
        return Vector2(x, y)

    def draw(self):
        if self.explode:
            Bomb_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
            Bomb_image_scaled = pygame.transform.scale(explode_image, (Cell_size, Cell_size))
        else:
            Bomb_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
            Bomb_image_scaled = pygame.transform.scale(bomb_image, (Cell_size, Cell_size))
        SCREEN.blit(Bomb_image_scaled, Bomb_rect)

    def check_collision(self, snake_head):
        if snake_head == self.position:
            self.explode = True


        return snake_head == self.position
```

```
class FRUIT:
    def __init__(self, correct_answer = None, incorrect_answer = None):
        self.position = self.random()
        self.correct_answer = correct_answer
        self.incorrect_answer = incorrect_answer
        self.is_correct = random.choice([True, False])

    def random(self):
        x = random.randint(0, Cell_number - 1)
        y = random.randint(0, Cell_number - 1)
        return Vector2(x, y)

    def generate_answers(self):
        num1 = random.randint(1, 10)
        num2 = random.randint(1, 10)
        correct_answer = num1 + num2  # For addition questions, you can change this as needed
        incorrect_answer = correct_answer + random.randint(1, 5)  # Generate an incorrect answer
        return correct_answer, incorrect_answer

    def draw(self):
        fruit_rect = pygame.Rect(OFFSET + self.position.x * Cell_size, OFFSET + self.position.y * Cell_size, Cell_size, Cell_size)
        fruit_image_scaled = pygame.transform.scale(fruit_image, (Cell_size, Cell_size))
        SCREEN.blit(fruit_image_scaled, fruit_rect)

        value_text = str(self.correct_answer) if self.is_correct else str(self.incorrect_answer)
        value_surface = SCORE_FONT.render(value_text, True, (0, 0, 0))  # Black color
        value_rect = value_surface.get_rect(center=fruit_rect.center)  # Center the text on the fruit
        SCREEN.blit(value_surface, value_rect)
```

```
class Snake:
    def __init__(self):
        self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
        self.direction = Vector2(1,0)
        self.add_part = False
        self.chewing_sound = pygame.mixer.Sound("Sound/eat.mp3")
        self.mistake_sound = pygame.mixer.Sound("Sound/wall.mp3")
        self.growth_count = 0  # Keep track of how many parts the snake has grown


    def draw(self):
        for part in self.body:
            parts_rect = (OFFSET + part.x * Cell_size, OFFSET + part.y * Cell_size, Cell_size, Cell_size)
            pygame.draw.rect(SCREEN, ORANGE, parts_rect, 0, 7)

    def movement(self):
        self.body.insert(0,self.body[0] + self.direction)
        if self.add_part and self.growth_count < 14: #(For some reaoson when you set the growth count to 10 it only grows 8 parts)
            self.growth_count += 1
        else:
            # If the snake has reached the maximum growth, remove the last part
            self.body = self.body[:-1]
        self.add_part = False


    def grow(self):
        # Check if the snake can grow (up to a maximum of 10 parts)
        if self.growth_count < 14: #(For some reaoson when you set the growth count to 10 it only grows 8 parts)
```

```
286 class Snake:
317     def reset_position(self):
318         self.body = [Vector2(6,9) , Vector2(5,9), Vector2(4,9)]
319         self.direction = Vector2(1,0)
320         self.growth_count = 0
321
322 class GAME:
323     initial_timer_duration = 10000  # Initial timer duration (10 seconds)
324     def __init__(self,mode):
325         self.SCREEN = SCREEN
326         self.fruit = FRUIT()
327         self.fruit2 = FRUIT()
328         self.snake = Snake()
329         self.stone = STONE()
330         self.mode = mode
331         self.bomb = Bomb()
332         self.name = username
333         self.get_name = False
334         self.home_rect = pygame.Rect(180,400,50,50)
335         self.play_again_rect = pygame.Rect(280,400,50,50)
336         self.mute_rect = pygame.Rect(380,400,50,50)
337
338
339
340
341         self.SCREEN = SCREEN
342         self.state = "ready"
343         self.score = 0
344         self.playing_music = False
```

```
335         self.play_again_rect = pygame.Rect(280,400,50,50)
336         self.mute_rect = pygame.Rect(380,400,50,50)
337
338
339
340
341         self.SCREEN = SCREEN
342         self.state = "ready"
343         self.score = 0
344         self.playing_music = False
345         self.play_background_music()
346
347         # Timer attributes
348         self.initial_timer_duration = 11000  # Initial timer duration (10 seconds)
349         self.time_left = self.initial_timer_duration  # Initial time for answer selection
350         self.timer_start_time = pygame.time.get_ticks()
351         self.timer_duration = self.initial_timer_duration  # Actual timer duration
352         self.start_time = pygame.time.get_ticks()  # Record the start time when the game starts
353
354         #Maths attributes
355
356         self.correct_answer = 0
357         self.incorrect_answer = 0
358         self.current_question = ""
359         self.selected_operation = "addition"
360         self.generate_question()
```

```
364     def draw_timer(self):
365         time_remaining = self.update_timers()
366         timer_text = f"Time left: {time_remaining // 1000} s"
367         timer_surface = SCORE_FONT.render(timer_text, True, (255, 0, 0))
368         timer_rect = timer_surface.get_rect(topright=(self.SCREEN.get_width() - 10, 20))
369         self.SCREEN.blit(timer_surface, timer_rect)
370 
371     def update_timers(self):
372         # Calculate the time elapsed since the timer started
373         current_time = pygame.time.get_ticks()
374         elapsed_time = current_time - self.timer_start_time
375 
376         # Calculate the time remaining
377         time_remaining = max(0, self.timer_duration - elapsed_time)
378 
379         # Handle decreasing timer duration based on score
380         if self.score >= 10:
381             self.timer_duration = 9000  # Decrease by 1 second
382         elif self.score >= 20:
383             self.timer_duration = 8000
384         elif self.score >= 230:
385             self.timer_duration = 7000
```

```
384         elif self.score >= 230:
385             self.timer_duration = 7000
386         elif self.score >= 40:
387             self.timer_duration = 6000
388         elif self.score >= 50:
389             self.timer_duration = 5000  # Set to 5 seconds
390 
391         # Check if time is up
392         if time_remaining == 0:
393             self.game_over()  # Time's up, end the game
394 
395         return time_remaining
396 
397     def reset_timer(self):
398         self.timer_start_time = pygame.time.get_ticks()
399         self.timer_duration = self.initial_timer_duration
400 
401     def calculate_time_survived(self):
402         current_time = pygame.time.get_ticks()  # Get the current time
403         time_survived = current_time - self.start_time  # Calculate time survived
404         return time_survived
405 
406     def draw_time_survived(self):
407         time_survived = self.calculate_time_survived()
408         time_text = f"Time Survived: {time_survived // 1000} s"
409         time_surface = SCORE_FONT.render(time_text, True, (255, 0, 0))
410         time_rect = time_surface.get_rect(topright=(self.SCREEN.get_width() - 10, 50))
```

```
414     # Maths Logic and functionality
415     def generate_question(self):
416
417         num1 = random.randint(1, 10)
418         num2 = random.randint(1, 10)
419
420         self.incorrect_answer = self.generate_incorrect_answer(self.correct_answer)
421         if self.mode != "":
422             if self.mode =='addition':
423                 self.correct_answer = num1 + num2
424             if self.mode =='subtract':
425                 self.correct_answer = num1 - num2
426
427             if self.mode =='multiply':
428                 self.correct_answer = num1 * num2
429
430             if self.mode =='divide':
431                 while num1 < num2 or num1 == 0 or num2 == 0 or num1 % num2 != 0:
432                     num1 = random.randint(1, 40)
433                     num2 = random.randint(1, 10)
434                 self.correct_answer = num1 // num2
435
436
437             # Reset timer duration to initial value
438             self.timer_duration = self.initial_timer_duration
439
440             # Generate the first incorrect answer
441             while True:
```

```
            while True:
                self.incorrect_answer = self.generate_incorrect_answer(self.correct_answer)
                if self.incorrect_answer is not None and self.incorrect_answer != self.correct_answer:
                    break

            # Generate the second correct and incorrect answers
            while True:
                num1_2 = random.randint(1, 10)
                num2_2 = random.randint(1, 10)
                if self.mode =='addition':
                    self.correct_answer2 = num1_2 + num2_2
                if self.mode =='subtract':
                    self.correct_answer2 = num1_2 - num2_2

                if self.mode =='multiply':
                    self.correct_answer2 = num1_2 * num2_2

                if self.mode =='divide':
                    while num1_2 < num2_2  or num1_2  == 0 or num2_2  == 0 or num1_2  % num2_2  != 0:
                        num1_2  = random.randint(1, 40)
                        num2_2  = random.randint(1, 10)
                    self.correct_answer2 = num1_2 // num2_2
```

```
            self.incorrect_answer2 = self.generate_incorrect_answer(self.correct_answer2)

            if self.correct_answer2 != self.correct_answer and self.incorrect_answer2 != self.correct_answer:
                break

        # Choose randomly which fruit will have the correct answer
        correct_fruit = random.choice([self.fruit, self.fruit2])
        correct_fruit.correct_answer = self.correct_answer
        correct_fruit.incorrect_answer = self.incorrect_answer
        correct_fruit.is_correct = True

        incorrect_fruit = self.fruit2 if correct_fruit is self.fruit else self.fruit
        incorrect_fruit.correct_answer = self.correct_answer2
        incorrect_fruit.incorrect_answer = self.incorrect_answer2
        incorrect_fruit.is_correct = False

        # Generate the current question string
        if self.mode =='addition':
            self.current_question = f"{num1} + {num2}"
        if self.mode =='subtract':
            self.current_question = f"{num1} - {num2}"

        if self.mode =='multiply':
            self.current_question = f"{num1} * {num2}"
```

```
            if self.mode =='divide':
                self.current_question = f"{num1} / {num2}"


            self.timer_start_time = pygame.time.get_ticks()


    def generate_incorrect_answer(self, correct_answer, other_incorrect_answer=None):
    # Generate an incorrect answer that is not equal to the correct answer
        while True:
            incorrect_answer = random.randint(1, 20)  # Adjust the range as needed
            if incorrect_answer != correct_answer and incorrect_answer != other_incorrect_answer:
                return incorrect_answer

    def load_background_music(self):
        pygame.mixer.music.load("Sound/bg_music_1.mp3")


    def play_background_music(self):

        if not self.playing_music:
            pygame.mixer.music.load("Sound/bg_music_1.mp3")  # Load the music file
            pygame.mixer.music.play(-1)  # Start playing
            self.playing_music = True
```

```
def draw(self):
    self.snake.draw()
    self.fruit.draw()

    # Display the current question using an f-string
    question_text = f"Question: {self.current_question}"
    question_surface = QUESTION_FONT.render(question_text, True, (255, 0, 0))
    question_rect = question_surface.get_rect(topleft=(OFFSET + 10, 20))
    self.SCREEN.blit(question_surface, question_rect)

    question_surface = QUESTION_FONT.render('score: '+str(self.score), True, (255,0, 0))

    self.SCREEN.blit(question_surface,(500,600))


def update(self):
    if self.state == "ready":
        self.play_background_music()
        self.snake.movement()
        self.collision_with_fruit()
        self.boundary_check()
        self.collision_with_body()
        self.draw_time_survived()  # Draw the "Time Survived" on the screen
        self.collision_with_stone() # Handles the any collisions with the stone
```

```
  def collision_with_fruit(self):
      if self.snake.body[0] == self.fruit.position:
          if self.fruit.is_correct:  # Check if the fruit contains the correct answer
              self.snake.grow()
              self.score += 1
              self.fruit.position = self.fruit.random()  # Generate a new random position for the first fruit
              self.fruit2.position = self.fruit2.random()  # Generate a new random position for the second fruit
              self.generate_question()
              self.reset_timer()

          else:
              self.game_over()  # snake hit the incorrect answer
          self.fruit.position = self.fruit.random()  # Generate a new fruit position

      elif self.snake.body[0] == self.fruit2.position:
          if self.fruit2.is_correct:  # Check if the fruit2 contains the correct answer
              self.snake.grow()
              self.score += 1
              self.fruit.position = self.fruit.random()  # Generate a new random position for the first fruit
              self.fruit2.position = self.fruit2.random()  # Generate a new random position for the second fruit
              self.generate_question()
              self.reset_timer()
              # Respawn the stone randomly when the score goes up
              self.stone.position = self.stone.random()
              self.bomb.position = self.bomb.random()
          else:
              self.game_over()  # snake hit the incorrect answer
```

```
574     def boundary_check(self):
575         if self.snake.body[0].x == Cell_number or self.snake.body[0].x == -1:
576             self.game_over()
577             print("Boundary Check - Timer Start Time:", self.timer_start_time)
578             print("Boundary Check - Time Left:", self.time_left)
579             self.reset_timer()
580 
581         if self.snake.body[0].y == Cell_number or self.snake.body[0].y == -1:
582             self.game_over()
583             print("Boundary Check - Timer Start Time:", self.timer_start_time)
584             print("Boundary Check - Time Left:", self.time_left)
585             self.reset_timer()  # Reset timer when snake hits boundaries
586 
587 
588     def collision_with_body(self):
589         for part in self.snake.body[1:]:
590             if self.snake.body[0] == part:
591                 self.reset_timer()  # Reset timer when snake hits itself
592                 self.game_over()
593                 print("Collision with Body - Timer Start Time:", self.timer_start_time)
594                 print("Collision with Body - Time Left:", self.time_left)
595 
596     def collision_with_stone(self):
597         if self.snake.body[0] == self.stone.position or self.snake.body[0] == self.bomb.position:
598             self.bomb.explode = True
599             self.bomb.draw()
600 
```

```
597         if self.snake.body[0] == self.stone.position or self.snake.body[0] == self.bomb.position:
598             self.bomb.explode = True
599             self.bomb.draw()
600 
601 
602             self.game_over()  # snake collided with the stone
603             self.reset_timer()
604             self.stone.position = self.stone.random()
605             self.bomb.position = self.bomb.random()
606 
607 
608 
609 
610 
611     def reset_game(self):
612         # Reset game attributes to their initial state
613         self.fruit = FRUIT()
614         self.snake = Snake()
615         self.get_name = False
616         self.state = "ready"
617         self.play_background_music()
618         self.bomb.explode = False
619         # self.name= ''
620         self.score = 0
621         self.stone.position = self.stone.random()
622         self.bomb.position = self.bomb.random()
```

```
624             # Generate new question values for both fruits
625             self.generate_question()
626             self.fruit2.generate_answers()  # Generate answers for the second fruit
627             self.reset_timer()  # Reset the timer
628 
629         def draw_score(self):
630             score_text = f"Score: {self.score}"
631             score_surface = SCORE_FONT.render(score_text, True, (255, 0, 0))
632             score_rect = score_surface.get_rect(bottomleft=(OFFSET + 10, self.SCREEN.get_height() - 10))  # Display at the bottom lef
633             self.SCREEN.blit(score_surface, score_rect)
634 
635         def game_over(self):
636             pygame.mixer.music.stop()  # Stop background music
637             self.snake.reset_position()
638             self.fruit.random()
639             self.start_time = pygame.time.get_ticks()  # Reset the start time when the game is over
640             self.state = "not ready"
641             final_score = self.score
642             self.playing_music = False
643 
644 
645             print("Final Score:", self.score)
646 
647             font = pygame.font.Font(None, 48)
648             text = font.render(f"Final Score: {self.score} ", True, (0,0,0))
649             text_rect = text.get_rect(center=(self.SCREEN.get_width() // 2, self.SCREEN.get_height() // 2))
```

```
650             start_text = font.render("Better luck next time", True, (0,0,0))
651             start_text_rect = start_text.get_rect(center=(self.SCREEN.get_width() // 2, self.SCREEN.get_height() // 2 + 50))
652             self.SCREEN.blit(text, text_rect)
653             self.SCREEN.blit(start_text, start_text_rect)
654             SCREEN.blit(Home,(self.home_rect.x,self.home_rect.y))
655 
656 
657             # pygame.draw.rect(SCREEN,'red',self.play_again_rect)
658             SCREEN.blit(Reset,(self.play_again_rect.x,self.play_again_rect.y))
659             if self.mode == 'addition':
660                 if self.score> int(Addition_elements[0][1]) or self.score> int(Addition_elements[1][1]) or self.score> int(Addition_eleme
661                     self.get_name = True
662 
663             if self.mode == 'subtract':
664                 if self.score> int(Subtract_elements[0][1]) or self.score> int(Subtract_elements[1][1]) or self.score> int(Subtract_eleme
665                     self.get_name = True
666 
667 
668             if self.mode == 'multiply':
669                 if self.score> int(Multiply_elements[0][1]) or self.score> int(Multiply_elements[1][1]) or self.score> int(Multiply_eleme
670                     self.get_name = True
671 
672 
673             if self.mode == 'divide':
674                 if self.score> int(Divide_elements[0][1]) or self.score> int(Divide_elements[1][1]) or self.score> int(Divide_elements[2]
675                     self.get_name = True
```

```
680         pygame.display.update()
681
682         self.reset_timer()  # Reset timer when the game is over
683
684         waiting_for_input = True
685         while waiting_for_input:
686             if self.playing_music :
687
688                 SCREEN.blit(speaker2,(self.mute_rect.x,self.mute_rect.y))
689             else:
690                 SCREEN.blit(speaker,(self.mute_rect.x,self.mute_rect.y))
691             MOUSE_POS = pygame.mouse.get_pos()
692
693
694
695             for event in pygame.event.get():
696                 if event.type == pygame.QUIT:
697                     pygame.quit()
698                     sys.exit()
699
700                 if event.type == pygame.MOUSEBUTTONDOWN:
701                     if self.mute_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]):
702                         if self.playing_music:
703                             self.playing_music = False
704
705                         else:
706                             self.playing_music = True
```

```
710                 if self.play_again_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]) or self.home_rect.collidepoint(MOUSE_POS[0],MOUS
711                     if True:
712
713                         if self.mode == 'addition':
714                             if self.score> int(Addition_elements[0][1]):
715                                 Insert=f'''UPDATE Addition_table SET Score1={self.score}'''
716                                 Insert2 = f'''UPDATE Addition_table SET Name1 = '{self.name}' '''
717
718                             elif self.score> int(Addition_elements[1][1]):
719                                 Insert=f'''UPDATE Addition_table SET Score2={self.score}'''
720                                 Insert2 = f'''UPDATE Addition_table SET Name2 = '{self.name}' '''
721
722                             elif self.score> int(Addition_elements[2][1]):
723                                 Insert=f'''UPDATE Addition_table SET Score3={self.score}'''
724                                 Insert2 = f'''UPDATE Addition_table SET Name3 = '{self.name}' '''
725
726
727
728                         if self.mode == 'subtract':
729                             if self.score> int(Subtract_elements[0][1]):
730                                 Insert=f'''UPDATE Subtract_table SET Score1={self.score}'''
731                                 Insert2 = f'''UPDATE Subtract_table SET Name1 = '{self.name}' '''
732
733                             elif self.score> int(Subtract_elements[1][1]):
734                                 Insert=f'''UPDATE Subtract_table SET Score2={self.score}'''
735                                 Insert2 = f'''UPDATE Subtract_table SET Name2 = '{self.name}' '''
```

```
736
737                         elif self.score> int(Subtract_elements[2][1]):
738                             Insert=f'''UPDATE Subtract_table SET Score3={self.score}'''
739                             Insert2 = f'''UPDATE Subtract_table SET Name3 = '{self.name}' '''
740
741
742
743
744                     if self.mode == 'multiply':
745                         if self.score> int(Multiply_elements[0][1]):
746                             Insert=f'''UPDATE Multiply_table SET Score1={self.score}'''
747                             Insert2 = f'''UPDATE Multiply_table SET Name1 = '{self.name}' '''
748
749                         elif self.score> int(Multiply_elements[1][1]):
750                             Insert=f'''UPDATE Multiply_table SET Score2={self.score}'''
751                             Insert2 = f'''UPDATE Multiply_table SET Name2 = '{self.name}' '''
752
753                         elif self.score> int(Multiply_elements[2][1]):
754                             Insert=f'''UPDATE Multiply_table SET Score3={self.score}'''
755                             Insert2 = f'''UPDATE Multiply_table SET Name3 = '{self.name}' '''
756
757
758
759
760
761                     if self.mode == 'divide':
762                         if self.score> int(Divide_elements[0][1]):
```

```
761                     if self.mode == 'divide':
762                         if self.score> int(Divide_elements[0][1]):
763                             Insert=f'''UPDATE Divide_table SET Score1={self.score}'''
764                             Insert2 = f'''UPDATE Divide_table SET Name1 = '{self.name}' '''
765
766                         elif self.score> int(Divide_elements[1][1]):
767                             Insert=f'''UPDATE Divide_table SET Score2={self.score}'''
768                             Insert2 = f'''UPDATE Divide_table SET Name2 = '{self.name}' '''
769
770                         elif self.score> int(Divide_elements[2][1]):
771                             Insert=f'''UPDATE Divide_table SET Score3={self.score}'''
772                             Insert2 = f'''UPDATE Divide_table SET Name3 = '{self.name}' '''
773                     try:
774                         db.execute(Insert)
775                         db.commit()
776                         db.execute(Insert2)
777                         db.commit()
778                     except:
779                         pass
780
781                 self.reset_game()  # Reset the game state
782                 waiting_for_input = False
783             if self.home_rect.collidepoint(MOUSE_POS[0],MOUSE_POS[1]):
784                 menu()
```

```
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN: # Restart the game on spacebar press
                        if True:
                            if self.mode == 'addition':
                                if self.score> int(Addition_elements[0][1]):
                                    Insert=f'''UPDATE Addition_table SET Score1={self.score}'''
                                    Insert2 = f'''UPDATE Addition_table SET Name1 = '{self.name}' '''

                                elif self.score> int(Addition_elements[1][1]):
                                    Insert=f'''UPDATE Addition_table SET Score2={self.score}'''
                                    Insert2 = f'''UPDATE Addition_table SET Name2 = '{self.name}' '''

                                elif self.score> int(Addition_elements[2][1]):
                                    Insert=f'''UPDATE Addition_table SET Score3={self.score}'''
                                    Insert2 = f'''UPDATE Addition_table SET Name3 = '{self.name}' '''



                            if self.mode == 'subtract':
                                if self.score> int(Subtract_elements[0][1]):
                                    Insert=f'''UPDATE Subtract_table SET Score1={self.score}'''
                                    Insert2 = f'''UPDATE Subtract_table SET Name1 = '{self.name}' '''

                                elif self.score> int(Subtract_elements[1][1]):
                                    Insert=f'''UPDATE Subtract_table SET Score2={self.score}'''
                                    Insert2 = f'''UPDATE Subtract_table SET Name2 = '{self.name}' '''
```

```
                                elif self.score> int(Subtract_elements[2][1]):
                                    Insert=f'''UPDATE Subtract_table SET Score3={self.score}'''
                                    Insert2 = f'''UPDATE Subtract_table SET Name3 = '{self.name}' '''



                            if self.mode == 'multiply':
                                if self.score> int(Multiply_elements[0][1]):
                                    Insert=f'''UPDATE Multiply_table SET Score1={self.score}'''
                                    Insert2 = f'''UPDATE Multiply_table SET Name1 = '{self.name}' '''

                                elif self.score> int(Multiply_elements[1][1]):
                                    Insert=f'''UPDATE Multiply_table SET Score2={self.score}'''
                                    Insert2 = f'''UPDATE Multiply_table SET Name2 = '{self.name}' '''

                                elif self.score> int(Multiply_elements[2][1]):
                                    Insert=f'''UPDATE Multiply_table SET Score3={self.score}'''
                                    Insert2 = f'''UPDATE Multiply_table SET Name3 = '{self.name}' '''



                            if self.mode == 'divide':
                                if self.score> int(Divide_elements[0][1]):
                                    Insert=f'''UPDATE Divide_table SET Score1={self.score}'''
```

```
                                    Insert2 = f'''UPDATE Divide_table SET Name1 = '{self.name}' '''

                                elif self.score> int(Divide_elements[1][1]):
                                    Insert=f'''UPDATE Divide_table SET Score2={self.score}'''
                                    Insert2 = f'''UPDATE Divide_table SET Name2 = '{self.name}' '''

                                elif self.score> int(Divide_elements[2][1]):
                                    Insert=f'''UPDATE Divide_table SET Score3={self.score}'''
                                    Insert2 = f'''UPDATE Divide_table SET Name3 = '{self.name}' '''
                            try:
                                db.execute(Insert)
                                db.commit()
                                db.execute(Insert2)
                                db.commit()
                            except:
                                pass

                        self.reset_game()  # Reset the game state
                        waiting_for_input = False

                    if event.key == pygame.K_BACKSPACE:
                        self.name = self.name[:-1]

                    else:
                        self.name+=event.unicode
                        print(self.name)
```

```
# Create a function to display rows of text
def draw_rows():
    row_height = 33
    y = 100  # Adjust the starting position as needed
    pygame.draw.rect(SCREEN,'black',(SCREEN.get_width()// 2,100,5,100))

    pygame.draw.rect(SCREEN,'black',(SCREEN.get_width()// 2,235,5,100))

    pygame.draw.rect(SCREEN,'black',(SCREEN.get_width()// 2,367,5,100))

    pygame.draw.rect(SCREEN,'black',(SCREEN.get_width()// 2,497,5,100))
    for i in range(16):
        pygame.draw.rect(SCREEN,'black',(0,y,SCREEN.get_width(),5))
        if i<3:
            text = QUESTION_FONT.render(Addition_elements[i][0]+ '                    '+Addition_elements[i][1], True, 'red')
            text_rect = text.get_rect()
            text_rect.center = (SCREEN.get_width()// 2, y+25)  # Center the text horizontally in each row
            SCREEN.blit(text, text_rect)



            text = QUESTION_FONT.render(Subtract_elements[i][0]+ '                    '+Subtract_elements[i][1], True, 'red')
            text_rect = text.get_rect()
            text_rect.center = (SCREEN.get_width()// 2, y+155)  # Center the text horizontally in each row
            SCREEN.blit(text, text_rect)


            text = QUESTION_FONT.render(Multiply_elements[i][0]+ '                    '+Multiply_elements[i][1], True, 'red')
```

```
            text_rect = text.get_rect()
            text_rect.center = (SCREEN.get_width()// 2, y+285)  # Center the text horizontally in each row
            SCREEN.blit(text, text_rect)


            text = QUESTION_FONT.render(Divide_elements[i][0]+ '                        '+Divide_elements[i][1], True, 'red')
            text_rect = text.get_rect()
            text_rect.center = (SCREEN.get_width()// 2, y+415)  # Center the text horizontally in each row
            SCREEN.blit(text, text_rect)

        y += row_height

    text = QUESTION_FONT.render("Addition Highscores", True, 'red')
    text_rect = text.get_rect()
    text_rect.center = (SCREEN.get_width()// 2, 90)  # Center the text horizontally in each row
    SCREEN.blit(text, text_rect)

    text = QUESTION_FONT.render("Subtraction Highscores", True, 'red')
    text_rect = text.get_rect()
    text_rect.center = (SCREEN.get_width()// 2, 220)  # Center the text horizontally in each row
    SCREEN.blit(text, text_rect)

    text = QUESTION_FONT.render("Multiplication Highscores", True, 'red')
    text_rect = text.get_rect()
    text_rect.center = (SCREEN.get_width()// 2, 350)  # Center the text horizontally in each row
    SCREEN.blit(text, text_rect)

    text = QUESTION_FONT.render("Division Highscores", True, 'red')
```

```
    text_rect = text.get_rect()
    text_rect.center = (SCREEN.get_width()// 2, 480)  # Center the text horizontally in each row
    SCREEN.blit(text, text_rect)

# Main loop
def leader_score():
    while True:
        MOUSE_POS = pygame.mouse.get_pos()

        SCREEN.fill((255, 255, 255))  # Use a tuple for the color

        # Draw rows
        draw_rows()

        # Create a "Back" button
        Back = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-190,50)), pos=(80,40)
                      text_input="Back", font=SCORE_FONT, base_color=(0, 0, 0), hovering_color=(0, 255, 0))

        for button in [Back]:
            button.changeColor(MOUSE_POS)
            button.update(SCREEN)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if Back.checkForInput(MOUSE_POS):
```

```
#-------------------------------------------------------------------------------------------
#Game Loop
def menu():
    img = pygame.transform.scale(pygame.image.load('math_symbol.png'),(120,120))
    while True:
        SCREEN.fill('white')
        SCREEN.blit(img,(50,30))
        pygame.draw.rect(SCREEN,'black',(0,600,1000,5))
        pygame.draw.rect(SCREEN,'grey',(SCREEN.get_width()/2+10,190,280,335))
        MOUSE_POS = pygame.mouse.get_pos()
        Addition = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCR
                          text_input="Addition", font=SCORE_FONT, base_color="black", hovering_color="green")

        Subtraction = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(
                          text_input="Subtraction", font=SCORE_FONT, base_color="black", hovering_color="green")

        Multiply = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCR
                          text_input="Multiplication", font=SCORE_FONT, base_color="black", hovering_color="green")

        Divide = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(SCREE
                          text_input="Division", font=SCORE_FONT, base_color="black", hovering_color="green")

        Leaderboard = Button(image=pygame.transform.scale(pygame.image.load("Options Rect2.png"),(SCREEN.get_width()/2-80,70)), pos=(
                          text_input="Leaderboard", font=SCORE_FONT, base_color="black", hovering_color="green")


        # Showing buttons
        for button in [Addition,Subtraction,Multiply,Divide,Leaderboard]:
            button.changeColor(MOUSE_POS)
```

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                if Addition.checkForInput(MOUSE_POS):
                    run_game('addition')

                if Subtraction.checkForInput(MOUSE_POS):
                    run_game('subtract')

                if Multiply.checkForInput(MOUSE_POS):
                    run_game('multiply')

                if Divide.checkForInput(MOUSE_POS):
                    run_game('divide')

                if Leaderboard.checkForInput(MOUSE_POS):
                    leader_score()


        pygame.display.update()


def run_game(mode):

    game = GAME(mode)
    FPS = 60
```

```
    RUN = True

    while RUN:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                RUN = False
                sys.exit()
            if event.type == snake_UPDATE:
                game.update()
            if event.type == pygame.KEYDOWN:
                if game.state == "not ready" and event.key == pygame.K_SPACE:
                    game.state = "ready"  # Change the game state to "ready"
                    game.reset_game()
                    waiting_for_input = False
                elif game.state == "ready":
                    if event.key == pygame.K_UP or event.key == pygame.K_w:
                        if game.snake.direction != Vector2(0, 1):
                            game.snake.direction = Vector2(0, -1)
                    elif event.key == pygame.K_DOWN or event.key == pygame.K_s:
                        if game.snake.direction != Vector2(0, -1):
                            game.snake.direction = Vector2(0, 1)
                    elif event.key == pygame.K_LEFT or event.key == pygame.K_a:
                        if game.snake.direction != Vector2(1, 0):
                            game.snake.direction = Vector2(-1, 0)
                    elif event.key == pygame.K_RIGHT or event.key == pygame.K_d:
                        if game.snake.direction != Vector2(-1, 0):
                            game.snake.direction = Vector2(1, 0)
```

```
        SCREEN.blit(BackGroundImage, (0, 0))
        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5, Cell_size * Cell_number + 10, Cell_size * Cell_number + 10), 5)
        game.snake.draw()
        game.fruit.draw()
        game.fruit2.draw()
        game.stone.draw()
        game.bomb.draw()
        game.draw()
        game.draw_timer()
        game.draw_time_survived()
        pygame.display.update()
        pygame.draw.rect(SCREEN, BLUE, (OFFSET - 5, OFFSET - 5, Cell_size * Cell_number + 10, Cell_size * Cell_number + 10), 5)

    pygame.quit()

if __name__ == '__main__':
    menu()
```

LogInSystem code:

```
Click to add a breakpoint.
from tkinter import *
from PIL import Image
from tkinter import messagebox
import ast
from pygame.locals import *
from subprocess import call



#File conversions:
input_png = 'LogInIcon.png'
output_ico = 'LogInIcon.ico'
input_jpg = 'RegisterImage.jpeg'
output_ico2 = 'RegisterImage.ico'

# Open the PNG file
Icon_img = Image.open(input_png)

# Save the icon as an ICO file
Icon_img.save(output_ico)

input_png = 'LogInIcon.png'
output_ico = 'LogInIcon.ico'

# Open the PNG file
Icon_img = Image.open(input_png)
```

```
# Save it as an ICO file
Icon_img.save(output_ico)


window = Tk()
window.title('Login')
icon_path = 'LogInIcon.ico'
window.iconbitmap(default=icon_path)
window.geometry('925x500+300+200')
window.configure(bg="#606060")
window.resizable(False,False)


def enter_digit(digit):
    current_pin = pin_entry.get()
    if len(current_pin) < 4:
        pin_entry.insert(END, str(digit))

def clear_pin():
    pin_entry.delete(0, END)

def check_pin():
    entered_pin = pin_entry.get()
    PIN = '1234'

    # Condition 1: Check if entered pin has exactly 4 digits
    if not entered_pin.isdigit() or len(entered_pin) != 4:
        messagebox.showerror('Invalid PIN', 'PIN must be exactly 4 digits')
        return
```

```
    # Condition 2: Check if entered pin matches the correct PIN
    if entered_pin == PIN:
        result_label.config(text="PIN Accepted")
        window2.destroy()
        set_up()  # Call your set up function
    else:
        messagebox.showerror('Invalid PIN', 'Incorrect PIN')\

def create_pin_entry_window():
    global window2
    # Create the window
    window2 = Tk()
    window2.title("PIN Entry")

    window2.geometry(f"{200}x{300}+{700}+{400}")
    window2.configure(bg="#2f2f30")

    # Create and place widgets
    pin_label = Label(window2, text="Enter 4-digit PIN:", bg="#2f2f30", fg="white")
    pin_label.pack(pady=10)

    global pin_entry
    pin_entry = Entry(window2, show="*", bg="white", fg="black", justify="center", font=("Helvetica", 16))
    pin_entry.pack(pady=10)

    # Create number buttons
    button_frame = Frame(window2, bg="#2f2f30")
    for i in range(1, 10):
        button = Button(button_frame, text=str(i), command=lambda i=i: enter_digit(i), width=4, height=2,bg='orange')
        button.grid(row=(i-1)//3, column=(i-1)%3, padx=5, pady=5)

    zero_button = Button(button_frame, text="0", command=lambda: enter_digit(0), width=4, height=2,bg='orange')
    zero_button.grid(row=3, column=1, padx=5, pady=5)

    clear_button = Button(button_frame, text="Clear", command=clear_pin, width=4, height=2,bg='orange')
    clear_button.grid(row=3, column=0, padx=5, pady=5)

    submit_button = Button(button_frame, text="OK", bg='orange', command=check_pin,width=4, height=2)
    submit_button.grid(row=3, column=2, padx=5, pady=5)

    button_frame.pack()




    global result_label
    result_label = Label(window2, text="", bg="#2f2f30", fg="white")
    result_label.pack(pady=10)

    # Run the main loop
    window2.mainloop()


def login():
    username=user.get()
    password=code.get()
```

```
    file=open('datasheet.txt','r')
    d=file.read()
    r=ast.literal_eval(d)
    file.close()
    try:
        file = open('datasheet.txt', 'r')
        data = file.read()
        user_data = ast.literal_eval(data)
        file.close()

        if username in user_data and password == user_data[username]:
            # Close the login window
            window.destroy()
        else:
            messagebox.showerror('Invalid', 'Invalid username or password')
    except Exception as e:
        messagebox.showerror('Error', 'An error occurred: ' + str(e))


    if username in r.keys() and password == r[username]:

        try:

            window.destroy()
        except:
            pass
```

```
          try:

              window2.destroy()
          except:
              pass



          from subprocess import call
          call(["python", "main 2.py", "--username", username])

      else:
          messagebox.showerror('Invalid','invalid username or password')
  def on_login_button_pressed():
      # window.destroy()
      pass

  def set_up():
      global window
      window = Toplevel(window)
      window.title("Register")
      window.geometry('925x500+300+200')
      window.configure(bg='#606060')
      window.resizable(False,False)
```

```
170     def register():
171         username=user.get()
172         password=code.get().strip()
173         conform_password=confirm_code.get().strip()
174         # Condition 1: Username must not be empty
175         if not username:
176             messagebox.showerror('Invalid', 'Username cannot be empty')
177             return
178          # Condition 2: Username must not contain special characters
179         if not username.isalnum():
180             messagebox.showerror('Invalid', 'Username must only contain alphanumeric characters')
181             return
182          # Condition 3: Password must be at least 6 characters long
183         if len(password) < 6:
184             messagebox.showerror('Invalid', 'Password must be at least 6 characters long')
185             return
186         # Condition 4: Password must contain an uppercase letter
187         if not any(char.isupper() for char in password):
188             messagebox.showerror('Invalid', 'Password must contain at least one uppercase letter')
189             return
190         # Condition 5: Password must contain a number
191         if not any(char.isdigit() for char in password):
192             messagebox.showerror('Invalid', 'Password must contain at least one number')
193             return
```

```
196             if password == conform_password:
197                 try:
198                     file=open('datasheet.txt','r+')
199                     d=file.read()
200                     r=ast.literal_eval(d)
201 
202                     dict2={username:password}
203                     r.update(dict2)
204                     file.truncate(0)
205                     file.close()
206 
207                     file=open('datasheet.txt','w')
208                     w=file.write(str(r))
209 
210                     messagebox.showinfo('Register','Successefully registered')
211 
212                 except:
213                     file=open('datasheet.txt','w')
214                     pp=str({'username':'password'})
215                     file.close()
216             else:
217                 messagebox.showerror('Invalid', 'Both Passwords have to match')
```

```
    window.title("Register")
    window.geometry('925x500+300+200')
    icon_path = 'LogInIcon.ico'
    window.iconbitmap(default=icon_path)
    window.configure(bg='#606060')
    window.resizable(False,False)


    img = PhotoImage(file = 'RegisterImage.png')
    img = img.subsample(4)
    Label(window,image=img,border=0,bg='#606060').place(x=50,y=90)

    frame = Frame(window, width=350, height = 390,bg='#313131')
    frame.place(x=480,y=50)

    heading = Label(frame,text='Register', fg='orange',bg = '#2f2f30', font=('Mincrosoft Yahei UI Light', 23,'bold'))
    heading.place(x=100,y=5)

    def accessing(e):
        user.delete(0,'end')
        user.config(fg='orange')

    def departing(e):
        if user.get()=='':
            user.insert(0, 'username')
            user.config(fg='orange')

    user = Entry(frame,width=25,fg='orange',border=2,bg='#2f2f30',font=('Mincrosoft Yahei UI Light', 11))
```

```
     user.place(x=30,y=90)
     user.insert(0,'username')
     user.bind("<FocusIn>",accessing)
     user.bind("<FocusOut>",departing)

     Frame(frame,width=295,height=2,bg='black').place(x=25,y=177)

     def accessing(e):
         code.delete(0,'end')
         code.config(fg='orange')


     def departing(e):
         if code.get()=='':
             code.insert(0, 'password')
             code.config(fg='orange')



     code = Entry(frame,width=25,fg='orange',border=2,bg='#2f2f30',font=('Mincrosoft Yahei UI Light', 11))
     code.place(x=30,y=150)
     code.insert(0,'password')
     code.bind("<FocusIn>",accessing)
     code.bind("<FocusOut>",departing)

     Frame(frame,width=295,height=2,bg='black').place(x=25,y=110)
```

```
    def accessing(e):
        confirm_code.delete(0,'end')
        confirm_code.config(fg='orange')


    def departing(e):
        if confirm_code.get()=='':
            confirm_code.insert(0, 'Confirm password')
            confirm_code.config(fg='orange')

    confirm_code = Entry(frame,width=25,fg='orange',border=2,bg='#2f2f30',font=('Mincrosoft Yahei UI Light', 11))
    confirm_code.place(x=30,y=220)
    confirm_code.insert(0,'Confirm password')
    confirm_code.bind("<FocusIn>",accessing)
    confirm_code.bind("<FocusOut>",departing)

    Frame(frame,width=295,height=2,bg='black').place(x=25,y=247)

    Button(frame,width=39,pady=7,text='Register',bg='orange',fg='black',border=0, command=register).place(x=35,y=280)
    label=Label(frame,text='I already have an account -->', fg='white',bg='#2f2f30',font=('Mincrosoft YaHei UI Light',9))
    label.place(x=35,y=340)

    login = Button(frame,width=6,text='Sign in',border=0,bg='#2f2f30',cursor='hand2',fg='orange',command=on_login_button_pressed)
    login.place(x=200,y=340)

    window.mainloop()
```

```
frame=Frame(window,width=350,height=350,bg="#313131")
frame.place(x=300,y=70)

heading=Label(frame,text = 'Log in',fg='orange',bg='#2f2f30',font=('Mincrosoft YaHei UI Light',23,'bold'))
heading.place(x=120,y=0)


user = Entry(frame,width=25,fg='orange',border=2,bg="#2f2f30",font=('Mincrosoft YaHei UI Light',11))
user.place(x=30,y=80)
user.insert(0,'username')


def departing(e):
    name=code.get()
    if name == '':
        user.insert(0,'Password')


def accessing(e):
    user.delete(0,'end')

user.bind('<FocusIn>', accessing)
user.bind('<FocusOut>', departing)
Frame(frame,width=295,height=2,bg='black').place(x=25,y=107)
```

```
def departing(e):
    name=code.get()
    if name == '':
        code.insert(0,'Password')



def accessing(e):
    code.delete(0,'end')


Frame(frame,width=295,height=2,bg='black').place(x=25,y=177)



register = Button(frame,width=6,text='Register', border=0,bg='#2f2f30',cursor='hand2',fg='orange', command = create_pin_entry_window)
register.place(x=215,y=270)


Button(frame,width=39,pady=7,text='Log in',bg='orange',fg='black',border=0,command=login,font=('Mincrosoft YaHei UI Light',9)).place(x
label=Label(frame,text="Don't have an account?",fg='white',bg='#2f2f30',font=('Mincrosoft YaHei UI Light',9))
label.place(x=75,y=270)


code = Entry(frame,width=25,fg='orange',border=2,bg="#2f2f30",font=('Mincrosoft YaHei UI Light',11))
code.place(x=30,y=150)
code.insert(0,'Password')
code.bind('<FocusIn>', accessing)
code.bind('<FocusOut>', departing)
```

```
  code = Entry(frame,width=25,fg='orange',border=2,bg="#2f2f30",font=('Mincrosoft YaHei UI Light',11))
  code.place(x=30,y=150)
  code.insert(0,'Password')
  code.bind('<FocusIn>', accessing)
  code.bind('<FocusOut>', departing)


  window.mainloop()
```

Button code:

```
# Class for buttons
class Button():
    # Parameters are image of rectangle,position of it,text of the button,font,and the color shown when mouse is above it
    def __init__(self, image, pos, text_input, font, base_color, hovering_color):

        # Each parameter is put in a self variable
        self.image = image
        self.x_pos = pos[0]
        self.y_pos = pos[1]
        self.font = font
        self.base_color, self.hovering_color = base_color, hovering_color
        self.text_input = text_input
        self.text = self.font.render(self.text_input, True, self.base_color)
        self.change = base_color

        # If rectangle image is not given the function checks if mouse is above the text
        if self.image is None:
            self.image = self.text
        self.rect = self.image.get_rect(center=(self.x_pos, self.y_pos))
        self.text_rect = self.text.get_rect(center=(self.x_pos, self.y_pos))

    # function to update
    def update(self, screen):
        # If there's image it is placed in such a way that text can be in the middle
        if self.image is not None:
            screen.blit(self.image, self.rect)
        screen.blit(self.text, self.text_rect)
```

```
28
29      # The fuction to check for input(Mouse button)
30      def checkForInput(self, position):
31          # Checks position of mouse and text and if the mouse has been clicked
32          if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
33              return True
34          return False
35
36      # function for hovering color
37      def changeColor(self, position):
38
39          # Checks mouse position and text position if they are same, the color of text changes
40          if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):
41              self.text = self.font.render(self.text_input, True, self.hovering_color)
42              self.change = self.hovering_color
43          else:
44              self.text = self.font.render(self.text_input, True, self.base_color)
45              self.change = self.base_color
```

References:

For the snake game I followed these tutorials to get the main prototype running:

Youtube Link: https://youtu.be/8dfePlONtls?si=eJxus3lQb7Ckg38x (To learn how to make a game in pygame)

Youtube Link: https://youtu.be/QFvqStqPCRU?si=X5lAY4a0FzTKq6wC

(To learn and implement the vector concept)

For the button:

I used this library I found in github: https://raw.githubusercontent.com/baraltech/Menu-System-PyGame/main/button.py