NEA PROJECT – SMASH PADDLE

OCR A-Level Computer Science H446-03

Thomas FEW
9056

Contents

Analysis

An outline of the problem

For my project I am planning to create a game that is like 'Atari: Breakout.' This game is a retro styled paddle game where you are given a paddle to move left and right across the screen and a ball to bounce against your paddle into blocks floating above. Also, the blocks above will be stacked similarly to a brick wall where the ones at the top, when hit, will increase the speed of the ball the higher up the block is. When a block is hit, the player will collect points depending on what colour block they hit, and the number of points collected will be added to the user's score total. Furthermore, the player is given 4 balls to use and if they happen to lose all four balls, the game is then over, and the player is given a 'game over' screen with their score displayed.

For this game to work, it will require: a score box to show the player's score whilst they play the game, a moving paddle to allow the player to knock the ball towards the blocks, four balls used to destroy the blocks, a counter displayed to show how many balls the player has left, and finally a wall of blocks that appear at the beginning of the game for the player to clear.

Stakeholders

The graphics of the game will be like that of early video games so therefore it will be a retro-styled video game with retro graphics.
There will also be no narrating or speaking inside this game therefore there will be no language complications so audiences of all ages will be able to play this game.
However, this game can only currently be played on the PC, therefore it is not one that you can play on the go and will have to sit down and take time out of your day to play.
The game could easily be adapted to play on other devices such as mobile phones as you could easily replace using the mouse to drag the paddle, with either using your finger or little buttons at the bottom of the screen.
Due to the previous things, I have stated about the game, I have decided that the target audience for my game will be all ages as the game is simple enough so that anyone is able to play the game.
I have selected a client to represent my target audience, Finlay Baker. He is a seventeen-year-old student in my computer science class who has a passion for gaming and has also played the Google representation of 'Atari: Breakout' so he will know what to expect from the game that I am creating. I will also have regular contact with my client as he is in my computing class as well as my form.

How the problem can be solved by computational methods

This game is well suited for a computer as it can be solved easily using computational methods. This is because of how easily games can be broken down from reality and how the game can contain multiple variables and conditions that can be set or the fact that the game contains simple algorithms that are used for games scenarios. I have listed the following examples to demonstrate how the game is suited to be solved by a computer program due to the computational methods.

Thinking Abstractly

I need to decide what I would like/not like to include in my game and using abstraction is perfect to help me do this.

Abstraction for 'Smash Paddle'

- Make sure the game is only 2 dimensions.
- Create a retro style for the game to make it like the original

- Add a counter at the top of the screen to help the user keep track of their score and another counter in the opposite corner to tell them how many balls they have left in the game.
- Possibly add a menu to allow the user to change certain settings like if they want the ball's speed to increase to make it harder.

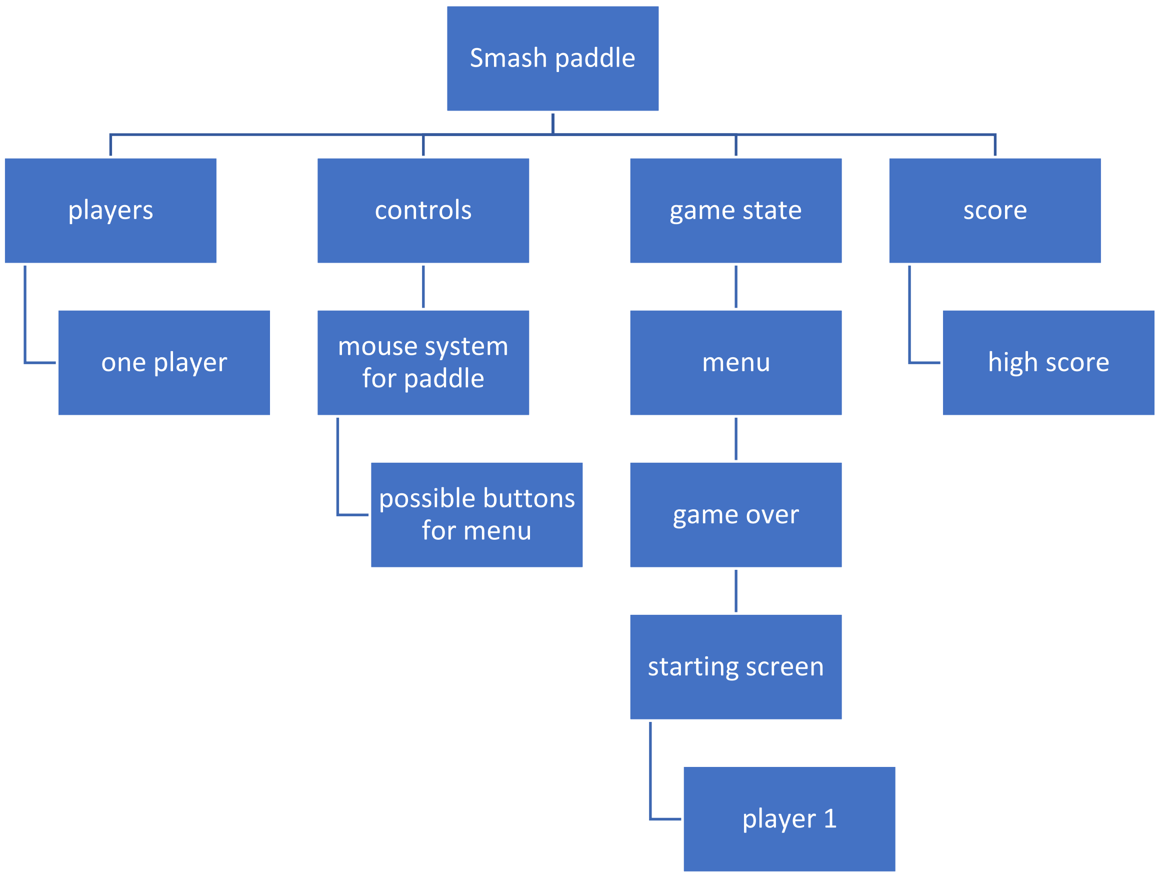
Thinking Ahead

This is a way of looking into the future and deciding what you want to have in your game. It allows you to be able to create a plan and to give your solution to a problem a bit more structure and clarity.

- I am planning to write my game in PyScripter as it has all the tools, I need for creating my game in pygame.
- Inputs for the game would be the user moving their mouse left and right across the screen to control the paddle and to hopefully navigate through the menu options where they can click on the buttons to change the settings.
- The outputs of the game will consist of sound effects such as when the ball hits the blocks and when the user runs out of lives or destroys all the blocks.
- A score system will also be put in place that will be visible throughout the game so that the player knows what score they are on. It will also give the user something to try and aim for e.g., a new high score.

Thinking Procedurally

This step is like thinking ahead as this will give the problem a little bit of structure making it slightly easier to work with. This is because now you can work with the problem in separate smaller parts which will make the entire process slightly more efficient.



This game has been broken down into four main problems: Players, Controls, Game State and Score, and by thinking procedurally, these should be solved in order of the game to create a basic functioning solution.

Players

This will decide how the game can be played. As this game will be one player, there will be one player moving the paddle across the screen trying to clear all the block on the screen.

Controls

These are vital for the player to interact and play the game. The player should be able to use the mouse to move the paddle across the screen but also hopefully to navigate through the menu to change things like ball speed.

Game State

These game states will help create a sense of depth and level in the game as it creates a basic structure for the game. There will be a starting screen to introduce the player to the game; this will then hopefully switch to the menu screen where the player can ask how the game works or change the ball speed. The player will then also be able to click start then which the screen will show the game, and the player begins to play. Furthermore, if certain conditions (e.g., no balls left) are met, the screen will show game over.

Score

This will create a sense of competition and achievement for the game. Scores will be logged in a leaderboard where other players can compete against each other to see who can get the highest score.

Thinking Logically

This step helps me to think of diverse ways I can create a solution to the logical side of the problem.

For instance, when a block is hit, a branch is created to add points to the player's score. As the game is running, a constant loop will be running to check what conditions are being met and if there is anything that needs to be added to the counters. There will also be constant iterations to check when a block is hit or when all blocks are cleared to show a game over screen.

Thinking Concurrently

This is critical for thinking about how to make my solution more efficient for when it is in use.

The game will be updating the screen as well as changing the screen when blocks need removing after being hit at the same time.

Conclusion

The previous examples can show that the solution can be created using computational methods which makes it suited for a computer program. These methods are useful for giving both the problem and solution structure so it can be created and solved more easily hence making it suited for a computer program. Computational methods such as abstraction are great as they can create benefits such as making the code more efficient, easier to understand and it makes the problem and creating a solution a lot simpler.

If the problem is successfully solved using a computer program the stakeholders will be able to play a working and functioning version of the game where they bounce a moving ball against blocks, that are hovering, which will give them several points. This is aimed to be a form of entertainment for the stakeholders in which they can enjoy the game and feel a sense of achievement and competition.

Research

INTERVIEW WITH FINLAY BAKER TO GAIN AN UNDERSTANDING OF GAME REQUIREMENTS

Interview: 14/06/24 Plan

This interview will be used to gather the basic plans and requirements for the game from the user. This will help give guidance to what type of game it needs to be.

The main two headings that I will be using for this interview are: What makes a good game and What kind of visuals/styles/effects should the game have.

I have produced blocks of smaller questions that fall under these two headings to get my client to think deeper about the questions and to check every box about the headings to cover all bases and make it is detailed as possible.

Question Layout:

What makes a good game?

- Are sound effects necessary for a good game?
- How important are visuals for an arcade game?
- Do you think that a score system should be implemented into the game?
- For 'Smash Paddle,' how many blocks do you think there should be to destroy?
- For the 'Smash Paddle' game, how would you want to be able to interact with the game?

What kind of visuals/effects should the game have?

- What type of game style would you like to be used for this game?
- Would you want any animations when a block is hit?
- Should there be a colour scheme for the blocks?
- Should there be a background for the game?

Interview Script: 14/06/24

What makes a good game?

Are sound effects necessary for a good game?

"Yes, because it makes the game more realistic and more immersive."

How important are visuals for an arcade game?

"Not too important as it does not need to be complex."

Do you think that a score system should be implemented into the game?

"Yes, because if you complete the game then you can always go back to try and beat your high score which makes the game more fun."

For the 'Smash Paddle' game, how many blocks do you think there should be to destroy?

"I think that the user should be given an option to have an 'infinite mode' where the blocks constantly regenerate until the user runs out of lives or they have the option to play the classic mode where there is say 15 blocks (3 blocks high and 5 blocks wide)."

For the 'Smash Paddle' game, how would you want to be able to interact with the game?

"Maybe use the arrow keys to move the paddle left and right or just the mouse to let the paddle follow it. I also think a menu that contains settings would be great to let the user configure the ball speed."

What kind of visuals/effects should the game have?

What type of game style would you like to be used for this game?

"Like retro, arcade-style because it is a simple game and can be used for a sense of nostalgia for the older audiences."

Would you want any animations to be used when a block is hit?

"Yes, when the block is hit perhaps it could appear to be shattered or broken into small pieces or something similar."

Should there be a colour scheme for the blocks?

"There should be an obvious colour difference between the ball, paddle, and blocks however the actual colours of the blocks does not really matter. However, the colour of the blocks could be used to signify how many hits it would take to destroy them."

Should there be a background for the game?

"I think you should be able to choose maybe between some presets for a background in the settings menu or ask the user to input three numbers for RGB to customise it how they want."

A review of the interview: 14/06/24

This interview was set up so that we could see what the stakeholders thought would make the game more interesting and enjoyable. The stakeholder and I both concluded that a settings menu would be vital and that making the blocks as detailed as possible would be crucial as well to make the game more unique and stand out.

For the visuals and sounds, we decided that an animation for when the blocks are destroyed would be good as it would add a sense of immersion to the game. Furthermore, the game will be retro styled to give a sense of nostalgia. Sound effects will also be used when the blocks are destroyed for extra detail. The game will be a 2D art style as it will be simple but still have enough detail to look clean and clear.

For the gameplay, we decided to include a score system that will log the player's score into a leaderboard after they have reached the game over screen and can play again to try and beat their previous score. We have also decided to go with the idea that the colour of the blocks will indicate how many times the ball needs to hit it to destroy it, and you will gain more points the harder the block was to destroy. The player will also be able to change the ball speed in the settings menu to add extra challenge.

Research into similar games (Atari: Breakout and Space Invaders)

The first game that I will need to research to help give some ideas for my game is 'Atari: Breakout.' This is an arcade game where the player is given a moving ball and paddle, and they need to bounce the ball against blocks to destroy them to gain points.

The game starts with 8 rows of blocks which are evenly divided into rows of colours each being 2 high. Using a ball given at the start of the game, the player must clear all the blocks on the screen by bouncing the ball against them to eliminate them. If the paddle misses the ball when it bounces back, the player loses a 'life' or a turn. The player is given three lives to clear two screens of blocks.

In the game, yellow blocks will score one point each, green blocks three, orange blocks five, and finally red blocks seven. The paddle shrinks to half its original size when the first red brick is eliminated and the ball's speed increases at set intervals at four hits, twelve hits, and when the orange/red blocks are eliminated.

The primary features of the game are:

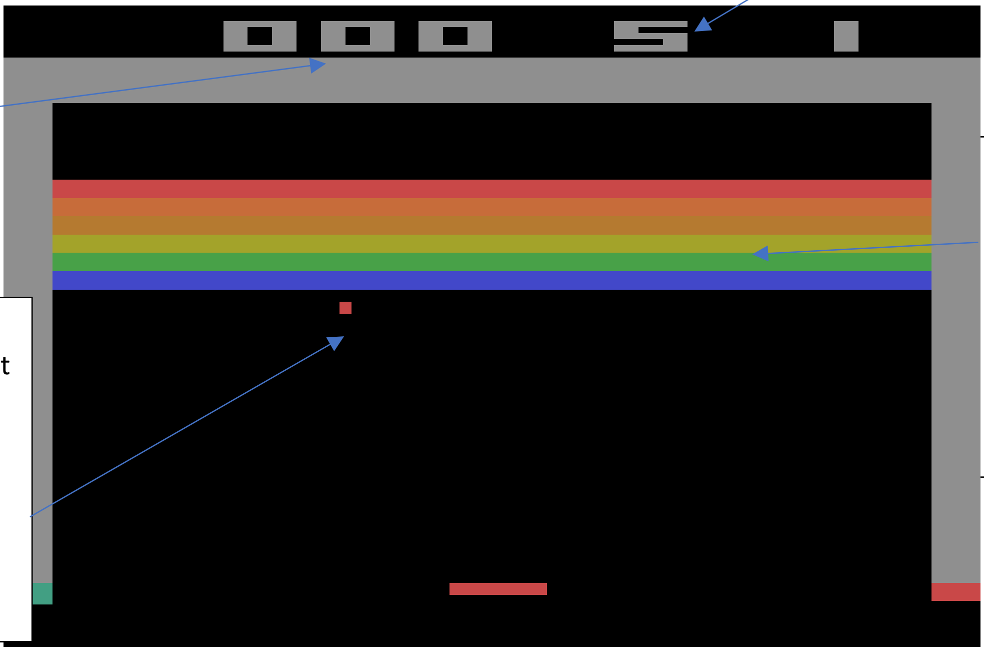
- Source for image 1 : https://www.researchgate.net/figure/Screenshot-of-Atari-2600-Breakout-game-The-ball-bounces-between-the-wall-lines-of_fig1_335394578
- Source for image 2: https://www.mobygames.com/game/9001/breakout/screenshots/atari-2600/40937/
- Source for image 3: https://javatari.org/?PAGE_BACK_CSS=rgb(188,179,143)&ROM=http://www.atarimania.com/2600/dumps/breakout.zip
- Source for image 4: https://www.youtube.com/watch?v=zluH_4eSmPs

Unfortunately, I was unable to find a playable copy of the game online and I do not possess an Atari 2600, therefore I had to take screenshots from online to show the screens of the game. All the links to the sources I got these images from have been stated above.

Atari Breakout:

The number of available balls left for the player is shown here.

The player's score is shown here.



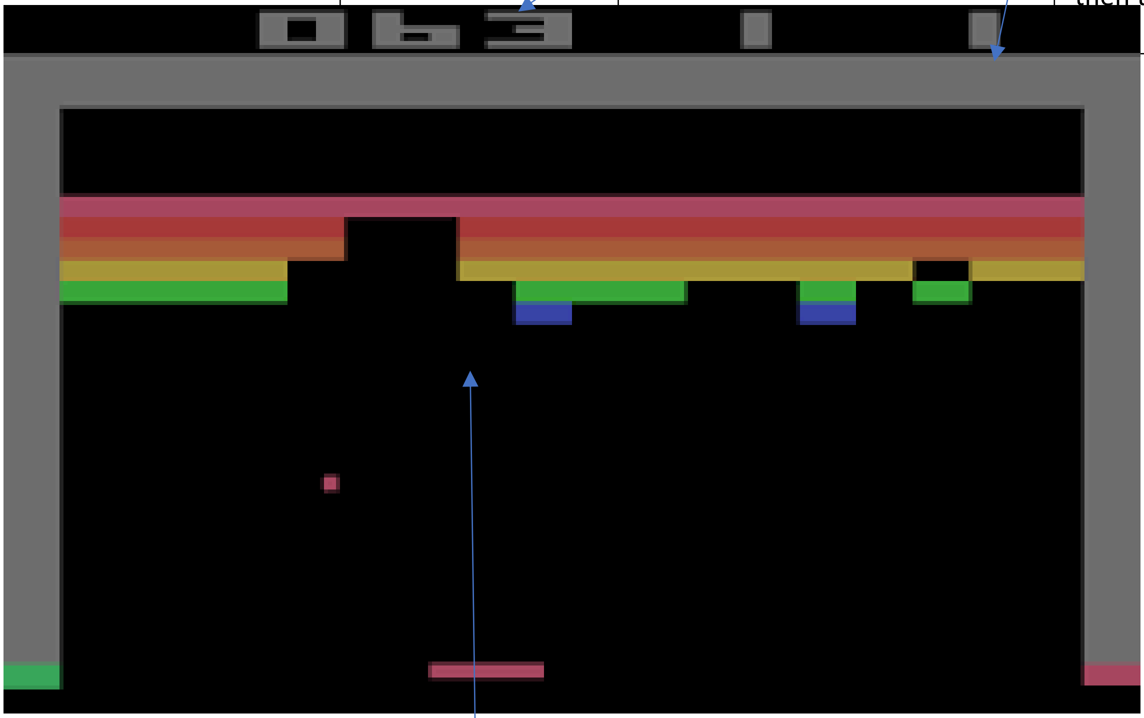
The blocks are displayed in rows of colours. The more/higher you hit, the quicker the ball becomes.

The player's ball is randomly thrown out at the start of the game and then it is the player's job to rebound it into the bocks to eliminate them.

The player's paddle is shown at the bottom of screen and the player can change the speed of the paddle.

Player's score increases when blocks are destroyed, and higher blocks give more points.

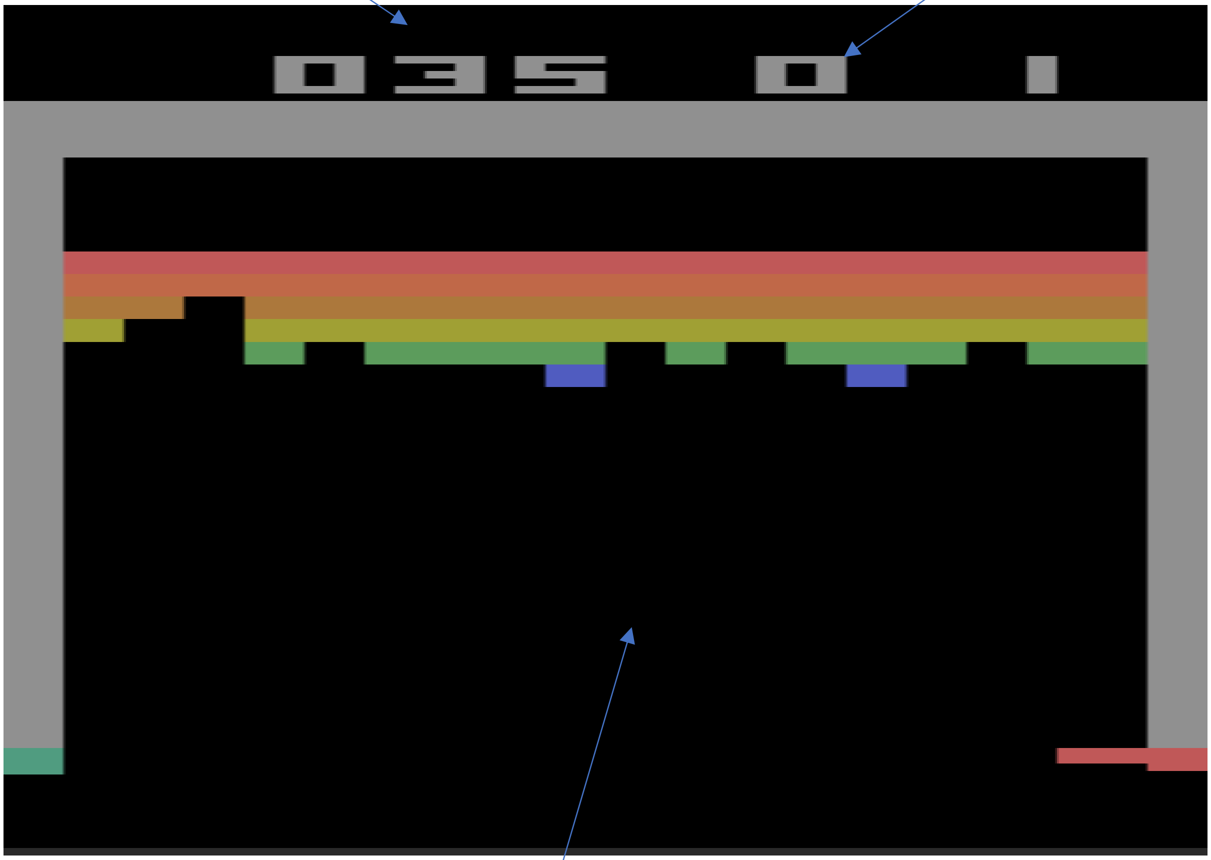
When a player loses a turn, this counter will decrease until it gets to zero and then the game will



One of the upper two layers have been destroyed so the ball's speed will increase. It will also increase as a minimum of five blocks have been hit.

The player's final core is displayed at the top of the screen.

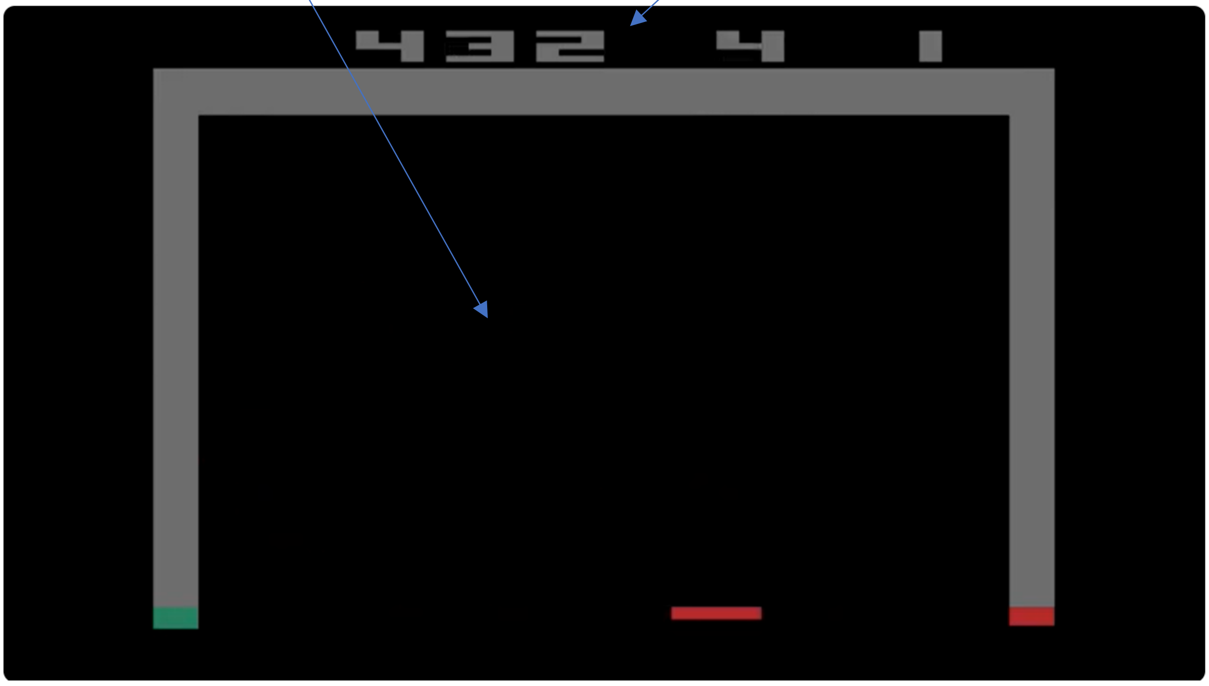
At this stage, the player has run out of turns, therefore the game is over, and they will have to start again.



Because the player has run out of lives, no more balls will appear and therefore the player will have to restart the game and try again.

Once all the blocks have been destroyed, the game is over, and the screen turns empty as shown. None of the remaining balls will be spawned once the game is won.

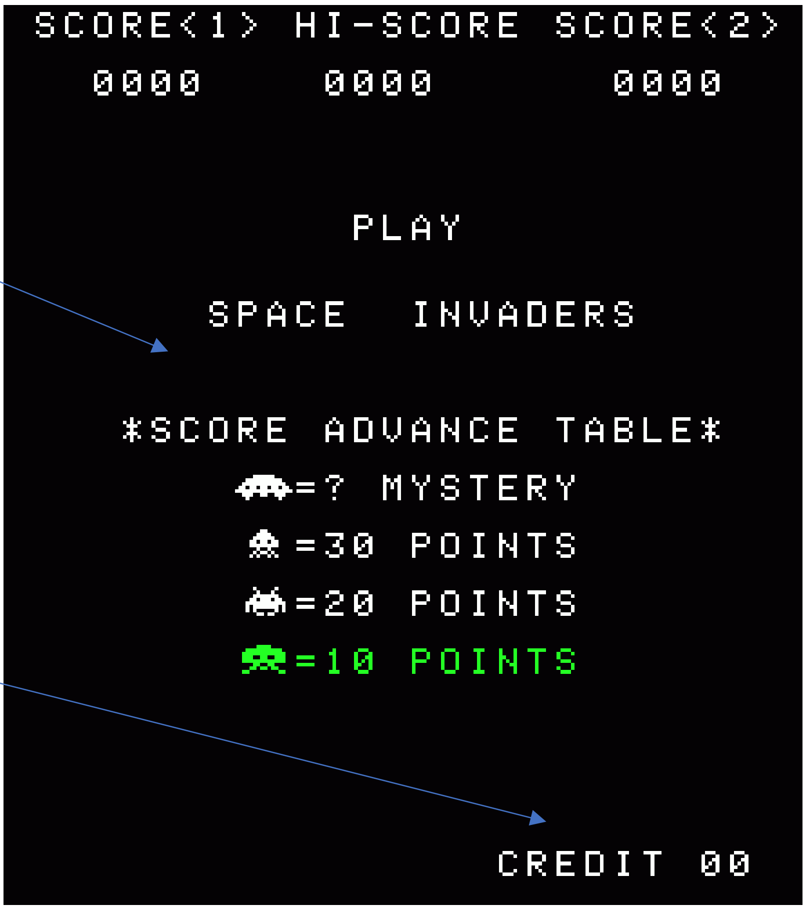
Just like when you run out of lives and the game is over, the player's final score is shown at the top of the screen.



Space Invaders:

- Source for image 1: https://www.researchgate.net/figure/Title-screen-and-illustration-of-gameplay-for-Space-Invaders-1_fig3_332254163
- Source for image 2: https://www.youtube.com/watch?v=phNT1yIrDzA
- Source for image 3: https://game-over-dex.fandom.com/wiki/Space_Invaders

At the top of the menu, it shows player 1 and player 2's score as well as what the current high score is. As players eliminate enemies, their score increases.



SCORE<1> HI-SCORE SCORE<2>
0000 0000 0000

PLAY

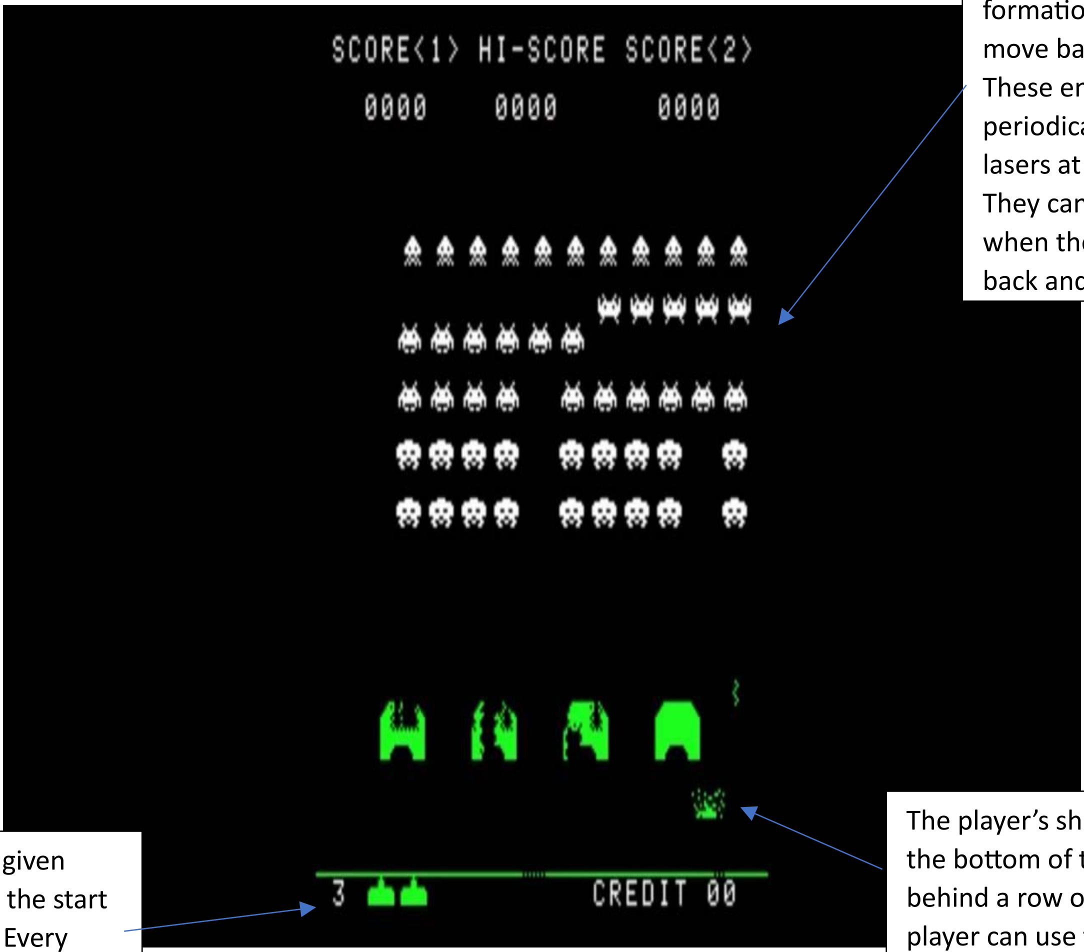
SPACE INVADERS

SCORE ADVANCE TABLE
=? MYSTERY
=30 POINTS
=20 POINTS
=10 POINTS

CREDIT 00

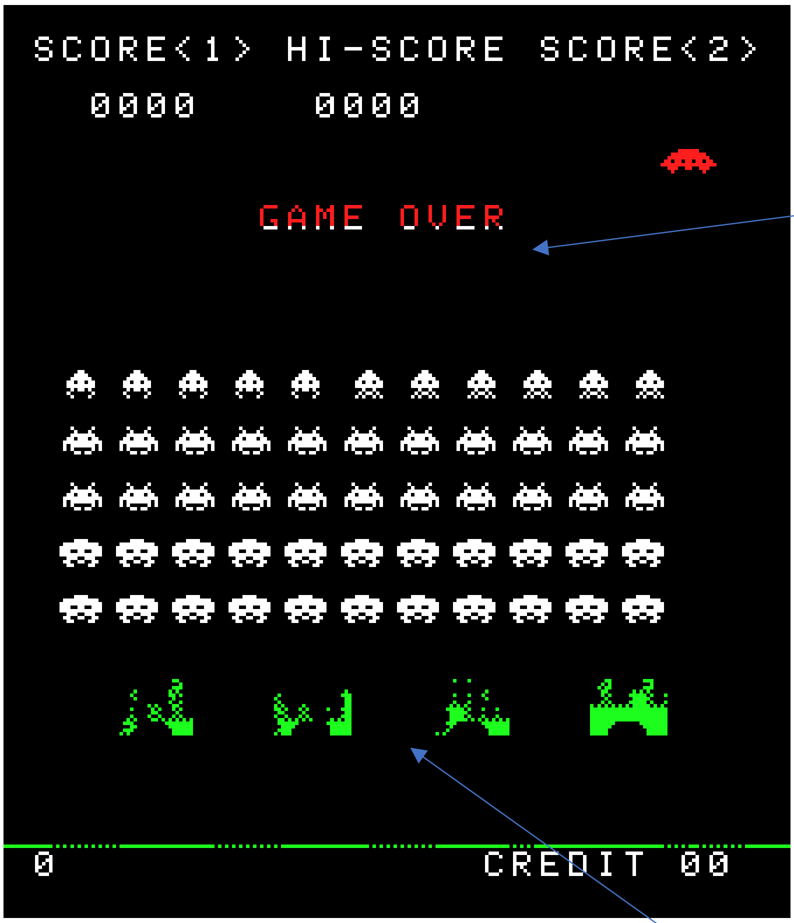
The menu displays the name of the game, a button to click to start the game; and a score table to show how much each enemy is worth in points after they have been defeated. The credit is because of the arcade era of the game where players would enter money for goes.

When the game starts up, enemies spawn on the screen in a grid formation and begin to move back and forth. These enemies will also periodically shoot tiny lasers at the player. They can be eliminated when the player fires back and hits them.



The player is given three lives at the start of the game. Every time that they are hit by an enemy's laser, the lose a life. If they lose all three of their lives before the game is over, they lose the game.

The player's ship spawns at the bottom of the screen behind a row of shields. The player can use these shields to take cover from enemy fire however when they are hit, they take damage and can disappear. The objective is for the player to destroy all of the enemies without losing all of their lives.



As this screen shows, when the player loses all three of their lives, the game comes to an end and their score is displayed at the top of their screen as well as the high score. If the high score is broken, a new one will be set and will appear the next time it is played.

As the game moves on, the shields are destroyed the more they are hit. So, if the player has reached through multiple waves, the shields will most likely be almost destroyed.

How do the features of the two games effect the gameplay and style of them?

Game 1 – Atari Breakout:

- Grey border – allows the player to see where the walls are so they can anticipate where the ball will bounce and rebound to. I can incorporate this into my game by giving the game a border to allow the player to see where the ball will bounce, however it will require changing a large amount of the code and would not be necessary as the games border is the small screen it plays on.
- Top half of the screen – the top half of the screen contains the player's score; how many turns they have left and how many players are currently in use. This is crucial for the game and is useful for the person playing the game as it allows them to be kept up to date with what is going on and how well they are doing. I can incorporate this into my game by displaying the players score at the bottom of the screen as well as the amount of lives the player has left.
- Vibrant-coloured blocks – at the top of the screen, vibrant coloured blocks are spawned for the player to destroy with the ball and paddle. It is useful for the player that the blocks are coloured differently as it makes it clearer what row they are on so that the player can tell how many points they are worth and how it will affect their score. I have incorporated this into my game by making each row of blocks a different colour so that the player can tell what they need to aim for to get the highest score.
- Sound effects – In the game, when you destroy a block, there is a high-pitched sound effect when the ball spawns, a low pitch effect when the paddle hits the ball, and a pitch sound that depending on what level of blocks you hit, changes pitch. I can incorporate this into my game by making adding a sound effect whenever the ball hits anything including the paddle, blocks, and walls. I can also make a sound effect for when a block is destroyed.

Game 2 – Space Invaders:

- Top of the screen – the top half of the screen displays the player's score, the high score and the second player's score. This useful for the player as they can see their live score and what they need to get to beat the high score. I will be able to incorporate this into my game to allow the player to see their scores and what they need to improve on to get the high score.
- Bottom of the screen – the bottom of the screen allows the player to see how many lives they have left. This is useful for the player as it allows them to see how careful they need to be which creates strategy in the game. I can incorporate the same idea into my game except the score will be at the top of the screen instead of the bottom.
- Menu – at the start of the game it displays a menu which allows the player to see how many points each enemy is worth when they are eliminated and allows them to start the game when they are ready. This is useful for the player as it gives them all the useful info they need before playing. I will be able to incorporate this into my game by creating a staring menu by giving the option to see how to play the game, controls and even possibly a settings menu to change the difficulty (speed of ball).
- Visuals/Animations – the visual of the game makes it more exciting and immersive as it creates a futuristic sense in space and the colours of the aliens pop out which could immerse the user more. I will incorporate this into my game by making the blocks have vibrant colours to stand out more and interest the player.

Interview 26/06/24 plan

This interview will go into more detail about the game to get an idea about specific requirements for the game. This includes things like certain visuals, sound effects, gameplay, and menu. This interview will also allow me to get an idea for the first few steps to creating the ideal solution to the problem.

Layout of Questions:

Sound effects/Visuals

- What animations do you want for the blocks?
- What sound effects should the blocks have when they are destroyed?
- Do you want there to be an animation for when the player clears the screen?
- What kind of theme/style should the game have visually?
- What sound do you want the ball to make when it hits the walls?
- What sound do you want the ball to make when it hits the paddle?
- Should there be a sound effect when the player wins/loses?

Gameplay/Menu

- How many rows of blocks should there be?
- Do you want an unlimited number of balls/lives, or should it be a set amount?
- How would you increase the difficulty of the game?
- Should the life counter be at the top or bottom of the screen?
- Where should the player's score be displayed?
- Would you like a pause/menu button and if so, what should it be?
- Do you want a controls/how to play section of the menu?

Score system

- Should there be a leaderboard of some sort?
- How many points should each block be worth?
- How many places should the leaderboard show?

Interview 26/06/24 script

Sound effects/Visuals

What animations do you want for the blocks?

"If they get hit, they flash for a second and then they disappear."

What sound effects should the blocks have when they are destroyed?

"A high-pitched sound to signify the fact you are getting points."

Do you want there to be an animation for when the player clears the screen?

"Yes, the 'You Win' message could float up from the bottom of the screen into the middle."

What kind of theme/style should the game have visually?

"Early retro therefore it looks simple and clean but also nostalgic and fun."

What sound do you want the ball to make when it hits the walls?

"A medium pitched sound as long as it is not the same as when it hits the blocks."

What sound do you want the ball to make when it hits the paddle?

"The same pitched sound as when it hits the walls."

Should there be a sound effect when the player wins/loses?

"It should say either you win or lose and there should be trumpet sounds when you win or a trombone when you lose."

Gameplay/Menu

How many rows of blocks should there be?

"Six rows."

Do you want an unlimited number of balls/lives, or should it be a set amount?

"A set amount which the player can change in the menu."

How would you increase the difficulty of the game?

"Increase the speed of the ball."

Should the life counter be at the top or bottom of the screen?

"Bottom of the screen so it does not overlap with the blocks."

Where should the player's score be displayed?

"At the bottom alongside the life counter."

Would you like a pause/menu button and if so, what should it be?

"Yes, and it should be 'P' for pause."

Do you want a controls/how to play section of the menu?

"Yes, as it would be useful for someone who has never played this style of the game."

Score System

Should there be a leaderboard of some sort?

"Yes, as it adds more competition to the game."

How many points should each block be worth?

"Top 2 rows should be worth 250 points; middle rows should be worth 150 points and bottom rows worth 50 points."

How many places should the leaderboard show?

"Top 10."

Review of the interview 26/06/2024

Through this interview, the main needs and features required of the game have been established and now I can work forward to meet these goals.

Sound effects/Visuals

- Animations for the blocks when they get destroyed.
- High pitched sound when blocks are destroyed.
- Lower pitched sound for whenever the ball hits the walls or paddle.
- Win screen with trumpet sounds.
- Loser screen with sad trombone sounds.

Gameplay

- Six rows of blocks.
- Set number of lives that can be changed in the menu.
- Changing the speed of the ball changes the difficulty.
- Both life counter and player's score should be displayed at the bottom of the screen.
- Pause button should be implemented to allow the player to access the menu during the game.
- Controls section will be created inside the menu.

Score System

- Leaderboard should be created.
- The top 10 scores will be displayed.
- Each couple rows of blocks will be worth different number of points.

Main Features of the proposed game:

These are the main features for the game I have proposed.

Score counter	This acts as a way for the player to keep track of their score as they play the game, so they will have an idea of how well they are doing.
Set number of lives per game	This was done to give the game a sense of tension and strategy as it would make the player think differently about how not to waste turns
Counter for how many balls is left	This acts as another indicator for the player to see how many lives are left in the game
Keyboard controls	This allows the user to move the paddle left and right by using the two arrow keys. This is the only control which means most people if not everyone can play the game
Game over screen	This allows the player to be able to see their final score and is given the option to restart or go back to the main menu.

Background	This will display the user's score towards the bottom of the screen so that it does not get in the way of the gameplay.
Sound effects	This gives the player an indication of what is going on in the game and how many points they have collected from the block. It also creates more immersion in the game.
Core mechanics	The game will use random x and y movements to send the ball in an accurate direction when it bounces.
Game over animation	Once the game is over, it will display the player's final score and there will sound effects and a pixel art game over title will appear.
Visuals	The retro style visuals will appear simple yet clean therefore will not be too distracting to the user when they are playing the game, but it will still be enjoyable to play.
Pause menu	Allows the player to pause the game by pressing a simple button to stop the game without losing their score/lives.

Future Features for the game

As I am currently taking two other A-Levels, I have a time constraint of around 6 months including the summer holidays as I am the only person currently working on this game. This means that I will not be able to complete all the features that I would have liked to include. Therefore, I have included some features below that I would like to have completed if not for the time constraint.

Feature	Justification	Time Constraint
Extra game modes	This will not only add more immersion, but also a whole new line of possibilities with modes like infinite mode where the player has unlimited blocks to destroy	This would take a lot of time as it would require a whole new game's worth of code to create a new level for which I do not have the time.
Shading/Shadows	This would add more immersion as well as depth and detail to the game.	This would require adding an extra layer of colouring or even possibly to make a whole new block to put on top.
Ball speed	When the ball hits the upper blocks, the ball would speed up. This would create tension and excitement as the player would be trying hard to clear the level.	This would take a lot of time to create and implement as it would require a lot of changes to both classes and a whole new separate script of code.
Paddle speed	This would allow the player to change the speed of the paddle to make it easier if the ball ever sped up.	As with the ball speed, this would require a lot of in-depth changes and a whole separate script

Hardware Requirements

Hardware	Justification
Monitor	The player would need a monitor to see the game.
2.5+GHz	Minimum requirement of Windows 10
4+GB	Minimum requirement of Windows 10, however running on Windows 11 would allow full potential for the game.
~GB of free space	To be able to download the game and all its textures/graphics.
Dedicated/Integrated Graphics	To be able to output the game onto the screen and be able to play it.
Speaker/Headphones	To be able to hear all the sound effects included in the game.

Software Requirements

Software	Justification
Windows 10	To be able to install and play the game.
Python	To be able to run the game as the language is in python.
Pygame	This is the game engine that is required to run the game and the sound effects

Success Criteria

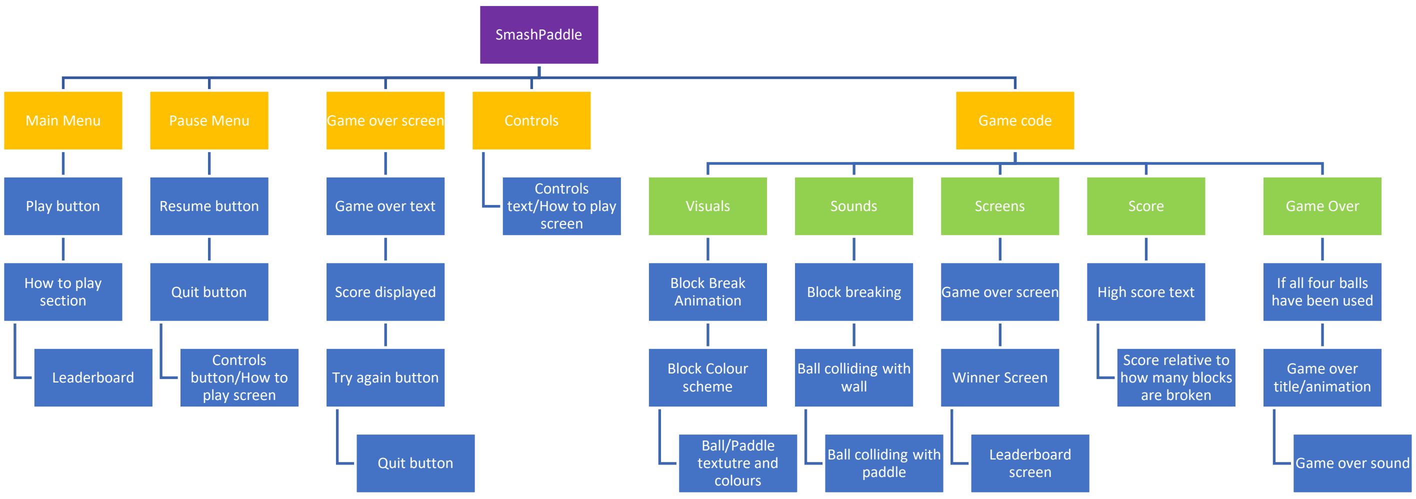
Requirements	Justification	☑	Reference
Main Menu	This allows the player to be able to play the game.	☐	Interview – 14/06/24
A score based on how many blocks/what blocks they destroyed	This allows the user to see their progress and how well they are doing	☐	Feature of “Atari Breakout” and “Space Invaders”
Ball Bouncing Mechanic	This is what allows the player to destroy the blocks and effectively play the game	☐	Feature of “Atari Breakout”
Pause Menu	This allows the user to pause the game if needed without losing their score	☐	Interview – 26/06/24
Realistic Ball Bounces	This makes the ball realistic and able to predict where it will bounce	☐	Feature of “Atari Breakout”
Game over screen	This will display the user’s final score and then direct them to the main menu	☐	Interview – 26/06/24

Only Keyboard Controls	This will make the game more accessible so that people of all ages and abilities can play the game	☐	Interview – 14/06/24
Relevant sound effects for the game	This is needed to make the game more immersive and fun to play	☐	Interview – 14/06/24
Score displayed at the bottom of the screen	This allows the player to see their score without being too distracted	☐	Stakeholder
Game over sound effect	To notify the user that the game is now over and that they can quit or restart	☐	Interview – 26/06/24
Square Screen	To make the window fit with the way that the game is drawn and orientated	☐	Feature of "Atari Breakout" and "Space Invaders"
Ball Bouncing Sound	To make the game more immersive and responsive	☐	Feature of "Atari Breakout"
Block Breaking Sound	To make the game more fun and immersive	☐	Feature of "Atari Breakout"
Paddle contact sound	Overall to make the game more satisfying by adding contact sound effects	☐	Feature of "Atari Breakout"
Easy to use Buttons	Simple buttons for the user to press so it is easier to navigate menus and play the game	☐	Stakeholder and Interview – 14/06/24
Smooth animations and gameplay	This will make the game more satisfying to play but also it will make the game look cleaner and well made	☐	Stakeholder
Different levels of blocks	Makes the game more immersive as it adds levels of depth and challenge	☐	Interview – 14/06/24
Pixel Art Texture for each menu screen	To make the game look more appealing and retro	☐	Stakeholder
How to play menu	To show the user how to play the game if they have never	☐	Interview – 26/06/24

	played similar style games before		
Level Cleared Screen	Congratulates the player when they have cleared the screen of blocks, so they have as sense of achievement	☐	Interview – 26/06/24
Scores Database	The player’s high score will be saved into an SQL Database, and they can have their score added to a leaderboard	☐	Stakeholder

Design

System Diagram



I am going to be using this concept throughout working on this project as it lets me break down the solution and problem into smaller chunks that I will be able to work on individually. This will mean I can plan out each little bit before I code it so that the code will be easier to make but also more detailed. This will also allow me to think ahead for what each part of the code will need to have.

Module Breakdown

Score

- The score is relevant to the number of blocks that have been broken as well as what colour/position they are.
- If the player's current score is higher than the current high score when the game is over, the high score will be updated.

- Displays the current score in the pause menu as well as the bottom of the screen while the player is playing the game. It is also displayed along with the current high score when the game over screen/winner screen appears.
- The high score will be saved in an SQL Database so that when the game closes the scores will be saved for the leaderboard.

Pause Menu

- This appears when the player presses the pause button during the game.
- When they press the pause button, the game will be paused and it will show the player's current score, the high score and gives them the option to continue or quit the game.

Main Menu

- This will display the game title and the current high score.
- There will be a play button displayed as well as an option to see the controls and how to play the game.

How to play screen

- This will display text and images informing the player of the controls as well as the objective of the game and how the scoring works so the player knows how much each block is worth in terms of points.
- There will also be a return button to click that will return the player to the main menu so that they can start the game.

Main Game

- Most of the variables will be reset such as the current score variable.
- The paddle will be drawn in the middle-bottom of the screen and the ball will appear when the player is ready to start the game.
- The ball will start at a slow speed so that the player does not immediately lose a life when starting the game.
- When the ball appears, it will be targeting towards the paddle so that even if the player does not move the paddle, it will still collide.
- If a collision occurs, the x and y positions will be used to give an accurate prediction to where the ball should bounce towards.
- The collide checker will be constantly checking to see if the ball has collided with something.

Left/Right Arrow Keys

- These allow the player to move the paddle left and right when they press the corresponding keys.

Sounds

- When the ball collides with a block, "ball block" sound is played.
- When the ball collides with the paddle, "ball paddle" sound is played.
- When the player loses a turn/life/ball, "ball loss" sound is played.
- When the game is opened, "game open" sound is played.
- When the game is over, "game over" sound is played.
- When the ball bounces off a wall, "ball wall" sound is played.

Speed

- To change the difficulty of the game, the player can change the ball speed in the settings menu to make it more challenging and exciting for them.

Game Over

- Once all the blocks are destroyed or the player has run out of turns, the game is over.
- If they have destroyed all the blocks, a winning animation comes on the screen with a title saying, ‘You Win!’ and a sound plays. This will then prompt the user to head back to the main menu or to start another game if they wish.
- If they have run out of lives before all the blocks are destroyed, the game over screen will come on with a different sound playing. It will display a title saying the player lost and will again prompt them back to the main menu or to start another game.
- The player’s final score and the current high score will also be displayed under the title that comes up in the middle of the screen.

Algorithms

Score Update

```
          if block strength is less than 1 then
               block strength lowers by one

            score = score + 5

          else:
            block is destroyed

            score = score + 20
```

Main Menu

```
procedure game_intro()
    intro = True

    while intro = True
      if quit button is pressed then
          quit game
        endif
      fill game screen with bg colour
      display game title in large text
      centre the title in the middle of the screen
        display start button
        display quit button
      update display
endprocedure
```

Button click arrow keys

```
If arrow key is pressed and game is running within boundaries then
        If left arrow key is pressed then
                paddle moves left
        if right arrow key is pressed then
                paddle moves right
        endif
```

Class Ball

```
Class ball()
        method initialize (x, y, speed)
                ball radius = 10
                x value equals x value minus ball radius
                y value = y value
                x velocity equals 4
                y velocity equals -4
                max velocity equals 5
                game over equals 0
```

Draw Ball

```
Function draw()
   Draw filled circle on the screen then
      Use paddle_colour
      Draw ball onto the middle of the paddle at the top
      Set radius of the circle to 10


   Draw outlined circle on the screen then
      Use colour paddle_outl
```

```
    Draw outlined circle onto the ball
    Set radius of the circle equal to the ball's radius
    Set outline thickness = 3
endfunction
```

Update Ball

```
Function move()
  collision_thresh = 5
  Initialize wall_destroyed to 1
  Initialize row_count to 0
  For each row in wall.blocks then
    Initialize item_count to 0

    For each item in row:
      If self.rect collides with item[0] then
        If collision is from above (bottom of self.rect is near top of item[0]) and self.speed_y is positive then
          Reverse self.speed_y direction
      endif
        If collision is from below (top of self.rect is near bottom of item[0]) and self.speed_y is negative then
          Reverse self.speed_y direction
        endif
        If collision is from the left (right of self.rect is near left of item[0]) and self.speed_x is positive then
          Reverse self.speed_x direction
        endif
        If collision is from the right (left of self.rect is near right of item[0]) and self.speed_x is negative then
          Reverse self.speed_x direction
        endif
        If block strength (item[1]) is greater than 1 then
          Decrease block strength by 1
```

```
            Else then
               Set block color to transparent (0, 0, 0, 0)
            endif
         If block is not transparent (color is not (0, 0, 0, 0)) then
            Set wall_destroyed to 0

         Increment item_count by 1

      Increment row_count by 1
   endif
   If wall_destroyed = 1 then
      Set self.game_ovr to 1
endif
   If self.rect is outside the horizontal bounds of the screen then
      Reverse self.speed_x direction
   endif
   If self.rect is outside the vertical bounds of the screen then
      If top of self.rect is above the screen then
         Reverse self.speed_y direction
      If bottom of self.rect is below the screen then
         Set self.game_ovr to -1

   If self.rect collides with player_pad then
      If collision is from above (bottom of self.rect is near top of player_pad) then
         Reverse self.speed_y direction
         Adjust self.speed_x by player_pad.direction
         Ensure self.speed_x is within speed_max bounds
      Else then
         Reverse self.speed_x direction

   Update self.rect position by adding self.speed_x and self.speed_y
```

```
 Return self.game_ovr
endfunction
```

Check Ball Collide

```
if ball hits left or right wall then
    change x value in opposite direction
endif
```

Class Paddle

```
Class paddle()
    method initialize (x, y, height, width, speed, direction)
        Paddle height equals 20
        Paddle width equals screen width divided by number of columns
        X value equals screen width divided by 2 minus the paddle width divided by 2
        Y value equals screen height minus the paddle height times by 2
        Paddle speed equals 10
        Paddle direction equals 0
    endmethod
```

Draw Paddle

```
procedure draw
    draw paddle
    draw outline
endprocedure
```

Check Ball collide with block

```
//checks if collision was from above
      If collision was from greater y value than ball then
        self.speed_y  reverses



      //checks if collision was from below
      If collision was from smaller y value than ball then
```

```
            self.speed_y  reverses



          //checks if collision was from the left
          If collision was from smaller x value than ball then
            self.speed_x  reverses



          //checks if collision was from the right
          If collision was from greater x value than ball then
            self.speed_x  reverses



          //lowers the blocks strength if it has been hit by the ball
          if block strength is less than 1 then
               block strength goes down by 1

            score = score + 5

          else:
            block is destroyed


            score = score + 20
```

Check Ball collide with paddle

```
If self.rect collides with player_pad then
   // Check if the collision is from above
   If the distance between the bottom of self.rect and the top of player_pad.rect is less than
collision_thresh then
```

```
  and y velocity is positive then
   Reverse the direction of y velocity
   Adjust x velocity by adding player_pad.direction
   endif
   // Ensure x velocity stays within maximum speed limits
   If x velocity exceeds max velocity then
     Set x velocity to max velocity
   Else if x velocity is negative and less than negative max velocity then
     Set x velocity to negative max velocity
 Else then
   Reverse the direction of x velocity
      endif
      endif
```

Class Wall

```
Class wall()
      method initialise (height, width)
              block width equals screen_width // columns
              block height equals 50
```

Draw Wall

```
Procedure draw_wall()
  For each row in self.blocks then
    For each block in row then
      If block strength is 3 then
        Set block_colour = yellow
      Else if block strength is 2 then
        Set block_colour = orange
      Else if block strength is 1 then
```

```
        Set block_colour = red

      Draw a filled rectangle on the screen then

        Use block_colour
        Draw the rectangle at block[0] (block position and size)

      Draw a rectangle outline on the screen then

        Use background color bg
        Draw the outline at block[0] with a thickness of 3

endprocedure
```

Game Over

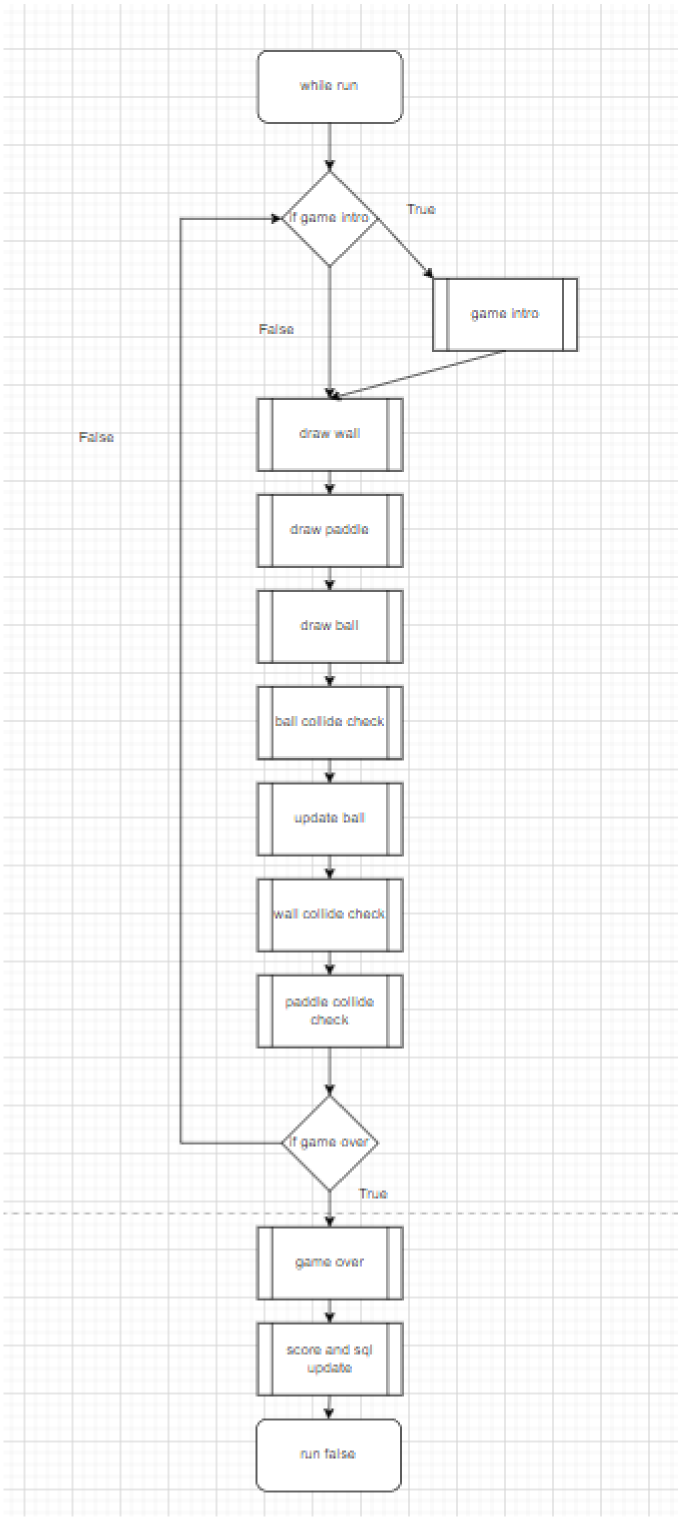
```
If not live_b then
  If game_ovr is 0 then
    Display text 'CLICK ANYWHERE ON THE SCREEN TO START'
      Use font
      Use text color txt_colour
      Position the text at (100, screen_h // 2 + 100)

  Else if game_ovr is 1 then
    Display text 'CONGRATS, YOU WON!'
      Use font
      Use text color txt_colour
      Position the text at (240, screen_h // 2 + 50)

    Display text 'CLICK ANYWHERE ON THE SCREEN TO START'
      Use font
      Use text color txt_colour
      Position the text at (100, screen_h // 2 + 100)

  Else if game_ovr is -1 then
    Display text 'BETTER LUCK NEXT TIME! :('
      Use font
      Use text color txt_colour
      Position the text at (175, screen_h // 2 + 50)

    Display text 'CLICK ANYWHERE ON THE SCREEN TO START'
      Use font
      Use text color txt_colour
      Position the text at (100, screen_h // 2 + 100)
```

SQL Database

```
Function sql_connection()

Try connect to ‘mydatabase.db’
```

```
If valid connection then
        Return valid connection
Else
        Return error
Endif
Endfunction
Function sql_table()
Try connect to ‘mydatabase.db’
If valid connection then
        Create a table called ‘Scores’ with columns for name and score
Else
        Display message
Endif
Endfunction
Function InsertData()
Try connect to ‘mydatabase.db’
Name equals string name
Score equals integer score
If valid connection then
        Insert score and name values into table ‘Scores’
        Display successful message
Else
        Display error message
endif
Finally
If valid connection then
        Close connection
        Display message
Endif
endfunction
```

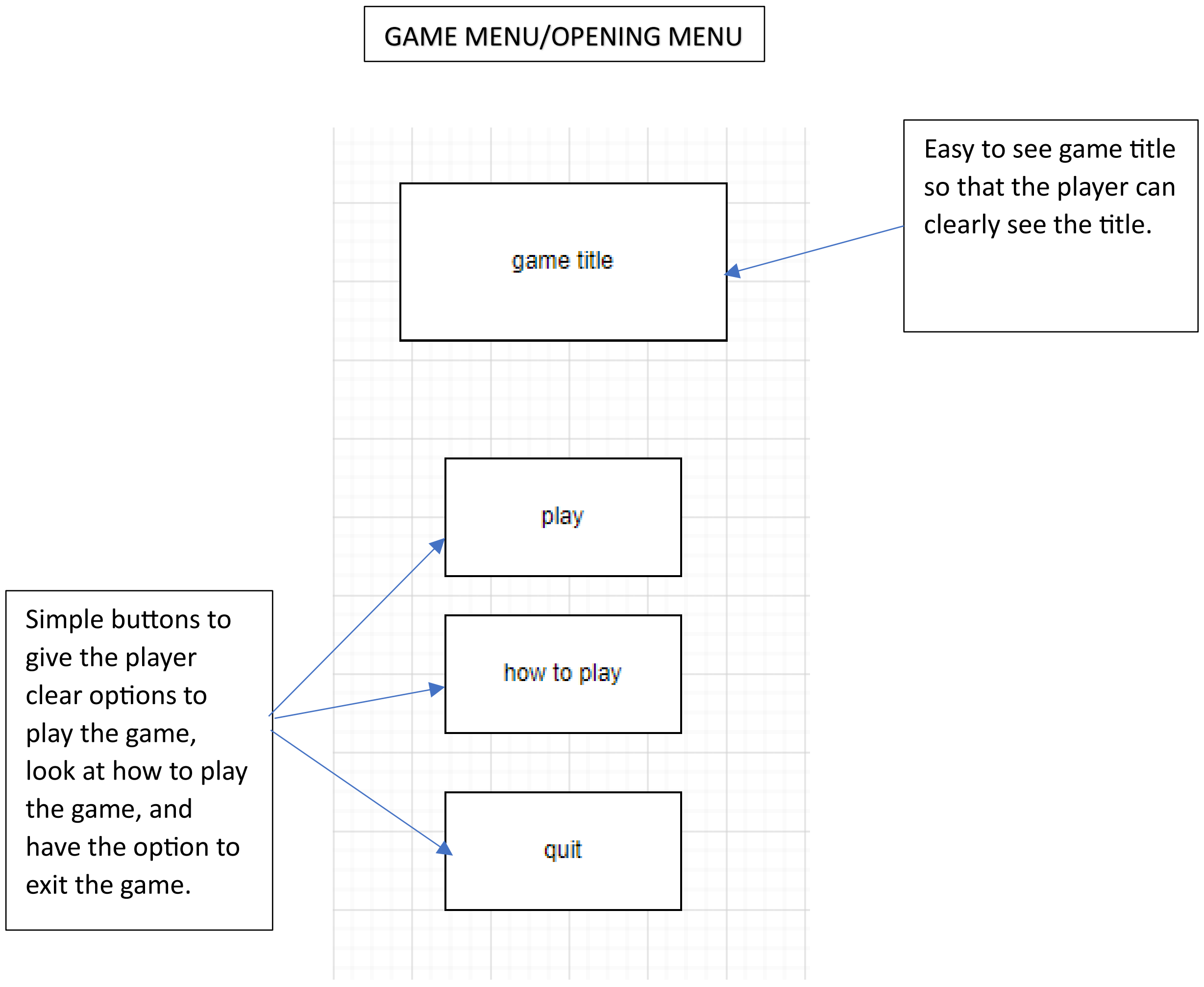
Flowchart to show how the algorithms work



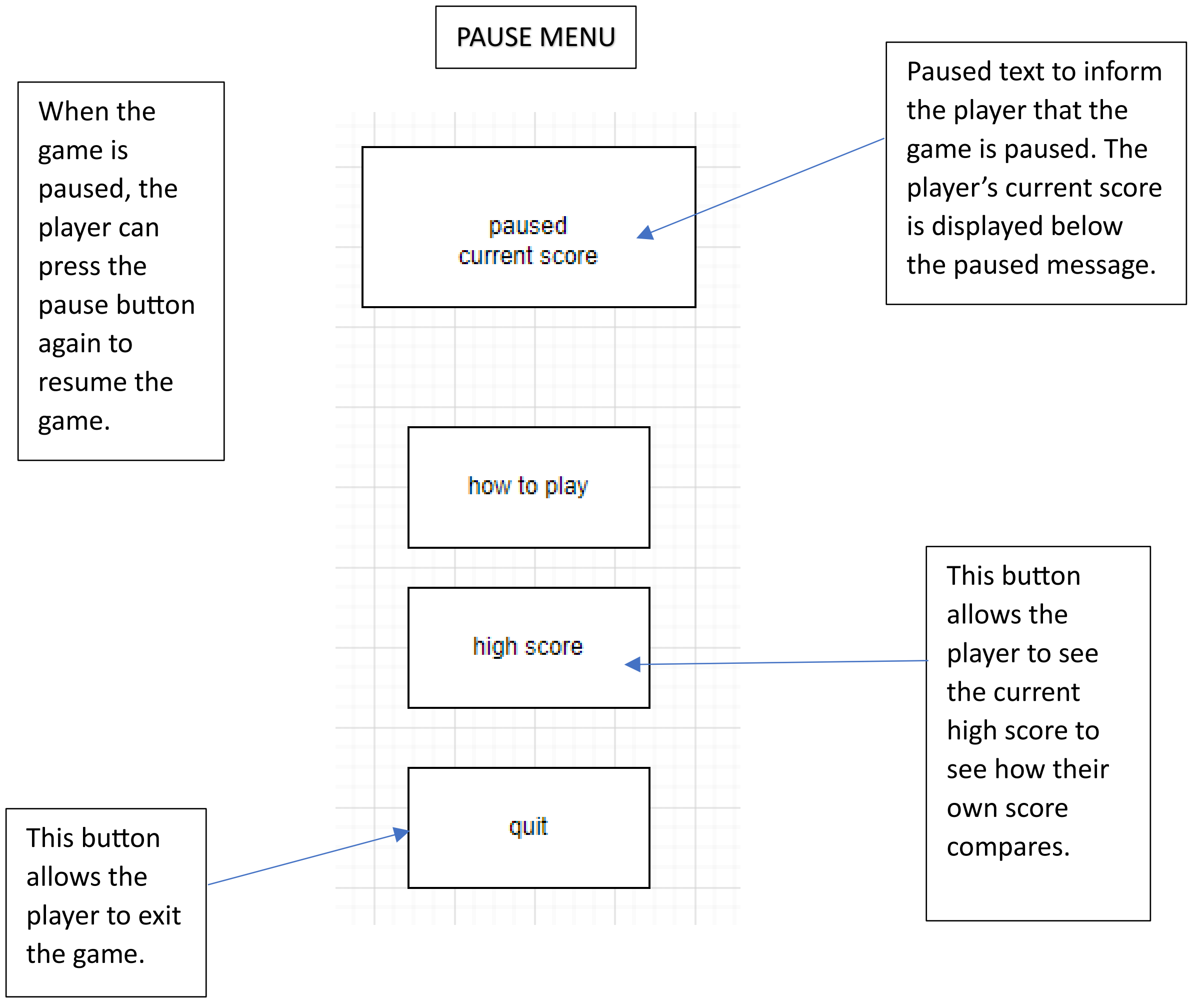
Usability Features

I have used a program called 'Draw.io' to generate designs for the usability features.

Game menu/Opening menu:

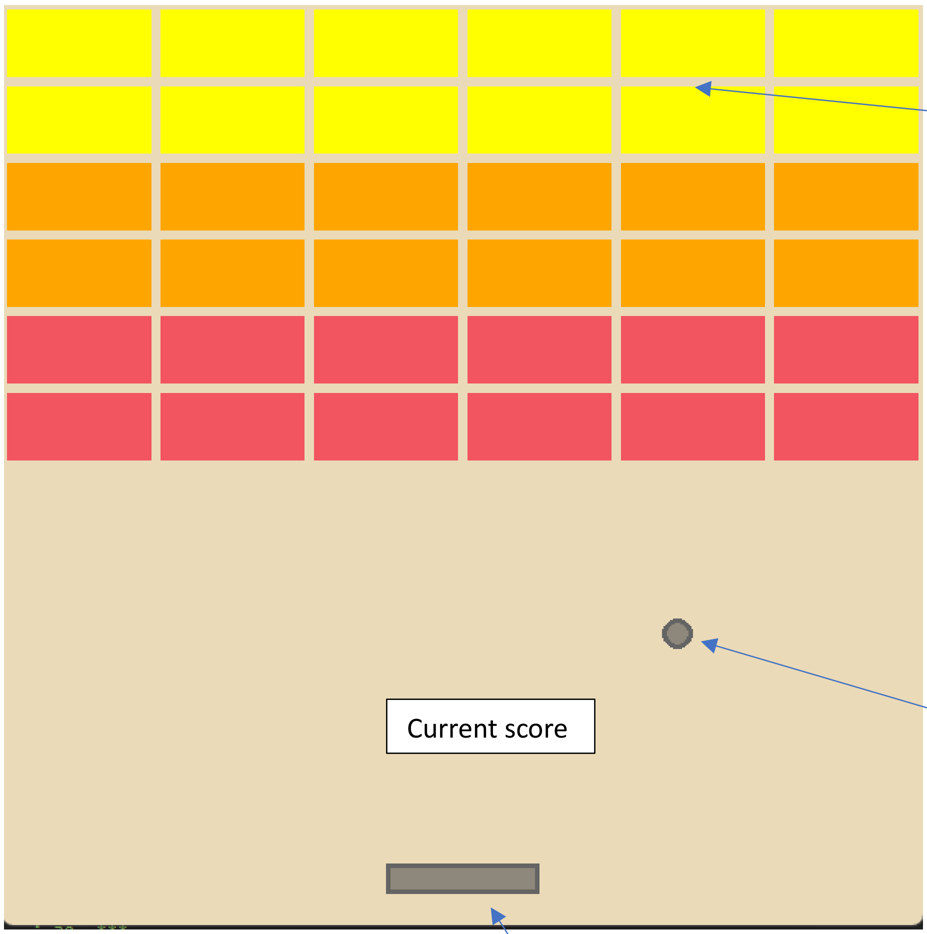

GAME MENU/OPENING MENU
game title
play
how to play
quit
Easy to see game title so that the player can clearly see the title.
Simple buttons to give the player clear options to play the game, look at how to play the game, and have the option to exit the game.

Pause menu:


PAUSE MENU
paused
current score
how to play
high score
quit
When the game is paused, the player can press the pause button again to resume the game.
Paused text to inform the player that the game is paused. The player's current score is displayed below the paused message.
This button allows the player to see the current high score to see how their own score compares.
This button allows the player to exit the game.

Playing the game:

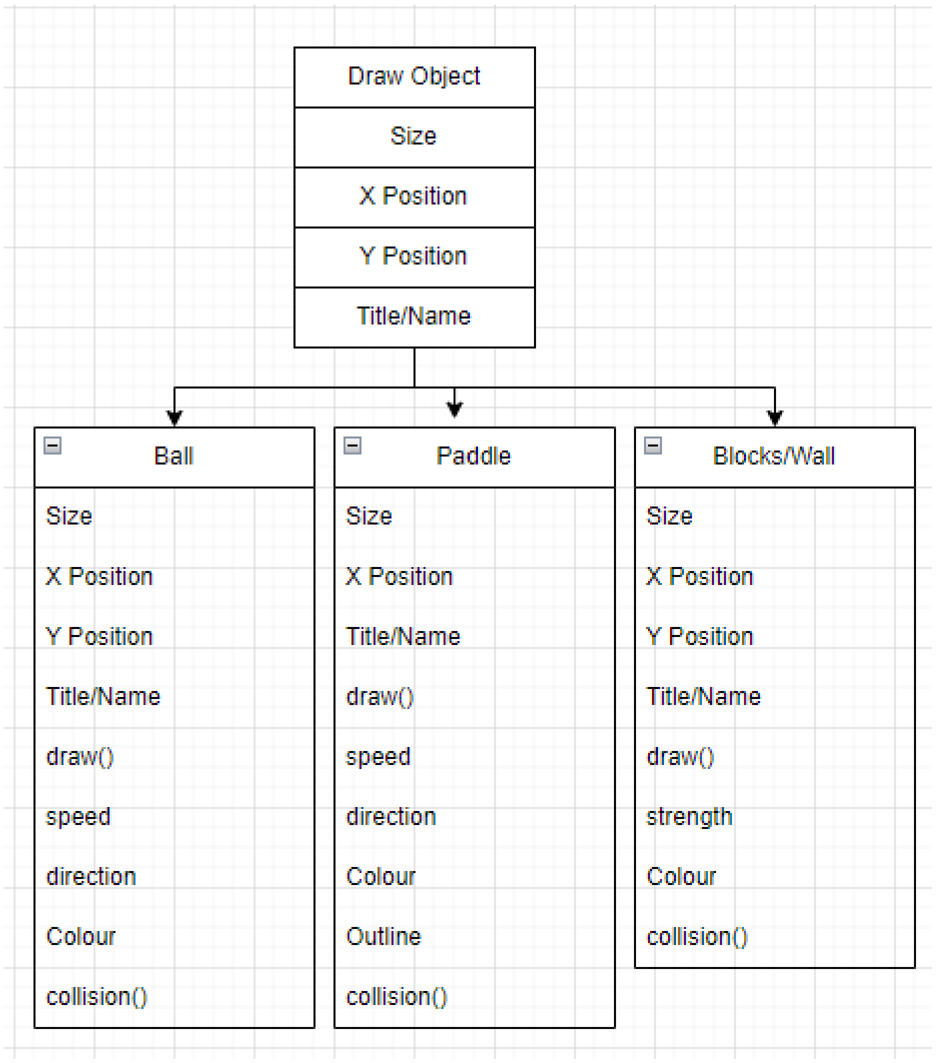
PLAYING THE GAME



The blocks are vibrant and distinct different colours so that the player knows what strengths the blocks are. They are also contrasting to the colour of the background. The ball is also a contrasting colour to the background so that the player can easily tell where the ball is.

The paddle is also a different colour to the background so that the player won't lose track of the paddle.

Classes



As you can see, I have clearly laid out the classes that I would like to use in my program in a simple class diagram which I can refer to when writing my code so that I know what methods and attributes I need to be using for each class in the program.

Variables Table

Method	Name	Data Type	Explanation	Justification	Validation
Main game loop					

Variable	screen_w, screen_h	Int	Dimensions of the screen size	Sets the height and the width of the screen when the game is running.	n/a
Variable	bg	Tuple	Background colour	Sets the colour of the background when playing the game	n/a
Variable	block_r, block_o, block_y	Tuple	Block colours	Sets the colour of the blocks that are used in the game	n/a
Variable	paddle_colour	Tuple	Paddle colour	Sets the colour of the paddle that the player uses in the game	n/a
Variable	paddle_outl	Tuple	Paddle outline	Sets the colour and dimensions of the paddle border so that it is clearly outlined	n/a
Variable	txt_colour	Tuple	Text colour	Sets the colour of the text that will appear when the player has options to do things and when the game is over	n/a
Variable	columns	Int	Number of columns in the block wall	Sets the number of columns of blocks that appear at the start of the game	n/a
Variable	rows	Int	Number of rows in the block wall	Sets the number of rows of blocks that appear in the wall at	n/a

				the start of the game	
Variable	clock	Clock	Pygame clock object	Used for controlling the frame rate of the game to 60fps	n/a
Variable	fps	Int	Frames per second	The number of frames per second that the game will run at	n/a
Variable	live_b	Boolean	Live ball	Determines if the ball is live and moving or not	n/a
Variable	game_ovr	Int	Game over value	Determines if the game is over or not	n/a
Method	draw_txt	method	Draws the text onto the screen	Draws the text onto the screen using pygame's draw function	n/a
Variable	screen	surface	Pygame surface object for the main game screen	Represents the main game screen	n/a
Variable	font	surface	Pygame surface object for the font of the text on the screen	Represents the font of the text on the main game screen	n/a
Method	player_pad.reset()	method	Resets the position of the paddle once the game is over	Resets the paddle back to its starting co-ordinates when the game is finished	n/a
Method	pygame.display.update()	method	Resets the screen when the game is over	Using pygame's display options, the screen resets when the game is finished	n/a

Method	player_pad.move()	method	Draws the paddle as it moves left and right	Draws the paddle in its new position when the player moves it left and right	n/a
Method	wall.draw_wall()	method	Draws the wall of blocks	Draws the wall of blocks that appears at the beginning of the game	n/a
Method	player_pad.draw()	method	Draws the paddle	Draws the paddle that appears at the bottom of the screen at the start of the game	n/a
Method	ball.draw()	method	Draws the ball	Draws the ball at the start of the game	n/a
Method	screen.fill(bg)	Pygame function	Sets the background colour	Sets the background to the colour that was stated at the beginning of the code	n/a
Method	clock.tick(fps)	clock	Sets the frame rate of the game	Sets the frame rate so that 60 frames pass every second	n/a
		Wall Class			
Class	wall	Class	Represents the wall of blocks	Encapsulates the properties and methods related to the wall of blocks	n/a
Method	draw_wall	Method	Draws the wall of blocks with their given strengths	Sets it so that each block is given a strength, so when the later function is called to draw the blocks, they	n/a

				will be set in strength	
Variable	Block_colour	tuple	Sets the colour of the blocks based on strength	Sets the blocks so that they have a colour based on their strength. The colours used are the ones stated in the main loop	n/a
Variable	Block_individ	array	Creates an empty list for an individual block	Sets each individual block to be an empty array, which will later be filled with strength and other variables	n/a
Variable	Block_row	array	Resets the block row to an empty array	Resets the block row to be an empty array that will be later filled with individual blocks and their assigned arrays	n/a
Variable	Block_x	float	X position for each block	Generates an x position for each block and creates a rectangle from it	n/a
Variable	Block_y	float	Y position for each block	Generates a y position for each block and creates a rectangle from it	n/a
Variable	strength	int	Assigns each block with a value for strength	Assigns each block with a strength number based on which row that block is in	n/a

Method	Block_row.append	method	Appends a block to the row	Appends an individual block to the block row once the strength has been assigned	n/a
Paddle Class					
Class	Paddle	Class	Represents a rectangular object	Encapsulates the properties and methods related to the paddle	n/a
Method	draw	method	Draws the paddle	Draws the paddle onto the screen using pygame's drawing function	n/a
Method	key	Method	Allows the user to press the arrow keys to move the paddle	Uses pygame's function to allow the player to move the paddle using the arrow keys which changes the direction of the paddle	Will only allow the player to move the paddle inside the screen's dimensions
Variable	direction	int	Tells the paddle which direction to go in	Signifies which direction the paddle will move depending on what value it holds after the arrow key has been pressed	Will only allow left and right as the directions and nothing else
Variable	speed	int	Holds the speed of the paddle	This variable contains the speed at which the paddle moves across the screen	n/a

Variable	Height/width	int	Dimensions of the paddle	Sets the dimensions of the paddle to large enough integer values	n/a
Ball Class					
Class	Ball	Class	Represents the ball as an object	Encapsulates the properties and methods related to the ball	n/a
Method	draw	method	Draws the ball onto the screen	Draws the ball using pygame's circle drawing function	n/a
Variable	ball_rad	int	Holds the balls radius value	Holds the integer value of the ball's radius to dictate the size of the ball	n/a
Variable	speed_max	int	Holds the maximum speed of the ball	Holds the integer for the maximum speed of the ball that it can travel across the screen	n/a
Variable	speed_x, speed_y	int	Holds the speeds of the ball going in the x and y direction	Holds the integer value of the speed the ball travels in the x and y direction	n/a
Variable	wall_destroyed	int	Let's the computer know if the wall has been destroyed or not	The value held in this variable indicates to the computer whether the wall has been destroyed or not	n/a
Variable	item_count	int	A count of how many blocks is in the row	This variable stores the number of blocks that are in the row selected	n/a

Variable	row_count	int	A count of how many rows there are	This variable stores how many rows of blocks there are	n/a
Method	move	method	Holds all the variables that cause the ball to move across the screen	This is a procedure that stores all the variables that cause the ball to move across the screen during the game	n/a
Method	collision	method	Checks if the ball has collided with anything	These variable checks if the ball has collided with anything including the paddle, blocks, and the walls	n/a
Variable	Score	int	Holds the score value	This variable holds the score value that the player earns as they progress in the game	Score can only be an integer and a maximum of 900.
			Menu System		
function	game_intro	n/a	This is what displays at the start of the game	This function returns all the actions needed for the start of the game including the main menu	n/a
function	start	n/a	This is what is carried out when the start button is clicked	This function returns all values when the start button is clicked and draws all the objects and starts the game.	If the player clicks the start button, game starts else nothing happens

function	quitgame	n/a	This is what is carried out when the quit button is clicked	This function returns everything that happens when the quit button is clicked – it exits the game	If player clicks the quit button, then the game closes else nothing happens.
function	button	n/a	This contains the code for the buttons	This function returns the values for the buttons displayed onto the screen including dimensions, colours and text	If plater clicks on button, then action is performed else nothing happens

Test Data

Throughout making the game, I will use validation therefore any of the data that is marked 'invalid' will not be accepted by the game.

High Score

Test Data	Type	Justification
Score > High Score	Valid	So, this means that the new highest score will be saved
Score < High Score	Invalid	This means that only the high score will be saved and not a lower score

Main Game Loop

Paddle Controls

Test Data	Type	Justification
Left Key Press (within bounds)	Valid	Simple controls that are easy to understand
Right Key Press (within bounds)	Valid	Simple controls that are also easy to understand
Right/Left Mouse click	Invalid	This is another control used to start the game.

Score

Test Data	Type	Justification
Current score > score	Valid	This allows the player's score to increase during the game

Current score < score	Invalid	This is just to make sure that the player's score wont decrease

Pause Button

Test data	Type	Justification
'P' button on keyboard	Valid	Simple and easy to remember
Any other button	Invalid	So the user won't accidentally pause the game in the middle of playing it

Pause Menu

High score

Test Data	Type	Justification
Score > High Score	Valid	So, this means that the new highest score will be saved
Score < High Score	Invalid	This means that only the high score will be saved and not a lower score

Resume button

Test data	Type	Justification
'P' button on keyboard	Valid	Simple and easy to remember as it is the same as the pause button
Any other button	Invalid	So the user will not accidentally resume the game when they are not ready

Main Menu

Play button

Test Data	Type	Justification
Left Mouse Click	Valid	To make it more easy and simpler for the user
Right Mouse Click	Invalid	To make sure the user hits the right button
Left Mouse Click (outside of the button)	Invalid	To make sure the user has hit the button to start the game

Exit Button

Test Data	Type	Justification
Left Mouse Click	Valid	To make it more simple and easier to understand

Right Mouse Click	Invalid	To make sure the user hits the right button and not miss-click
Left Mouse Click (outside of the button)	Invalid	To make sure the user has hit the button to exit the game

How to play button

Test Data	Type	Justification
Left Mouse Click	Valid	To make it less complicated for the user, so they only have to use one button for everything in the menu
Right Mouse Click	Invalid	To make sure the user hits the right button
Left Mouse Click (outside of the button)	Invalid	To make sure the user has actually hit the button to open up the page

High score

Test Data	Type	Justification
Score > High Score	Valid	So, this means that the new highest score will be saved
Score < High Score	Invalid	This means that only the high score will be saved and not a lower score

Acceptance Test Table

Throughout making the game, I will be using validation as stated earlier and therefore again, any data that has been marked 'invalid' will not be accepted by the game.

No.	What is being tested?	Inputs	Expected Outcome

1.	Updating the high score in the database.	Valid:
		Insertdata("Arlo," 69)
		Invalid:
		insertdata(9, 69)
			Valid:
			The high score will update in the database
			Invalid:
			The database will be updated, and an error will be printed out

2.	The score to be updated as the player destroys the blocks	Valid: Score = block strength *5 Invalid: The player either hits a wall or has died	Valid: The score to be updated on the screen reflecting the blocks that they have destroyed Invalid: The score to stop

			being updated and only showing the player's current/last score
3.	The score being displayed at the bottom-middle of the screen	n/a	The score to be towards the lower middle of the screen
4.	The high score being showed to the player when the game is over or in the main menu or the pause menu	n/a	The high score to be displayed to the player when the game is over, in the pause menu and in the main menu also
5.	The player to stay inside the game screen if not the game is over.	Valid: Player x position is within the game boundaries Invalid: Player x position is outside the game boundaries	Valid: The game to continue until the game is over Invalid: The game will run into a logic error and the paddle will get stuck.
6.	The button click function	Valid: Left click on the button Invalid: Right click on the button	Valid: Game will start Invalid: Nothing will happen
7.	Sound effect when the ball bounces	Valid: If the ball has collided with a wall or a block that won't be destroyed or the paddle	Valid: A sound effect will play after the bounce
8.	Sound effect when the ball breaks a block	Valid: If the ball has broken a block when collided	Valid: A sound effect should play when the block breaks
9.	Sound effect when the ball misses the paddle, and the game is over	Valid: If the ball has missed the paddle and game_ovr = 1	Valid: A sound effect should play after the ball hits the bottom of the game screen
10.	Exit button	Valid: Left click the exit button Invalid: Right click on the exit button	Valid: The game should shut down. Invalid: Nothing should happen at all
11.	Pause menu key pressed	Valid: The 'P' key is pressed	Valid: The pause menu will appear.

		Invalid: Any other letter key is pressed	Invalid: Nothing should happen
12.	Resume button	Valid: The ‘P’ key is pressed again Invalid: Any other letter key is pressed	Valid: The game should resume as normal. Invalid: Nothing should happen
13.	Main menu button	Valid: Left click on the main menu button Invalid: Right click on the main menu button	Valid: The player should return to the main menu. Invalid: Nothing should happen
14.	Checking if there is a new high score	Valid: High score < score Invalid: High score > score	Valid: The high score variable and the database will be updated. Invalid: The high score variable and the database won’t be updated
15.	Ball destroying a block in the wall	Valid: The ball collides with the block and destroys it Invalid: The ball does not destroy the block	Valid: If the ball hit the block and destroys it, it will disappear from the row. Invalid: The block stays in the row of blocks
16.	All the blocks have been destroyed	Valid: The rows of blocks have disappeared, and the game is over Invalid: Not all the blocks are gone, and the game is still running	Valid: game_ovr = 1 and the player is sent back to the main menu with their score Invalid: Nothing will happen until the game is over
17.	The ball missing the paddle and going off screen	Valid: the ball hits the bottom of the screen and the game stops Invalid: The game continues to run as normal	Valid: game_ovr = 1 and the player is sent back to the main menu Invalid: Nothing should happen as the game should still be running as usual

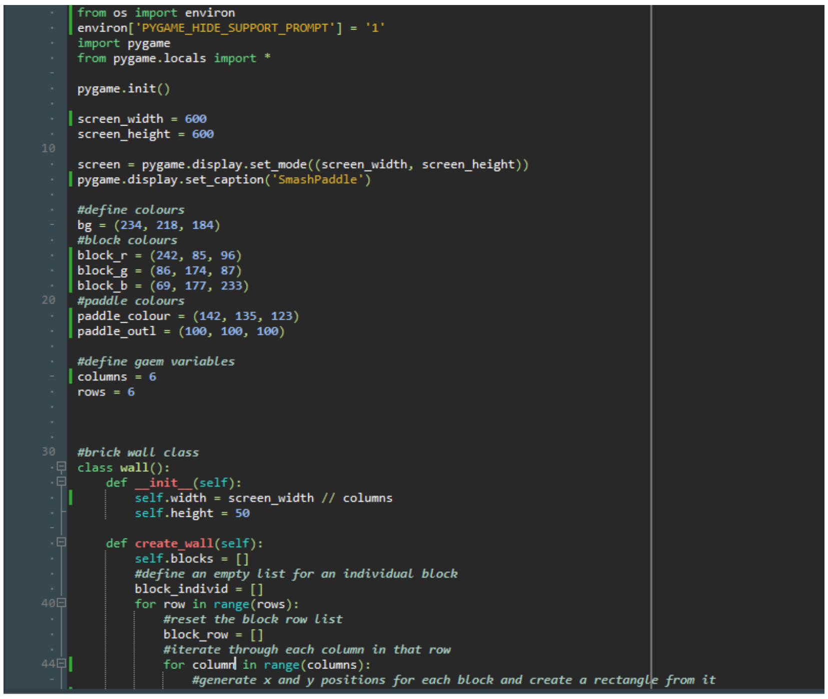
18.	How to play button	Valid: Left click on the button Invalid: Right click on the button	Valid: The player is taken to the how to play section where they are shown the controls Invalid: Nothing should happen

19.	Score will be saved to the database	Valid:
		InsertData("Arlo", 100)
		Invalid:
		InsertData(0)
			Valid:
			Score and name are entered into the database and shown when the table is viewed
			Invalid:
			Data is not inserted into the database.

Development:

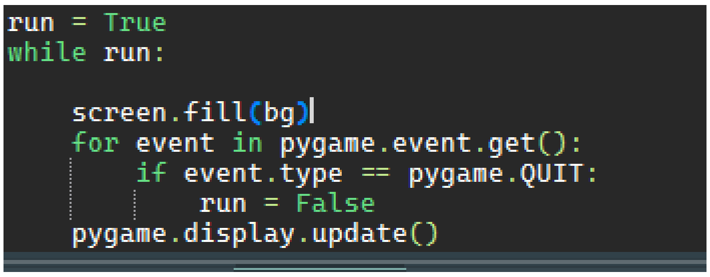
Prototype 1

Creating a screen and the wall class (12/06/24):



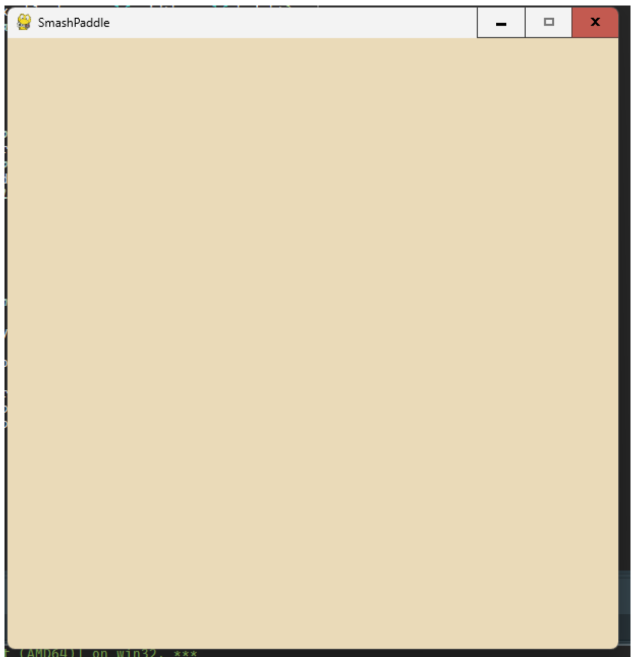
```python
from os import environ
environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1'
import pygame
from pygame.locals import *

pygame.init()

screen_width = 600
screen_height = 600

screen = pygame.display.set_mode((screen_width, screen_height))
pygame.display.set_caption('SmashPaddle')

#define colours
bg = (234, 218, 184)
#block colours
block_r = (242, 85, 96)
block_g = (86, 174, 87)
block_b = (69, 177, 233)
#paddle colours
paddle_colour = (142, 135, 123)
paddle_outl = (100, 100, 100)

#define gaem variables
columns = 6
rows = 6



#brick wall class
class wall():
    def __init__(self):
        self.width = screen_width // columns
        self.height = 50

    def create_wall(self):
        self.blocks = []
        #define an empty list for an individual block
        block_individ = []
        for row in range(rows):
            #reset the block row list
            block_row = []
            #iterate through each column in that row
            for column in range(columns):
                #generate x and y positions for each block and create a rectangle from it
```

I imported pygame and set up the screen by defining the background colour and the dimensions of the screen. This updated the screen. I then defined the wall class afterwards which included some principal attributes and methods. The background dimensions were set to a smaller size as once more blocks have been cleared; the game screen starts to get very empty towards the middle.



```
run = True
while run:

    screen.fill(bg)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
    pygame.display.update()
```

Sets up the main game loop and creates the background by filling the screen with the background colour.



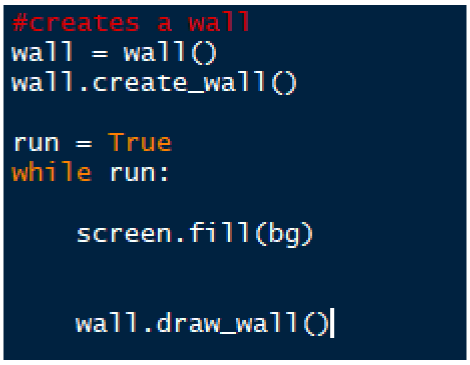
This is the background colour that will be set for the rest of the game. It is a basic colour that will be distinctly different from the colour of the blocks and everything else so that the player will be able to see everything more clearly whilst playing the game.

Testing:

What is being tested?	Inputs	Justification	Expected Outcome	Actual Outcome
The screen dimensions change	Width – 600 Height - 600	Increases the size of the screen so that the player can see the whole wall of blocks	The screen dimensions increase to the size inputted	The screen adjusted to the size and because the blocks have not been fully finished, the screen was filled with the background colour only

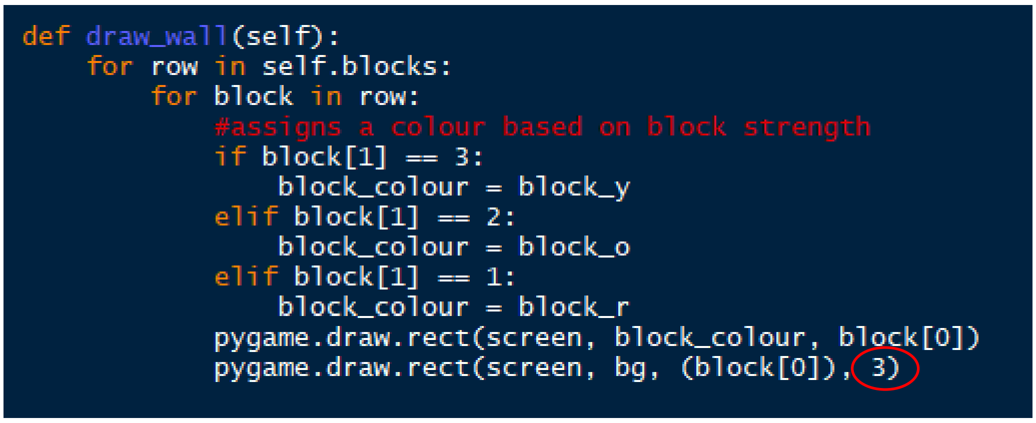
Changing the colour of the screen	(234, 218, 184)	So that the screen is the desired colour from the criteria, so it is a different colour to everything else.	The screen to be a light raffia colour.	The screen was light raffia.

15/06/24:



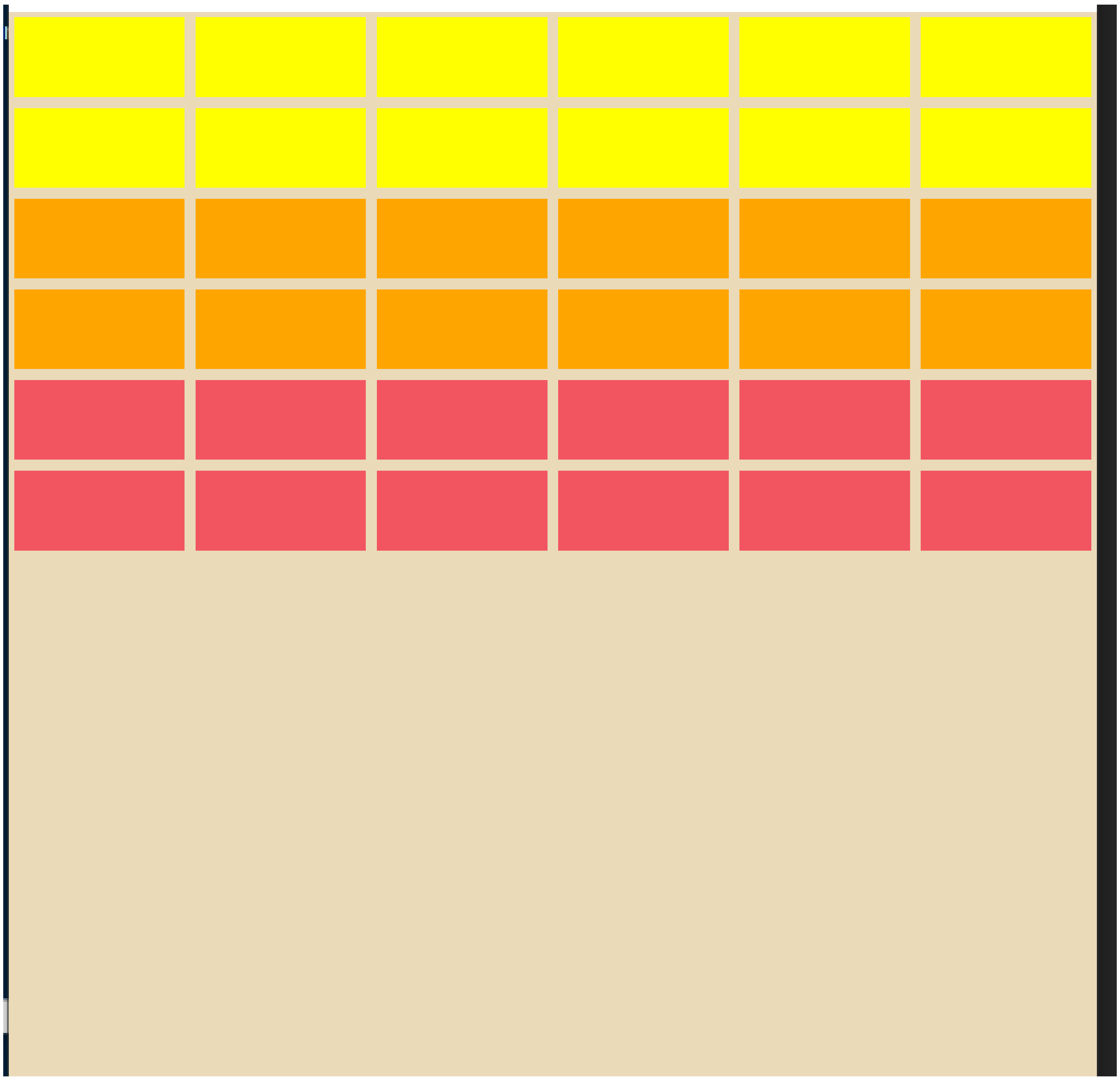
```
#creates a wall
wall = wall()
wall.create_wall()

run = True
while run:

    screen.fill(bg)


    wall.draw_wall()
```

Here I have added a few lines of code that draw the wall onto the screen. The blocks will be sorted into rows and columns to form a wall where the bottom row will appear as the colour red and have a strength value of 1, the middle row will appear as the colour orange and will have a strength value of 2, and finally the top row will appear as the colour yellow and will have a strength value of 3. The strength value corresponds/represents the number of times the player needs to hit the block with the ball before it is destroyed.



```
def draw_wall(self):
    for row in self.blocks:
        for block in row:
            #assigns a colour based on block strength
            if block[1] == 3:
                block_colour = block_y
            elif block[1] == 2:
                block_colour = block_o
            elif block[1] == 1:
                block_colour = block_r
            pygame.draw.rect(screen, block_colour, block[0])
            pygame.draw.rect(screen, bg, (block[0]), 3)
```

This code is the draw function for the wall that assigns the colour of the blocks based on their strength assigned in the previous function. The area circled above represents the value of the line spacing in between blocks so that it is clear what size the blocks are and the colour of them. The higher that value is, the more spaced out the blocks are/the smaller the blocks become.



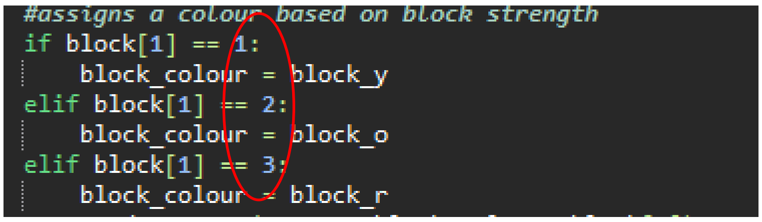
Test Data	Type	Justification	Actual Test/Result
Positive integer that is preferred to be below 5	Valid	Makes small spaces in between blocks that are not too large so that the blocks are too small	Valid
Negative integer	Invalid	This would remove the spacing in between the blocks and results in three rows of colours	Invalid
Anything else e.g., float, string	Invalid	This would create an error	Invalid

Draw Wall Procedure:

```
Procedure draw_wall:
    For each row in self.blocks:
        For each block in row:
            If block strength is 3:
                Set block_colour to yellow
            Else if block strength is 2:
                Set block_colour to orange
            Else if block strength is 1:
                Set block_colour to red

            Draw a filled rectangle on the screen:
                Use block_colour
                Draw the rectangle at block [0] (block position and size)

            Draw a rectangle outline on the screen:
                Use background color bg
                Draw the outline at block [0] with a thickness of 3
endprocedure
```

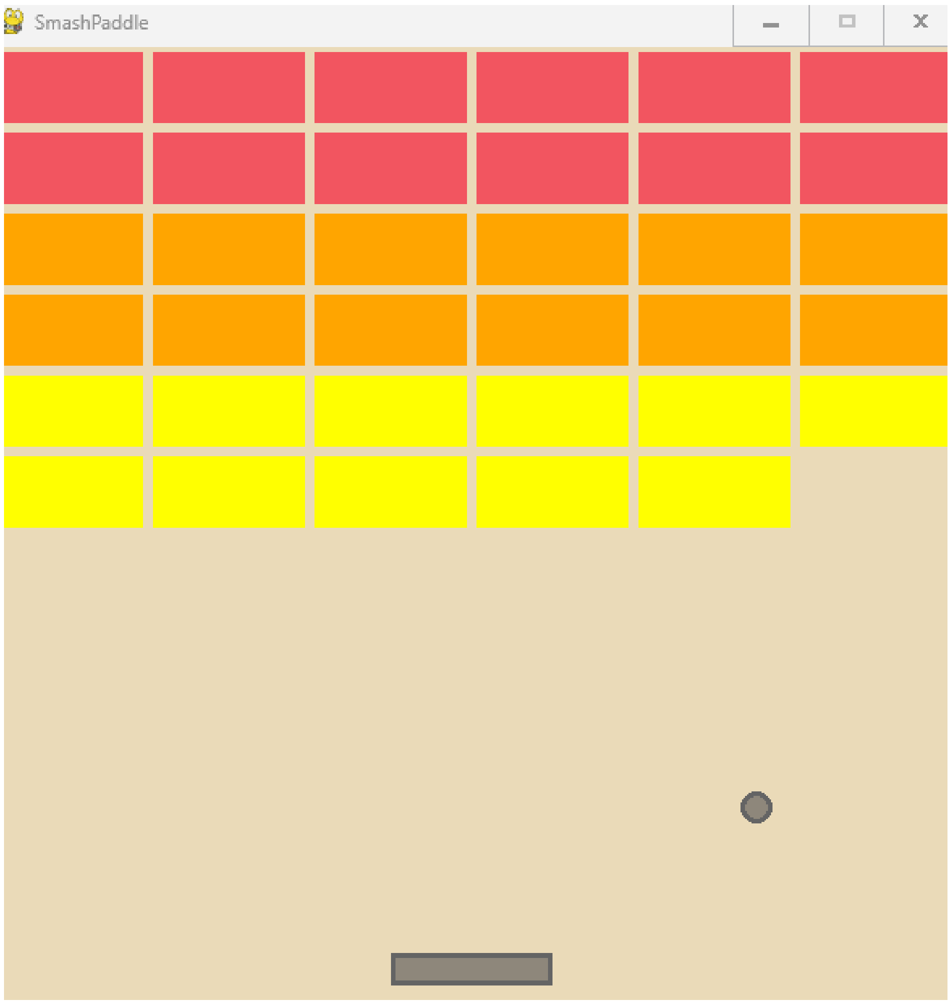
As you can see, I have written the pseudocode for the draw procedure.

Error:

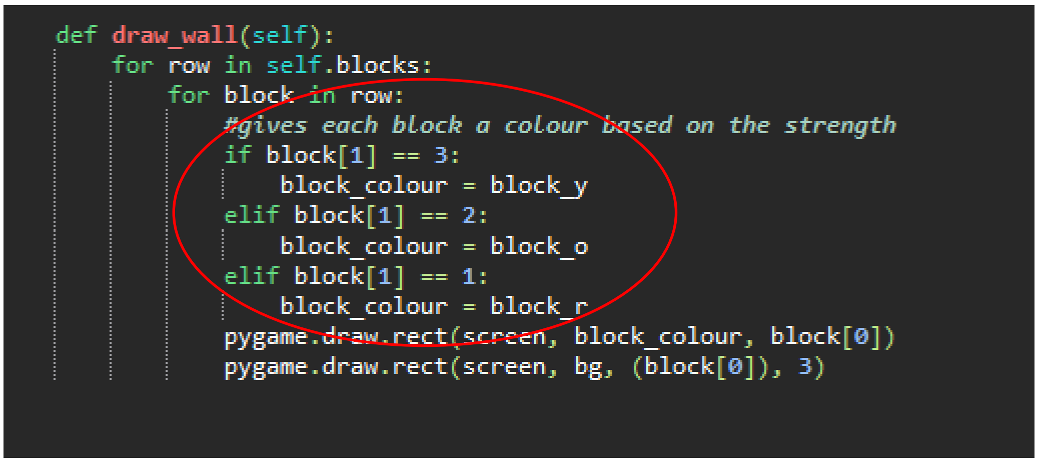


```
#assigns a colour based on block strength
if block[1] == 1:
    block_colour = block_y
elif block[1] == 2:
    block_colour = block_o
elif block[1] == 3:
    block_colour = block_r
```

Here I have assigned the colours the wrong way around as this state that yellow is the weakest colour and that red blocks are the strongest when the yellow blocks should be the strongest and the red blocks should be the weakest. The screenshot below displays what it looks like with this error.

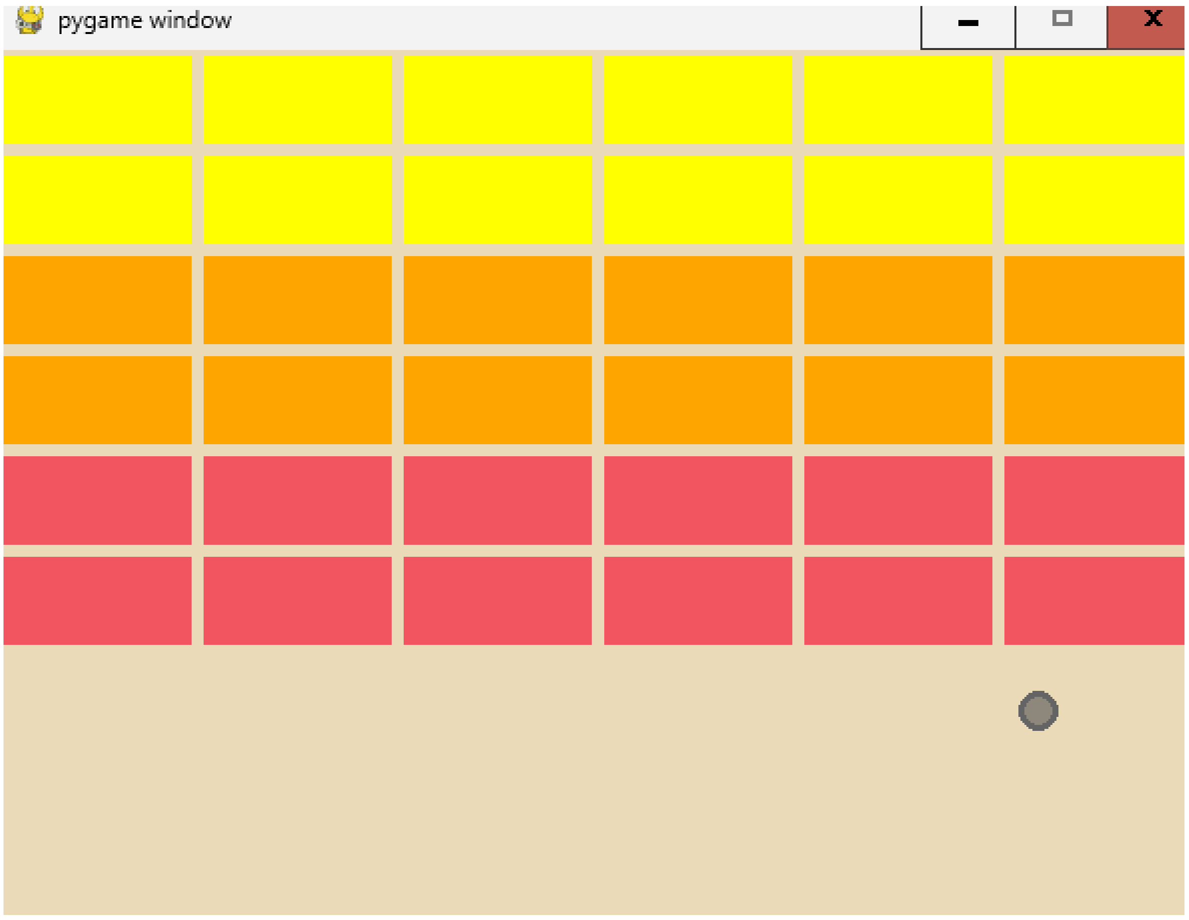


As you can see, the colour scheme for the block is the wrong way around as the red blocks should be on the bottom as they should have the lowest strength, however with the error, they are on the top two rows with the highest strengths instead of the yellow blocks. To fix this, I have rewritten the code so that the colour correctly corresponds with their designated colours. The code below is what I have created as the fix for this error.



```
def draw_wall(self):
    for row in self.blocks:
        for block in row:
            #gives each block a colour based on the strength
            if block[1] == 3:
                block_colour = block_y
            elif block[1] == 2:
                block_colour = block_o
            elif block[1] == 1:
                block_colour = block_r
            pygame.draw.rect(screen, block_colour, block[0])
            pygame.draw.rect(screen, bg, (block[0]), 3)
```

I have changed the code to this as this code allows the right colour system to be displayed when the player is playing the game. This fix was crucial to the game as the blocks need to be the right way around for the game to work as expected and so that the game is not too difficult to begin with where the highest strength blocks are at the bottom. The screenshot below shows the fixed and final version of the wall of blocks and how they should look.



Prototype 1 Review:
Success Criteria:

Requirements	Justification	☑	Reference
Colour Scheme for blocks	This allows the player to see the distinct blocks and tell how strong they are	☑	Stakeholder interview 1/Feature of “Atari: Breakout”
Simple retro-styled scheme	This makes the game look simple but appealing to all ages	☑	Stakeholder/Feature of “Atari: Breakout”
Several rows of blocks to appear at the top of the screen	This allows the player to have something to destroy and play the game	☑	Stakeholder/Feature of “Atari: Breakout.”

I successfully achieved all three of the requirements set for prototype 1 at the beginning of this project however with the rows of blocks, my client stated in the first interview that they wanted 3 rows of blocks that contained 5 each. I have ended up going beyond that and have created 6 rows of 6 blocks. I have managed to get all those blocks and the background to show despite running into a few errors concerning the strength and colour of those blocks. This accomplishment means that the player will have something to destroy and a main objective to complete to beat the game.

Secondly, I have also stuck to maintaining the retro style of the game to ensure that it is clean but also simple to use and appealing to all ages that play the game to allow accessibility. This design choice overall improves the game as it makes every aspect of the game look well-kept but also distinct and easy to see. This will also improve immersion.

Lastly the final requirement, that each row of blocks would be a different colour, was also met. This is crucial for the game as it allows the player to tell the difference between the strengths of the blocks but also so that they do not confuse the colour or size of the blocks in the rows.

Prototype 1 Interview Plan:

- How many frames per second should the game run at?
- What dimensions should the game screen have?
- Do you think that the blocks should be larger in any way?
- Do you think that the colour scheme of the blocks is too colourful?
- Does the retro style suit the game?

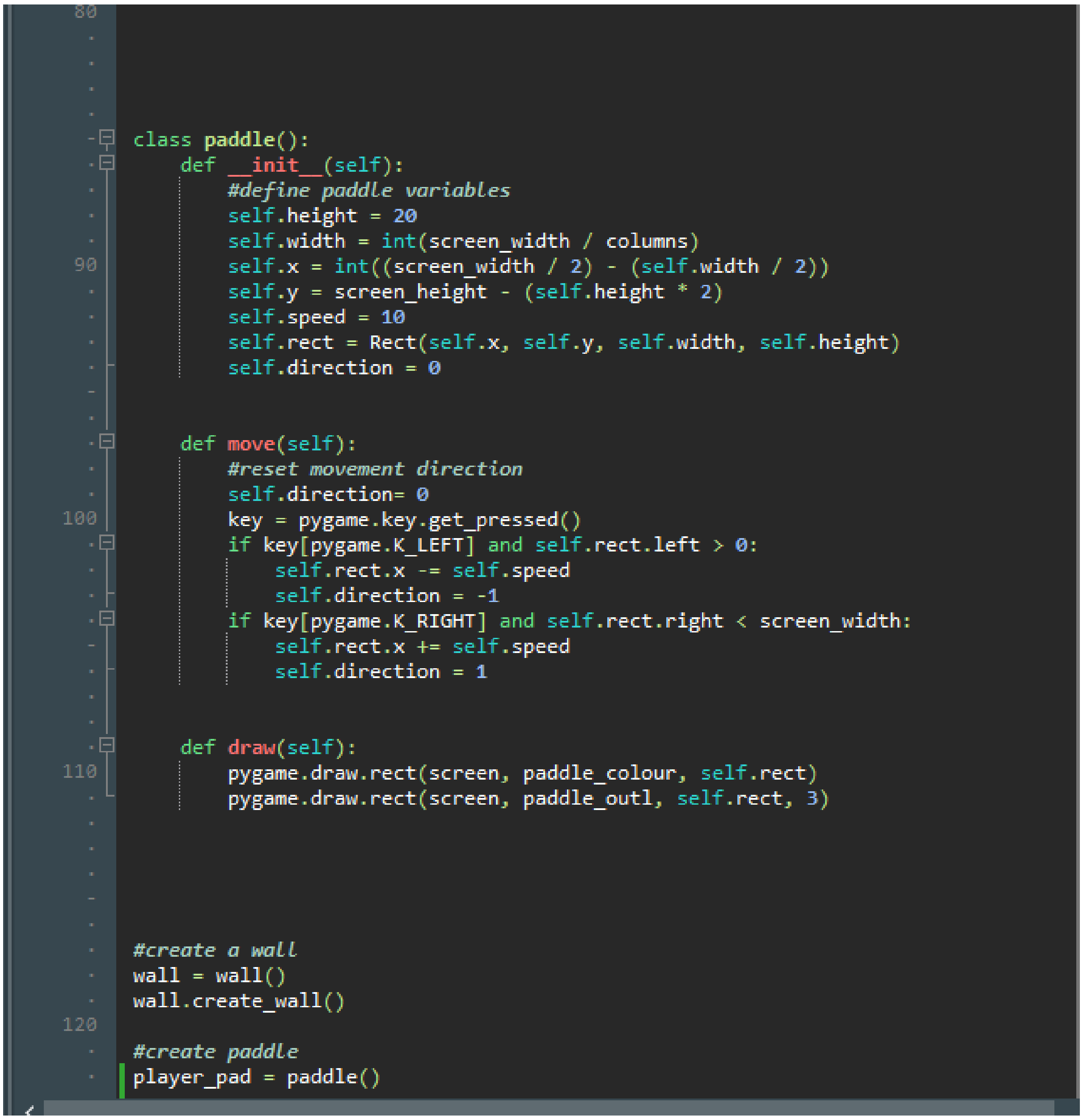
Prototype 1 Interview with Finlay Baker:

- How many frames per second should the game run at?
  "60fps is fine."
- What dimensions should the game screen have?
  "I think that 500 or 600 squared is fine as it shouldn't be too large."
- Do you think that the blocks should be larger in any way?
  "I think that they are large enough that the player can see them well."
- Do you think that the colour scheme of the blocks is too colourful?
  "I think that the colour scheme is just right as the player can easily tell the difference between the colours of the blocks."
- Does the retro style suit the game?
  "I think that the retro-style was a good choice as the game looks simple yet fun."

Conclusion:
The game will run at 60 frames per second, and the screen will be a square of six hundred pixels each length. This also means that the game will be windowed and not a full screen game. The colour scheme shall remain bright and colourful so that the player can easily tell what everything is, and the retro style will also be kept making the game look tidy yet fun and interesting.

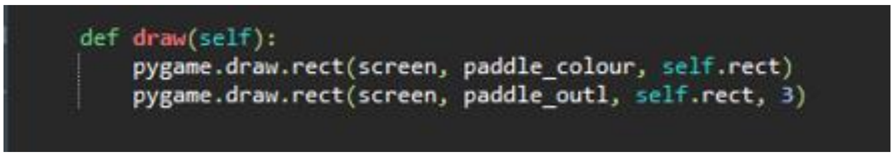
Prototype 2

Adding a paddle to the screen (19/06/24):



```
class paddle():
    def __init__(self):
        #define paddle variables
        self.height = 20
        self.width = int(screen_width / columns)
        self.x = int((screen_width / 2) - (self.width / 2))
        self.y = screen_height - (self.height * 2)
        self.speed = 10
        self.rect = Rect(self.x, self.y, self.width, self.height)
        self.direction = 0


    def move(self):
        #reset movement direction
        self.direction= 0
        key = pygame.key.get_pressed()
        if key[pygame.K_LEFT] and self.rect.left > 0:
            self.rect.x -= self.speed
            self.direction = -1
        if key[pygame.K_RIGHT] and self.rect.right < screen_width:
            self.rect.x += self.speed
            self.direction = 1


    def draw(self):
        pygame.draw.rect(screen, paddle_colour, self.rect)
        pygame.draw.rect(screen, paddle_outl, self.rect, 3)





#create a wall
wall = wall()
wall.create_wall()

#create paddle
player_pad = paddle()
```

I created the paddle class containing the draw function, and the move function. The move function recognises if the player has pressed either of the horizontal arrow keys and this will allow the paddle to move in the direction that the key was pressed.

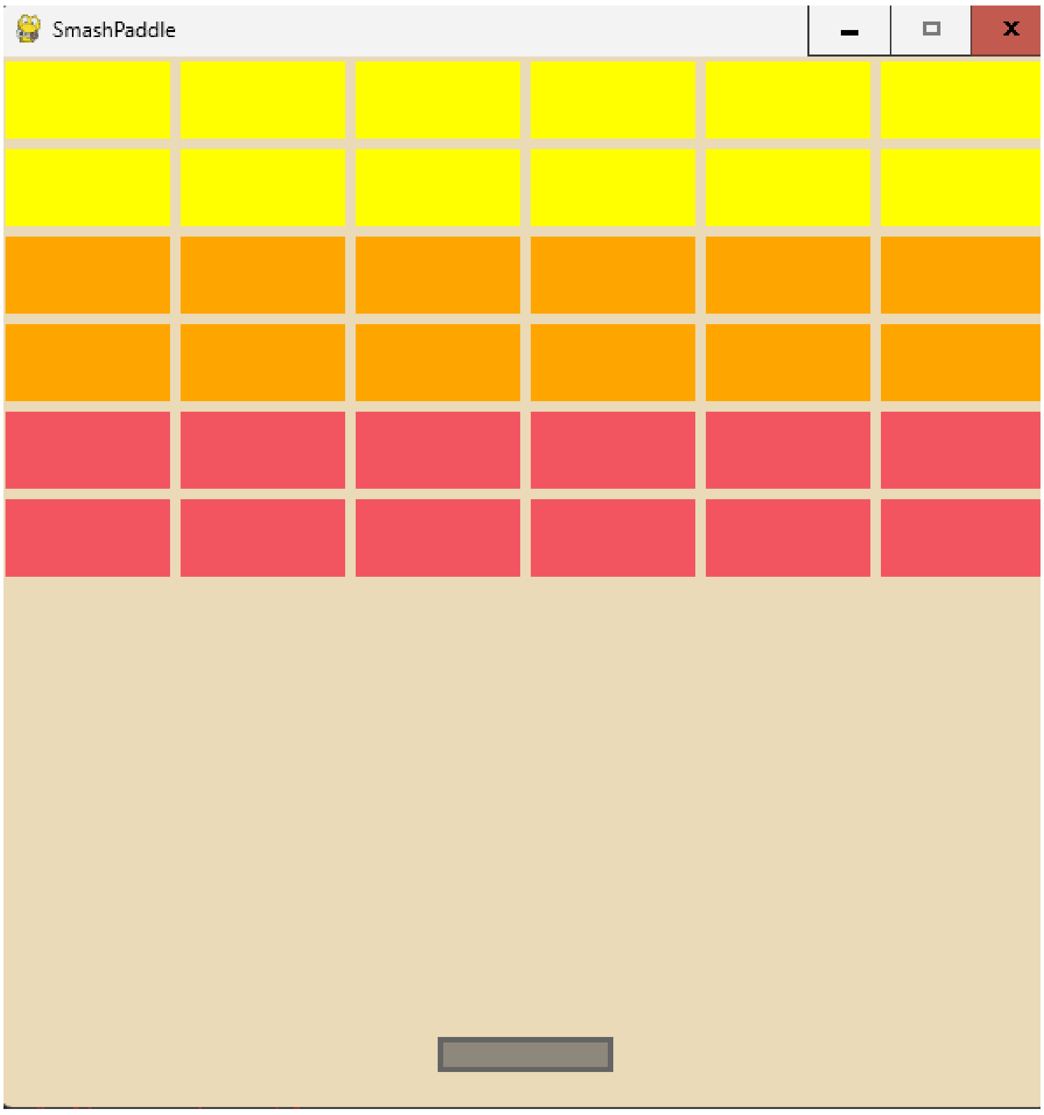


```
def draw(self):
    pygame.draw.rect(screen, paddle_colour, self.rect)
    pygame.draw.rect(screen, paddle_outl, self.rect, 3)
```

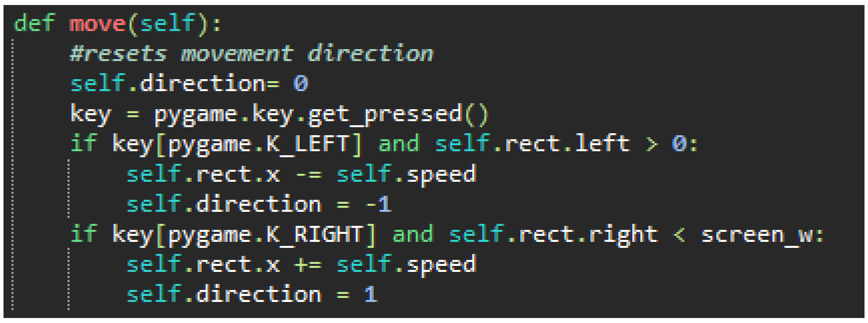
This function draws the outline of the paddle which will be a slightly darker colour than the main colour of the block. This will set the dimensions of the paddle and allow the player to tell where the paddle is and where the ball comes into contact with the paddle.

Testing:

What is being tested?	**Input**	**Justification**	**Expected Outcome**	**Actual Outcome**
Arrow key inputs	Left and Right arrow keys pressed	Moves the paddle left and right to control movement	The paddle moves left and right across the bottom of the screen	The paddle does as expected and moves left and right across the screen.
Paddle colour changes	Different RGB number to original paddle RGB number	To change the colour of the paddle	The paddle changes to the new RGB number's colours	The paddle changes colour.
Paddle outline colour changes	Different RGB number to original outline RGB number	To change the colour of the outline of the block	The paddle's outline changes colour to the new RGB number.	The paddle's outline changes colour.

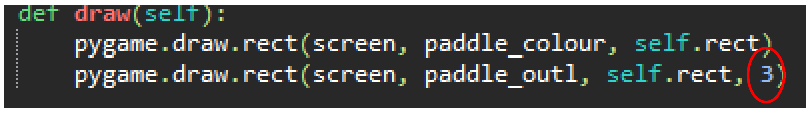


This is the output after creating the paddle class. There are no movement options yet, but they will be added next. The paddle is drawn as a small grey rectangle with a clear outline that makes the block easy to see against the background of the game screen.



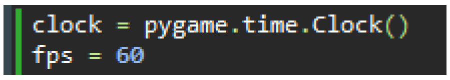
```
def move(self):
    #resets movement direction
    self.direction= 0
    key = pygame.key.get_pressed()
    if key[pygame.K_LEFT] and self.rect.left > 0:
        self.rect.x -= self.speed
        self.direction = -1
    if key[pygame.K_RIGHT] and self.rect.right < screen_w:
        self.rect.x += self.speed
        self.direction = 1
```

This is the move function that allows the paddle to move across the bottom of the screen when the player presses the left and right arrow keys. It uses validation to make sure that it will only recognise what the actual controls are and will not recognise it if the player presses any other key which is good because it means the game won't pick up accidental random inputs from the player which could result in their game being ruined.



```
def draw(self):
    pygame.draw.rect(screen, paddle_colour, self.rect)
    pygame.draw.rect(screen, paddle_outl, self.rect, 3)
```

As previously mentioned, this is the draw function for the paddle. The circled integer correlates to the thickness of the outline of the paddle.



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

The fps of the game has also been set to 60 frames per second so that the game can run smoothly and efficiently so it looks clean when the player starts the game.

Test Data	Type	Justification	Actual Test/Result
Positive integer that is preferred to be between 1 and 3	Valid	Makes an outline around the paddle so that it is clearer to see against the background of the game screen.	Valid
Negative integer	Invalid	This would remove the outline as a whole and the paddle would become a grey block with no clear outline.	Valid – However it is clearly not what I would like for the game but the game functions with this value regardless.
Anything that is not a number e.g., string	Invalid	This would create an error	Invalid

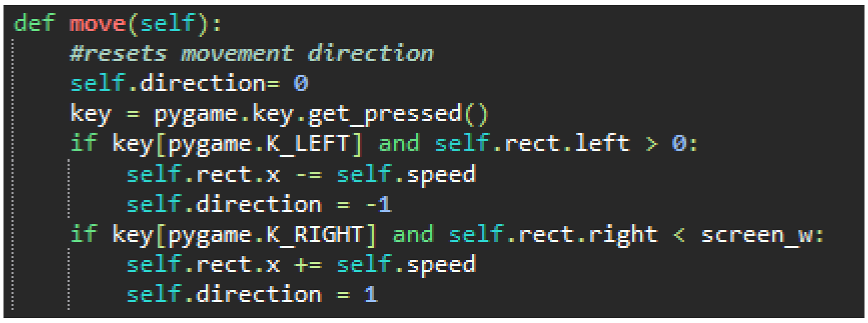
Move Paddle Function:

This is the pseudocode for the move function in the paddle class. It allows the paddle to move left and right across the screen.

```
function move()
   // Reset movement direction
   set direction to 0

   // Get currently pressed keys
   key = get_pressed_keys()

   // Check if LEFT key is pressed and character is not at the left screen edge
   if LEFT key is pressed AND character's left side is greater than 0 then
      // Move the character to the left
      move character's position by subtracting speed from x-coordinate
      set direction to -1
   endif

   // Check if RIGHT key is pressed and character is not at the right screen edge
   if RIGHT key is pressed AND character's right side is less than screen width then
      // Move the character to the right
      Move character's position by adding speed to x-coordinate
      set direction to 1
   endif
endfunction
```

<u>Validation:</u>

The paddle is controlled by the player by pressing the left and right arrow keys to move it. I will use validation to make sure that the paddle is unable to move outside of the screen's borders. The screenshot of the code below shows how I have done this.

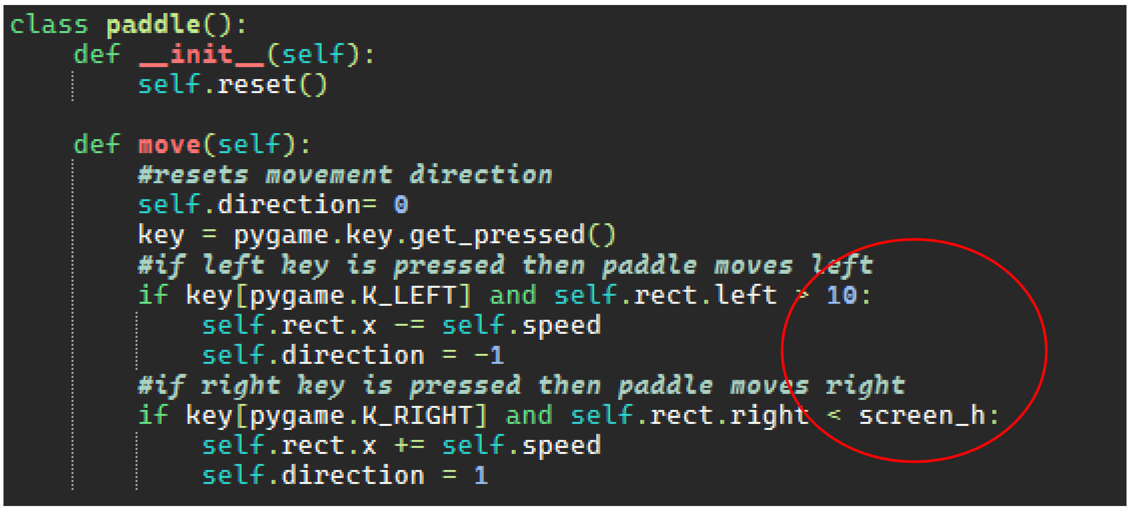


```
def move(self):
    #resets movement direction
    self.direction= 0
    key = pygame.key.get_pressed()
    if key[pygame.K_LEFT] and self.rect.left > 0:
        self.rect.x -= self.speed
        self.direction = -1
    if key[pygame.K_RIGHT] and self.rect.right < screen_w:
        self.rect.x += self.speed
        self.direction = 1
```

This code makes sure that the paddle won't go beyond the screen's dimensions and therefore there is no chance of an error occurring.

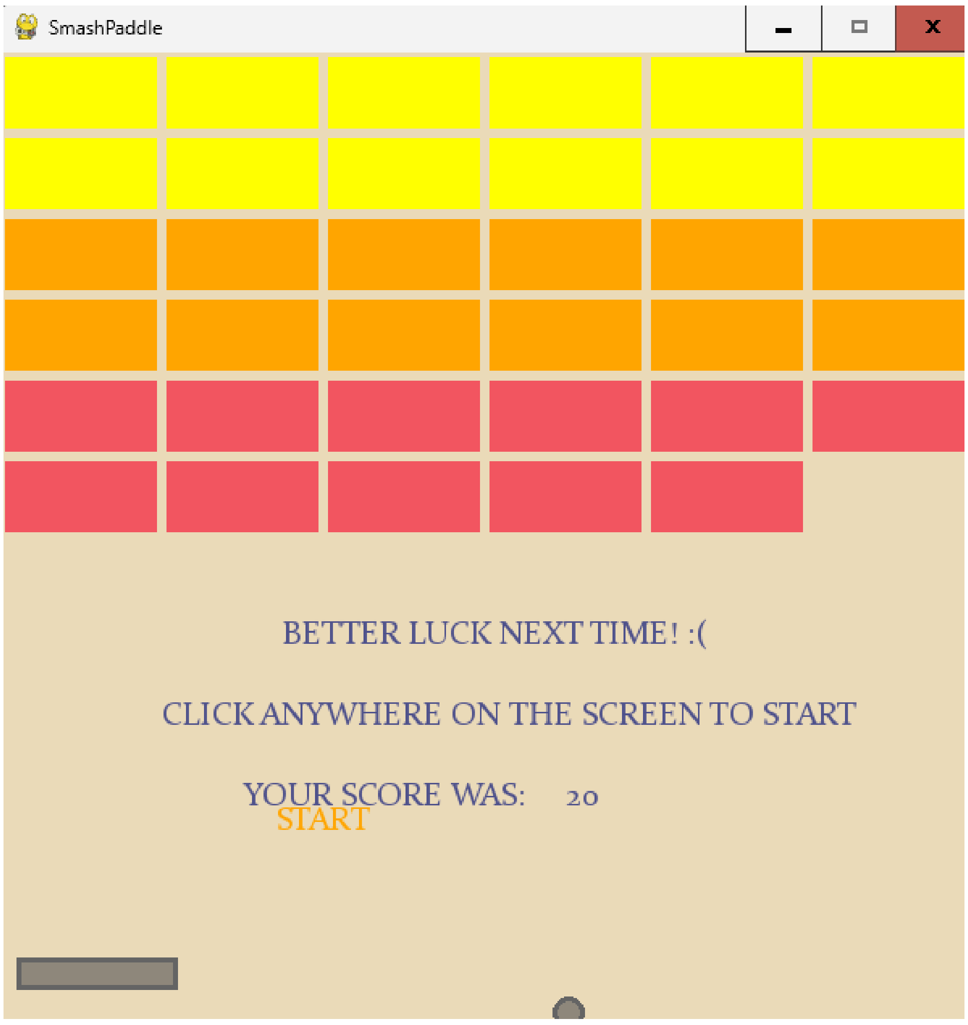
Error:

When creating the code for the paddle validation, I came across a simple syntax error where the bounds for the movement of the paddle were wrong. The screenshot below shows the code, and the red circle indicates the code that needed fixing.



```
class paddle():
    def __init__(self):
        self.reset()

    def move(self):
        #resets movement direction
        self.direction= 0
        key = pygame.key.get_pressed()
        #if left key is pressed then paddle moves left
        if key[pygame.K_LEFT] and self.rect.left > 10:
            self.rect.x -= self.speed
            self.direction = -1
        #if right key is pressed then paddle moves right
        if key[pygame.K_RIGHT] and self.rect.right < screen_h:
            self.rect.x += self.speed
            self.direction = 1
```

As you can see, the left arrow key bounds are written wrong as it should be ten and not zero. This meant that when I ran the code and moved left with the paddle as far as I could, it would not go all the way to the left. The second part of the error was a simple syntax error too as it should be screen_w however this ended up not effecting the game as the screen width and height are equal. The screenshot below shows the paddle not being able to move all the way to the left when playing the game.

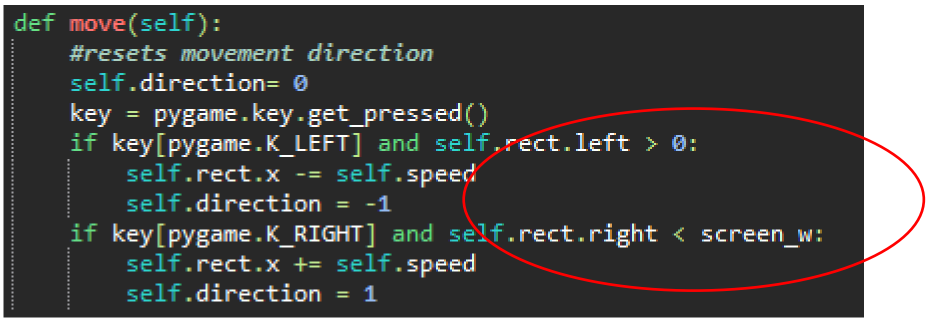

SmashPaddle

BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

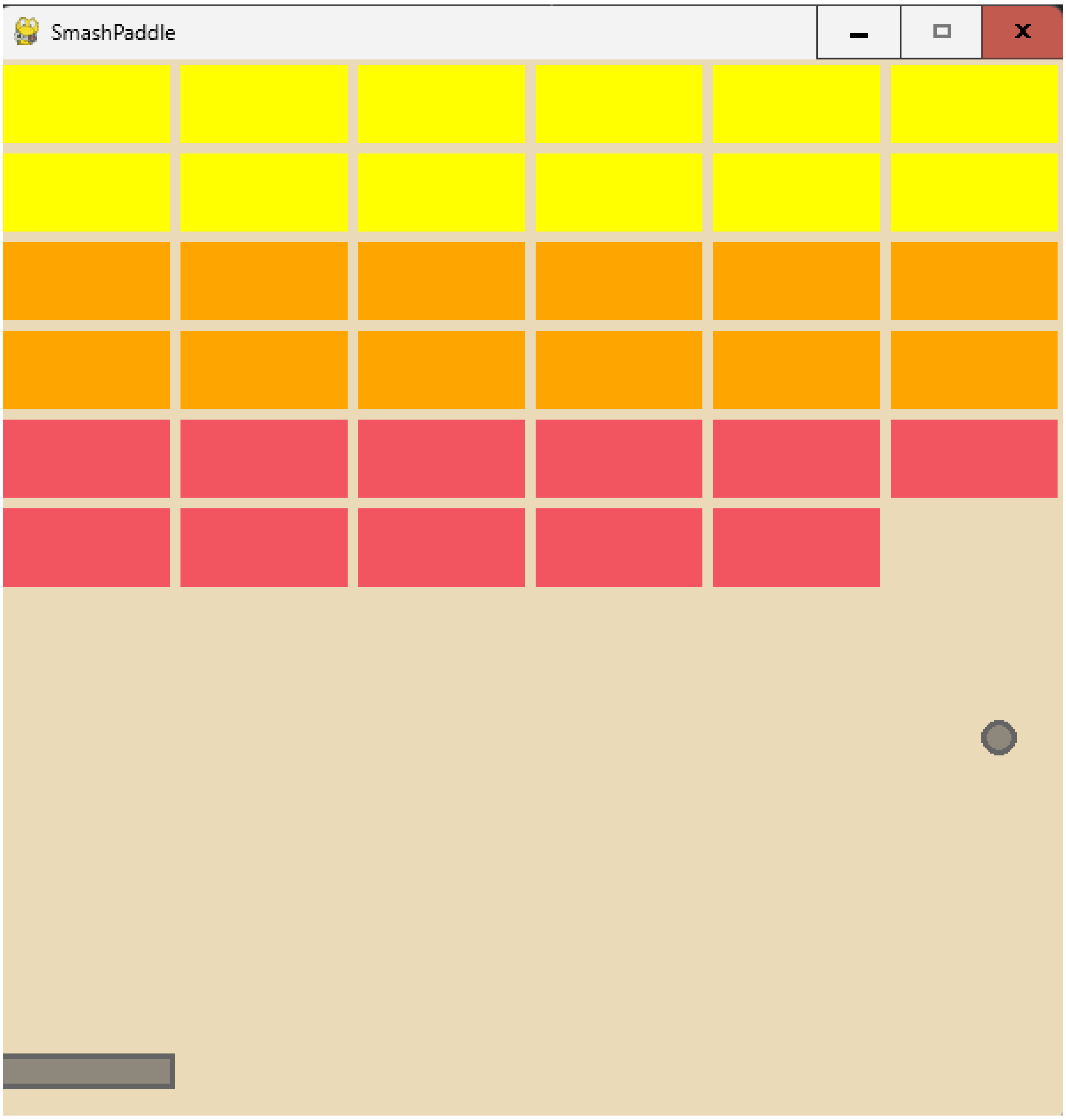
YOUR SCORE WAS: 20

As you can see, the paddle does not move all the way to the left of the screen as it should do. The code below shows the fix for this error. I have fixed the syntax errors and made 10 equal to zero, and screen_w instead of screen_h. I changed this code because it now allows the paddle to move freely within the correct bounds and therefore allows the player to have the full extent of paddle movement that they need to play the game.



```
def move(self):
    #resets movement direction
    self.direction= 0
    key = pygame.key.get_pressed()
    if key[pygame.K_LEFT] and self.rect.left > 0:
        self.rect.x -= self.speed
        self.direction = -1
    if key[pygame.K_RIGHT] and self.rect.right < screen_w:
        self.rect.x += self.speed
        self.direction = 1
```

The screenshot below shows the paddle moving correctly and the validation working as expecting.



Prototype 2 Review:
Success Criteria:

Requirements	**Justification**	☑	**Reference**
Colour set for the paddle	This allows the player to see the paddle clearly against the background of the game screen	☑	Stakeholder interview 1/Feature of "Atari: Breakout"
Clear outline around paddle	This again makes the paddle clear to see against the background of the game screen	☑	Stakeholder /Feature of "Atari: Breakout"

Paddle to be able to move left and right across the screen when the player presses the corresponding arrow keys	This allows the player to have something to move and hit the ball with to destroy the blocks and win the game.	☑	Stakeholder/Feature of “Atari: Breakout.”

I successfully achieved all three of the requirements set for prototype 2 at the beginning of this project however with the paddle controls, I went with the use of arrow keys instead of the mouse idea as this would make it accessible to even more people as the controls would be easier to use. I have managed to get the paddle to be drawn in the right place on the screen at the start of the game. This accomplishment means that the player will have something to move around and use to destroy the wall along with the ball when that comes into play.

Secondly, I have also accomplished in making the paddle an easy to see colour that is contrasting to the game screen background. This allows the player to easily see the paddle and know what the size of the paddle is so that they can anticipate where they need to move the paddle to when the ball comes into play.

Lastly the final requirement, that the paddle would be drawn with a clear and distinct outline, was also met. This again allows the player to clearly see where the paddle is at all times and what dimensions the paddle is. It also allows the paddle to look cleaner yet simpler too. This adds to the immersion in the game.

Prototype 2 Interview Plan:

- What size should the paddle be?
- Where should the paddle be placed at the start of the game?
- Do you think that the paddle should be a different shape in any way?
- Do you think that the colour of the paddle is suitable?
- Is the outline something that should be kept in the final prototype?

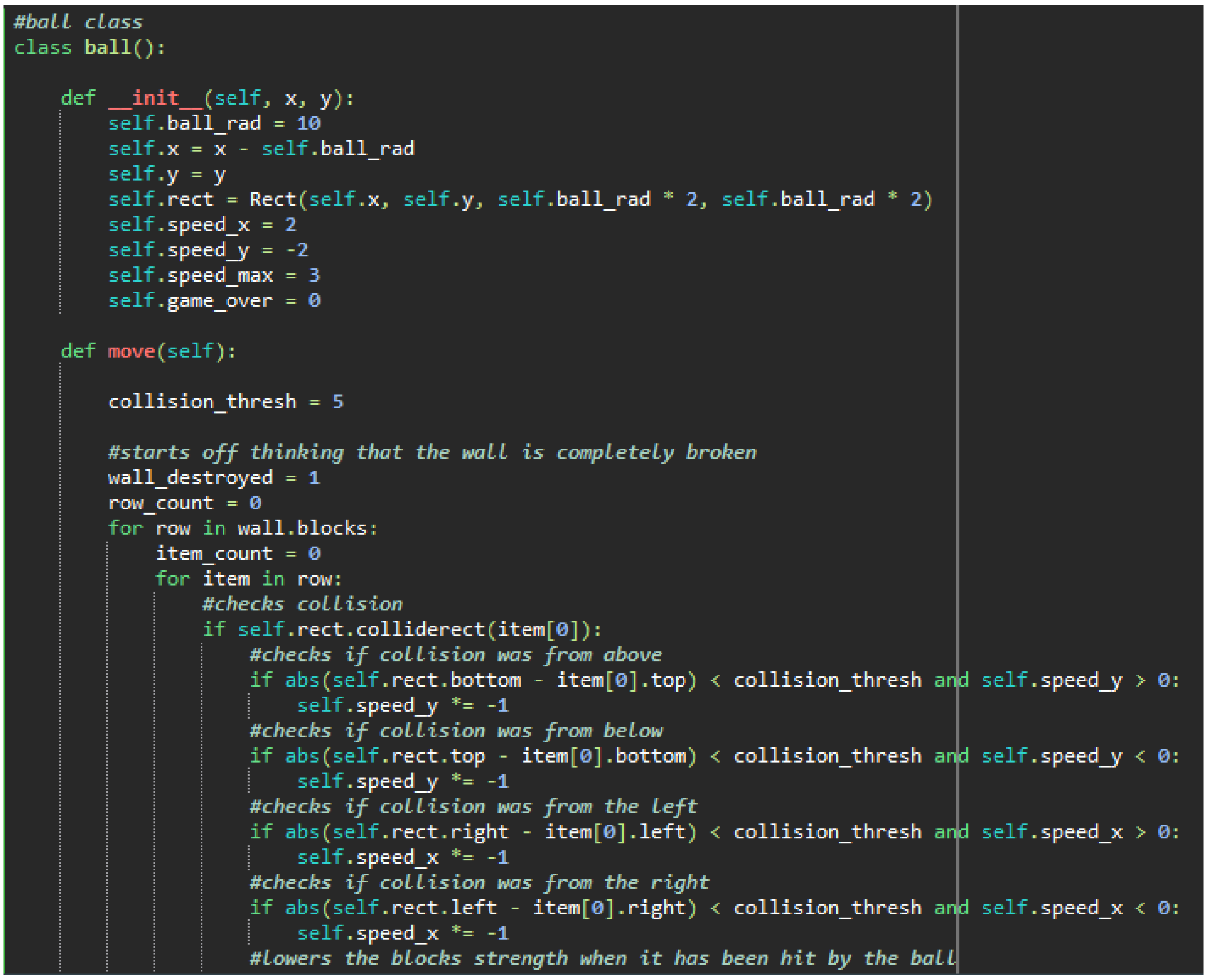
Prototype 2 Interview with Finlay Baker:

- What size should the paddle be?
  “Relatively small.”
- Where should the paddle be placed at the start of the game?
  “In the middle towards the bottom of the screen.”
- Do you think that the paddle should be a different shape in any way?
  “No, a long rectangular shape is fine”.
- Do you think that the colour of the paddle is suitable?
  “Yes, because it contrasts with the background.”
- Is the outline something that should be kept in the final prototype?
  “Yes, it should be kept because it makes it easier to see where the paddle is.”

Conclusion:
The game will start with a grey paddle with a black outline being drawn at the bottom of the screen centralised. It will be able to move left and right using the arrow keys. It will be a short but wide rectangular shape and will be clear to see against the background of the game.

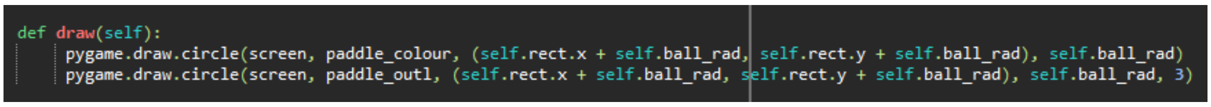
Prototype 3

Creating the ball class (21/06/24):



```python
#ball class
class ball():

    def __init__(self, x, y):
        self.ball_rad = 10
        self.x = x - self.ball_rad
        self.y = y
        self.rect = Rect(self.x, self.y, self.ball_rad * 2, self.ball_rad * 2)
        self.speed_x = 2
        self.speed_y = -2
        self.speed_max = 3
        self.game_over = 0

    def move(self):

        collision_thresh = 5

        #starts off thinking that the wall is completely broken
        wall_destroyed = 1
        row_count = 0
        for row in wall.blocks:
            item_count = 0
            for item in row:
                #checks collision
                if self.rect.colliderect(item[0]):
                    #checks if collision was from above
                    if abs(self.rect.bottom - item[0].top) < collision_thresh and self.speed_y > 0:
                        self.speed_y *= -1
                    #checks if collision was from below
                    if abs(self.rect.top - item[0].bottom) < collision_thresh and self.speed_y < 0:
                        self.speed_y *= -1
                    #checks if collision was from the left
                    if abs(self.rect.right - item[0].left) < collision_thresh and self.speed_x > 0:
                        self.speed_x *= -1
                    #checks if collision was from the right
                    if abs(self.rect.left - item[0].right) < collision_thresh and self.speed_x < 0:
                        self.speed_x *= -1
                    #lowers the blocks strength when it has been hit by the ball
```

I have created the ball class which contains the draw function, move function and a collision checker. The move function recognises when the ball has come into contact with any of the walls or paddle ad finally the blocks.



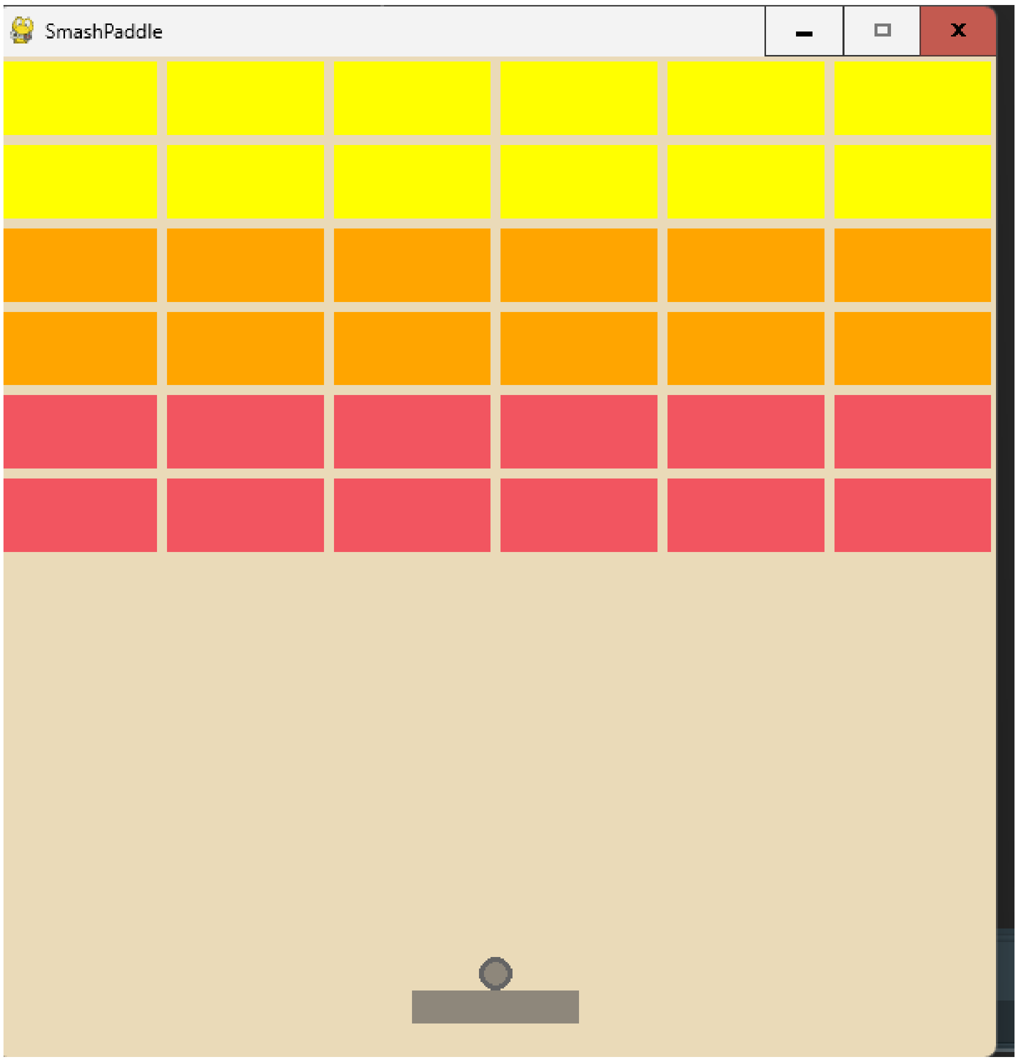
```python
def draw(self):
    pygame.draw.circle(screen, paddle_colour, (self.rect.x + self.ball_rad, self.rect.y + self.ball_rad), self.ball_rad)
    pygame.draw.circle(screen, paddle_outl, (self.rect.x + self.ball_rad, self.rect.y + self.ball_rad), self.ball_rad, 3)
```

This is the draw function, it creates the ball by being drawn onto the top of the paddle so that when the player starts the game, the ball comes off the paddle and is flung towards the wall of blocks.

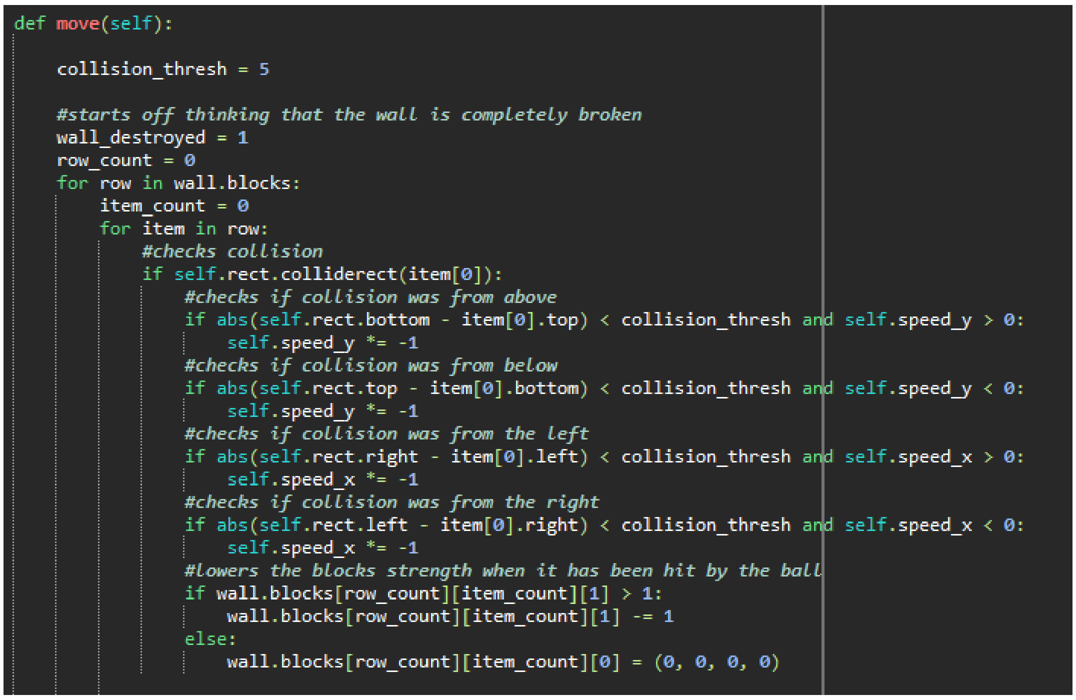
Testing:

What is being tested?	Input	Justification	Expected Outcome	Actual Outcome
Ball size	Change in integer for the ball radius value	Changes the size of the ball	The ball changes size and has a different radius	The ball changes size as expected

Ball colour changes	Different RGB number to original ball RGB number	To change the colour of the ball	The ball changes to the new RGB number's colours	The ball changes colour.
Ball attached to the paddle at the start	Location coordinates of the top of the paddle so the ball can be drawn on top of the paddle	Starts the game with the ball on top of the paddle so that when the game starts, it launches towards the blocks with the player looking at it.	The ball is drawn on top of the paddle when the game starts.	The ball appears on the paddle at the start of the game.

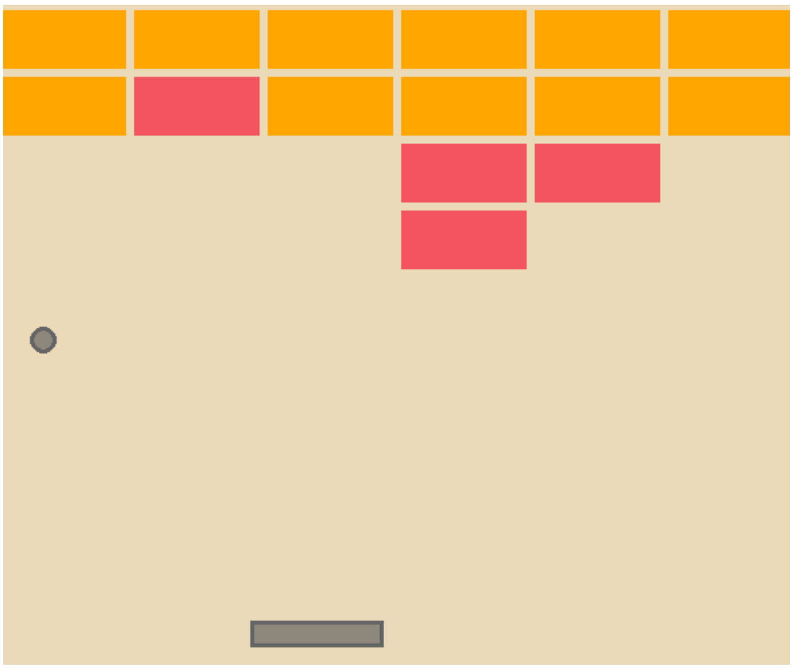


This is a screenshot of the game after the ball class has been generated and the location of the ball when the game starts. This gives the player time to see where the ball will start and anticipate their first move when the game gets going and the ball starts to move.

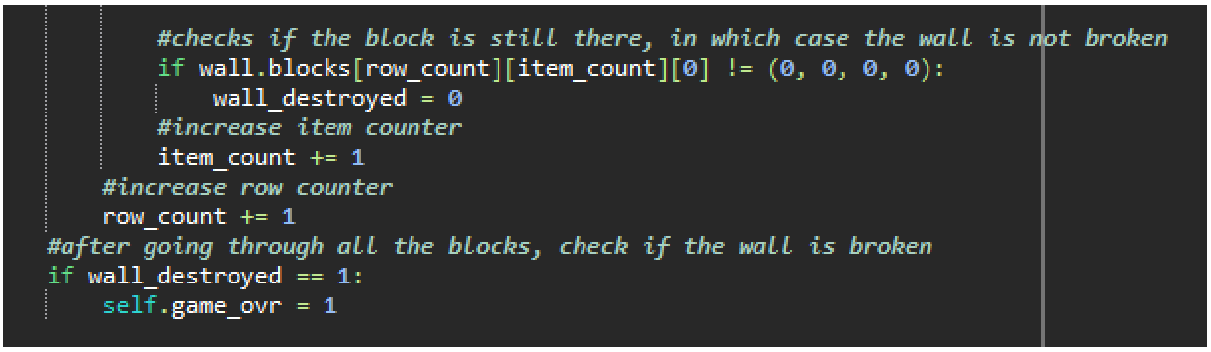


```
def move(self):

    collision_thresh = 5

    #starts off thinking that the wall is completely broken
    wall_destroyed = 1
    row_count = 0
    for row in wall.blocks:
        item_count = 0
        for item in row:
            #checks collision
            if self.rect.colliderect(item[0]):
                #checks if collision was from above
                if abs(self.rect.bottom - item[0].top) < collision_thresh and self.speed_y > 0:
                    self.speed_y *= -1
                #checks if collision was from below
                if abs(self.rect.top - item[0].bottom) < collision_thresh and self.speed_y < 0:
                    self.speed_y *= -1
                #checks if collision was from the left
                if abs(self.rect.right - item[0].left) < collision_thresh and self.speed_x > 0:
                    self.speed_x *= -1
                #checks if collision was from the right
                if abs(self.rect.left - item[0].right) < collision_thresh and self.speed_x < 0:
                    self.speed_x *= -1
                #lowers the blocks strength when it has been hit by the ball
                if wall.blocks[row_count][item_count][1] > 1:
                    wall.blocks[row_count][item_count][1] -= 1
                else:
                    wall.blocks[row_count][item_count][0] = (0, 0, 0, 0)
```

This is the first part of the move function. This section of code checks for collisions with the ball to see if it has collided with any of the walls or blocks. If the ball has collided with any of those, this code checks which direction the collision came from so that the ball can bounce in the appropriate direction because of the collision. If it collided with a block, it checks what strength/colour the block was so that if it had a higher strength than one, it would also lower that block's strength after the collision.

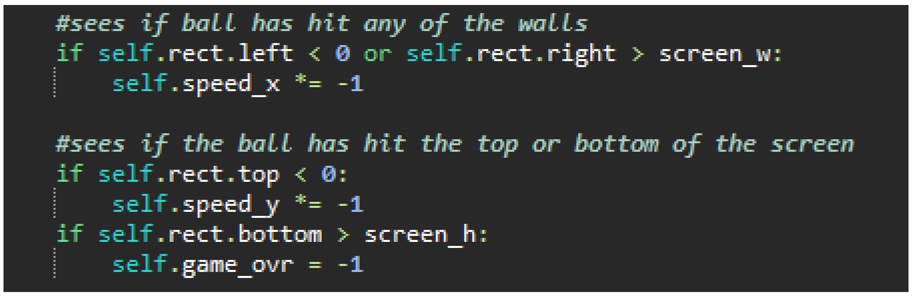


This is an example of that code being used as the strength of the orange block has decreased after it was hit by the ball. The ball was also sent in the other direction as an appropriate result of the collision.



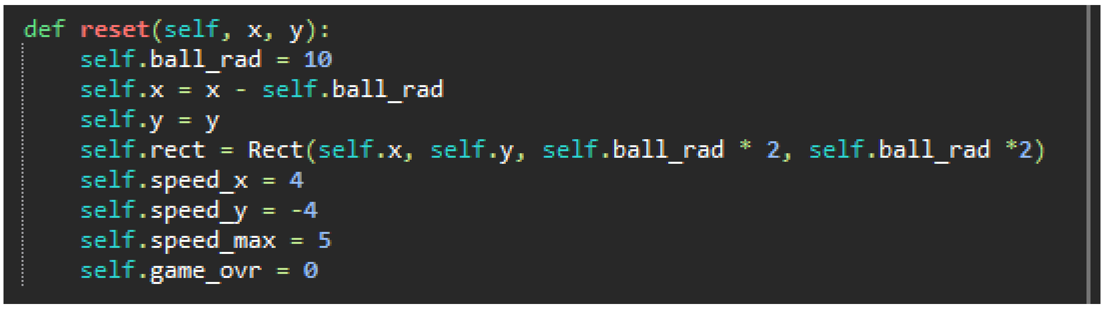
```
        #checks if the block is still there, in which case the wall is not broken
        if wall.blocks[row_count][item_count][0] != (0, 0, 0, 0):
            wall_destroyed = 0
        #increase item counter
        item_count += 1
    #increase row counter
    row_count += 1
#after going through all the blocks, check if the wall is broken
if wall_destroyed == 1:
    self.game_ovr = 1
```

This section of code checks whether all the blocks have been destroyed by the ball. If so, the game is over, and if not, then the wall is not yet broken.



```
#sees if ball has hit any of the walls
if self.rect.left < 0 or self.rect.right > screen_w:
    self.speed_x *= -1

#sees if the ball has hit the top or bottom of the screen
if self.rect.top < 0:
    self.speed_y *= -1
if self.rect.bottom > screen_h:
    self.game_ovr = -1
```

This code checks if the ball has collided with any of the walls so that the ball can bounce normally and none of the blocks will be affected. It also checks to see if the ball has collided with the bottom of the screen. If it has, that means that the ball has missed the paddle and therefore the game is also over.



```
def reset(self, x, y):
    self.ball_rad = 10
    self.x = x - self.ball_rad
    self.y = y
    self.rect = Rect(self.x, self.y, self.ball_rad * 2, self.ball_rad *2)
    self.speed_x = 4
    self.speed_y = -4
    self.speed_max = 5
    self.game_ovr = 0
```

Finally, the last part of the ball class resets the ball's values when the game is over and the player restarts. The ball will go back to being attached to the paddle and the blocks will be reset.

Test Data	Type	Justification	Actual Test/Result
Positive integer that is preferred to be between 1 and 3 for the radius of the ball	Valid	Adjusts the size of the ball's radius and therefore changes the size of the ball	Valid
Negative integer	Invalid	This would remove the ball from the screen however the game would carry on	Invalid
Anything that is not a number e.g., string	Invalid	This would create an error	Invalid

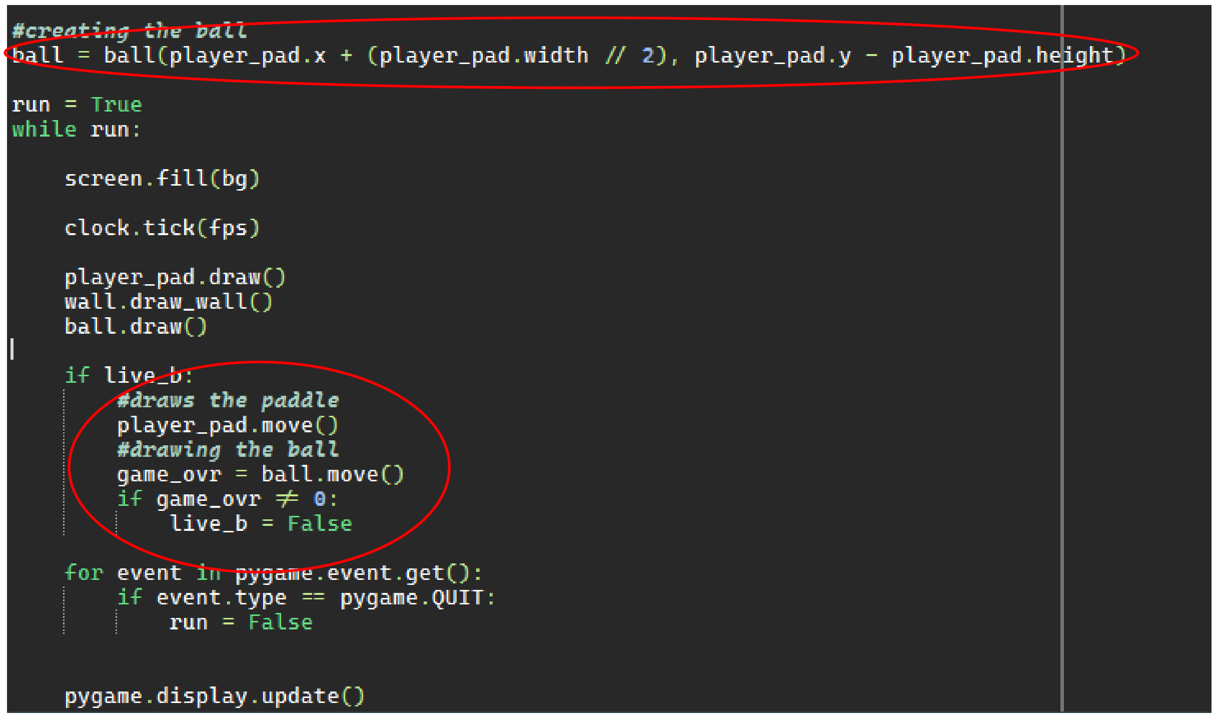
Draw Ball Function:

This is the draw function of the ball written in pseudocode.

```
Function draw():
  Draw filled circle on the screen then
    Use paddle_colour
    Center circle at (self.rect.x + self.ball_rad, self.rect.y + self.ball_rad)
    Set radius of the circle = self.ball_rad

  Draw outlined circle on the screen then
    Use colour paddle_outl
    Center circle at (self.rect.x + self.ball_rad, self.rect.y + self.ball_rad)
    Set radius of the circle = self.ball_rad
    Set outline thickness = 3
endfunction
```

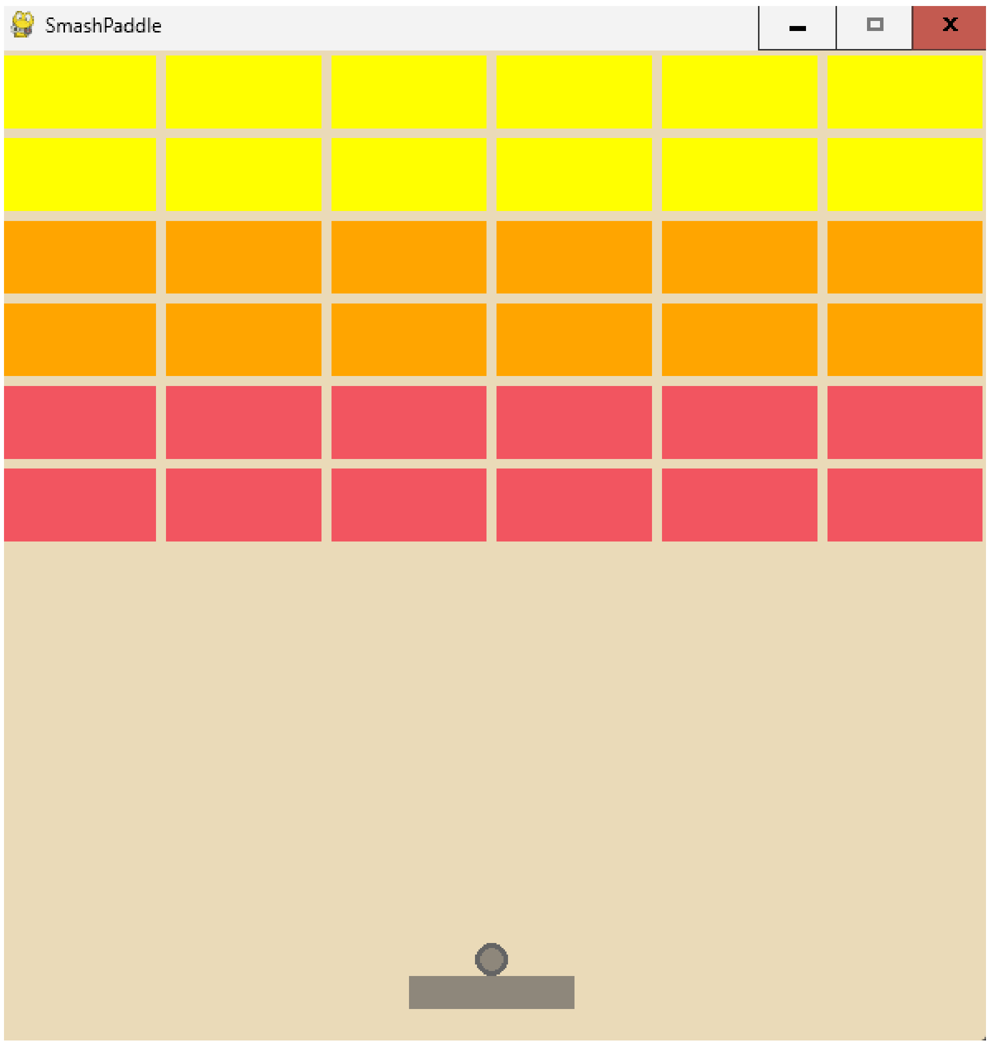
Errors:

At the beginning of this prototype, I encountered an error where the ball would be stuck to the paddle and no matter what key the player pressed, the ball would not move. The screenshot below is the code that contains the error.

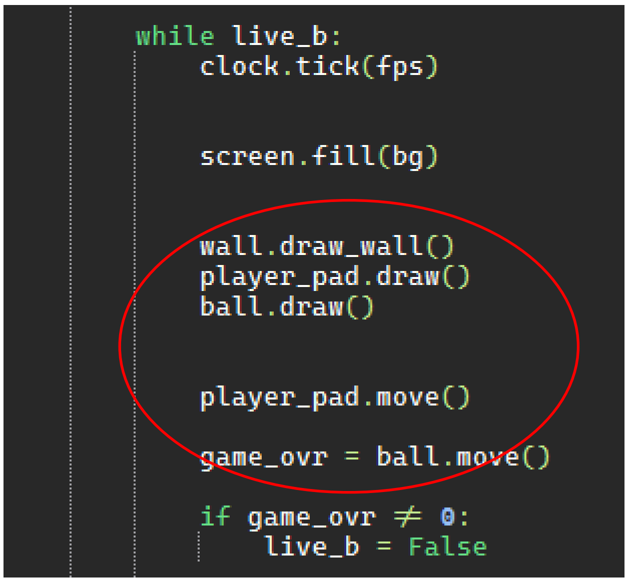


```
#creating the ball
ball = ball(player_pad.x + (player_pad.width // 2), player_pad.y - player_pad.height)

run = True
while run:

    screen.fill(bg)

    clock.tick(fps)

    player_pad.draw()
    wall.draw_wall()
    ball.draw()

    if live_b:
        #draws the paddle
        player_pad.move()
        #drawing the ball
        game_ovr = ball.move()
        if game_ovr != 0:
            live_b = False

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


    pygame.display.update()
```

As you can see, there is a piece of code that draws the ball onto the middle of the paddle. However, inside the main loop, there is a piece of code missing that allows the ball to move when the game is started. The output of this code is below.

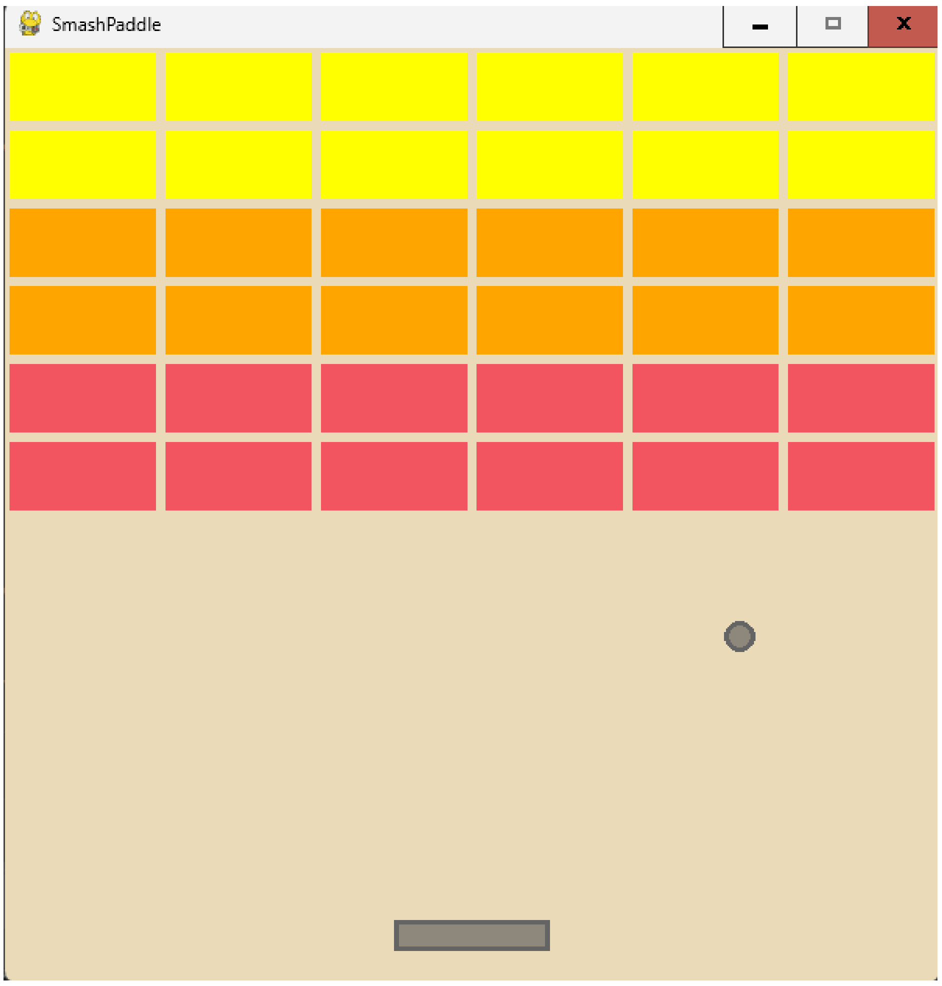


As you can see from the screenshot of the output, the ball has not been able to move from the paddle at the start of the game and is essentially ‘stuck’. I have attached a screenshot below of the new code below that fixes this issue.



```
    while live_b:
        clock.tick(fps)


        screen.fill(bg)


        wall.draw_wall()
        player_pad.draw()
        ball.draw()


        player_pad.move()

        game_ovr = ball.move()

        if game_ovr ≠ 0:
            live_b = False
```

In this code, I have created a while loop that essentially draws all the objects at the start of the game and gives the paddle and ball the ability to move when the game starts. The part circled in red is the specific code that allows this to happen. I changed this code so that the ball can actually move at the start of the game because the player needs the ball to move in order to play the game. The screenshot below shows the new output of the game working as it should do with the ball moving.



Prototype 3 Review:
Success Criteria:

Requirements	**Justification**	☑	**Reference**
Ball bouncing mechanics	This allows the player to play the games and use the paddle and ball to destroy the wall of blocks	☑	Stakeholder interview 1/Feature of “Atari: Breakout”
Realistic ball bounces	This makes the game seem realistic and lets the player anticipate where the ball will bounce to after a collision	☑	Stakeholder/Feature of “Atari: Breakout”

Ball travels smoothly when running	This allows the game to look clean and effective. It also makes sure that the player's experience is the best it can be	☑	Stakeholder/Feature of "Atari: Breakout."

I have successfully achieved all three of the main requirements that were set out at the beginning of the project for the ball class. Firstly, the ball bouncing mechanics work very well and are better than expected. However, there were a few errors and currently there is a main one which is a random occurrence where the ball will sometimes destroy a block and then continue its original path to the block behind which can sometimes result in the player destroying more blocks than they should have.

Secondly, the realistic ball bounces again work as well as expected but suffer from that same issue where sometimes the ball continues a direct path to the next block. However, when the ball collides with the walls or the paddle, the bounces work very well.

Lastly, when the ball travels across the screen between collisions, it runs as smoothly as expected and runs at 60fps as expected.

Prototype 3 Interview Plan:

- Should the ball be a different size?
- Should the ball's speed be changed or kept to what it currently is?
- Are the bounces and collisions of the ball realistic enough?

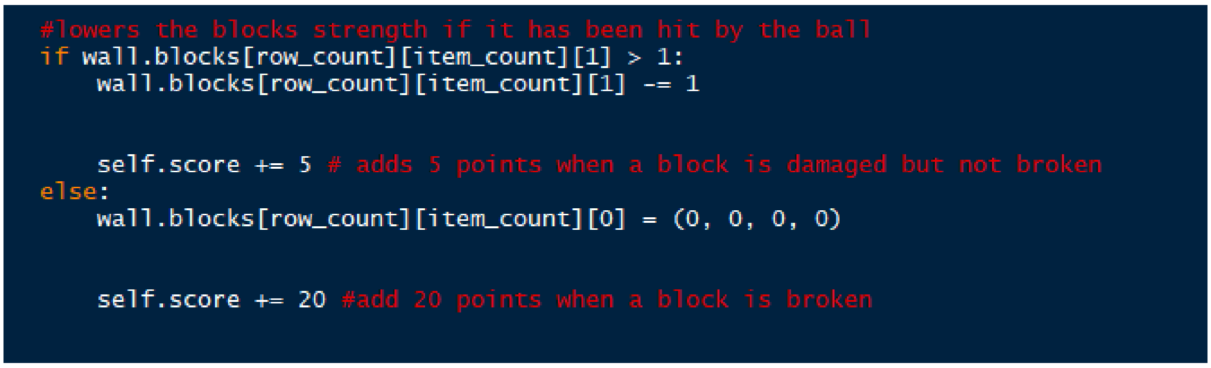
Prototype 3 Interview with Finlay Baker:

- Should the ball be a different size?
  "No, it is fine as it is."
- Should the ball's speed be changed or kept to what it currently is?
  "Keep the ball's speed at its current rate as it means the game doesn't pass too quickly."
- Are the bounces and collisions of the ball realistic enough?
  "Yes, they are fine as they are."

Conclusion:
The ball's speed and current size will be kept the same as they were originally written. When the game starts, the ball will be attached to the top of the paddle so that when the game starts, the player can clearly see the path of the ball before its first collision. I am currently working on a fix for the random collision glitch however everything else works as well as anticipated.

Prototype 4

Creating the score system (01/10/24):

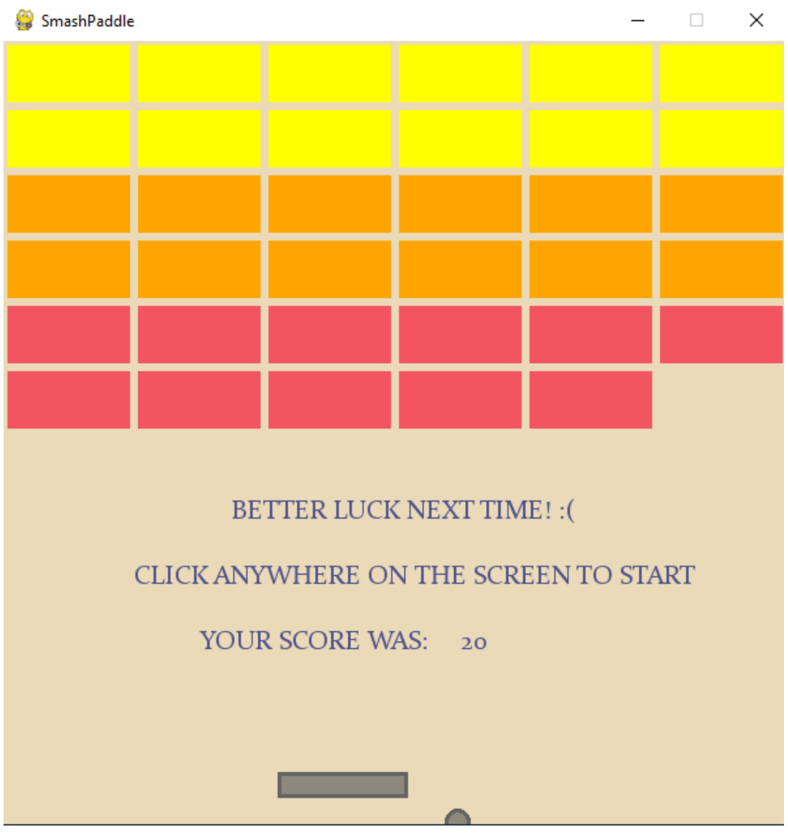


```
#lowers the blocks strength if it has been hit by the ball
if wall.blocks[row_count][item_count][1] > 1:
    wall.blocks[row_count][item_count][1] -= 1


    self.score += 5 # adds 5 points when a block is damaged but not broken
else:
    wall.blocks[row_count][item_count][0] = (0, 0, 0, 0)


    self.score += 20 #add 20 points when a block is broken
```

I have created a score system where the score updates every time a block is hit by the ball or destroyed. When a block is destroyed, the score goes up by 20, and when a block of a higher strength is hit, the strength lowers, and the player is awarded an extra 5 points. This code therefore uses validation as it is checking whether the block has been destroyed or if a block with a higher strength has been hit as well.

Testing:

What is being tested?	**Input**	**Justification**	**Expected Outcome**	**Actual Outcome**
Score resets after the player loses	Game over = 1	Score goes back to zero every time a new game is started	The score resets to zero	The score resets to zero as expected
Score goes up by 5 when the ball hits a block with strength greater than 1	Hit a higher strength block with the ball	To see if the score goes up the required amount when the collision occurs	The score goes up by 5	The score goes up by 5 as expected
Score goes up by 20 when the ball destroys a block	Destroy a block with the ball at the start of the game	To make sure that by the end of the game, the player will be shown the correct score that they earned	The score goes up by 20 when the block is destroyed	The score goes up by 20 as expected

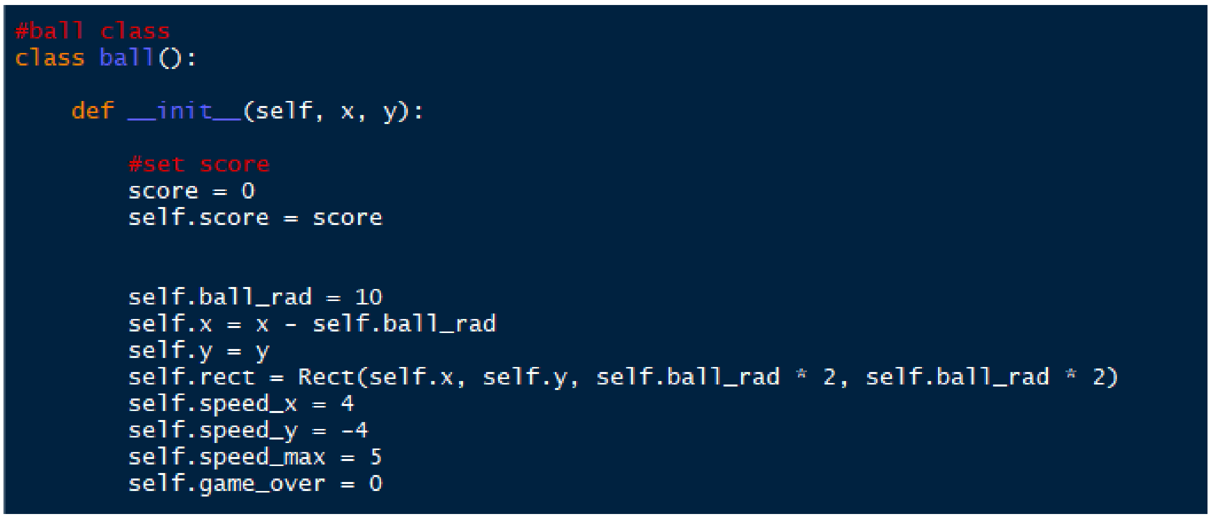


BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

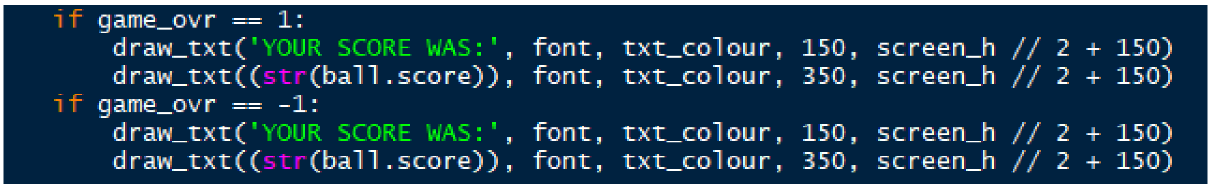
YOUR SCORE WAS: 20

As you can see, when the game is over the score is displayed at the top of the screen and shows the correct score that the player has earned corresponding to how many blocks, they have damaged or broken.



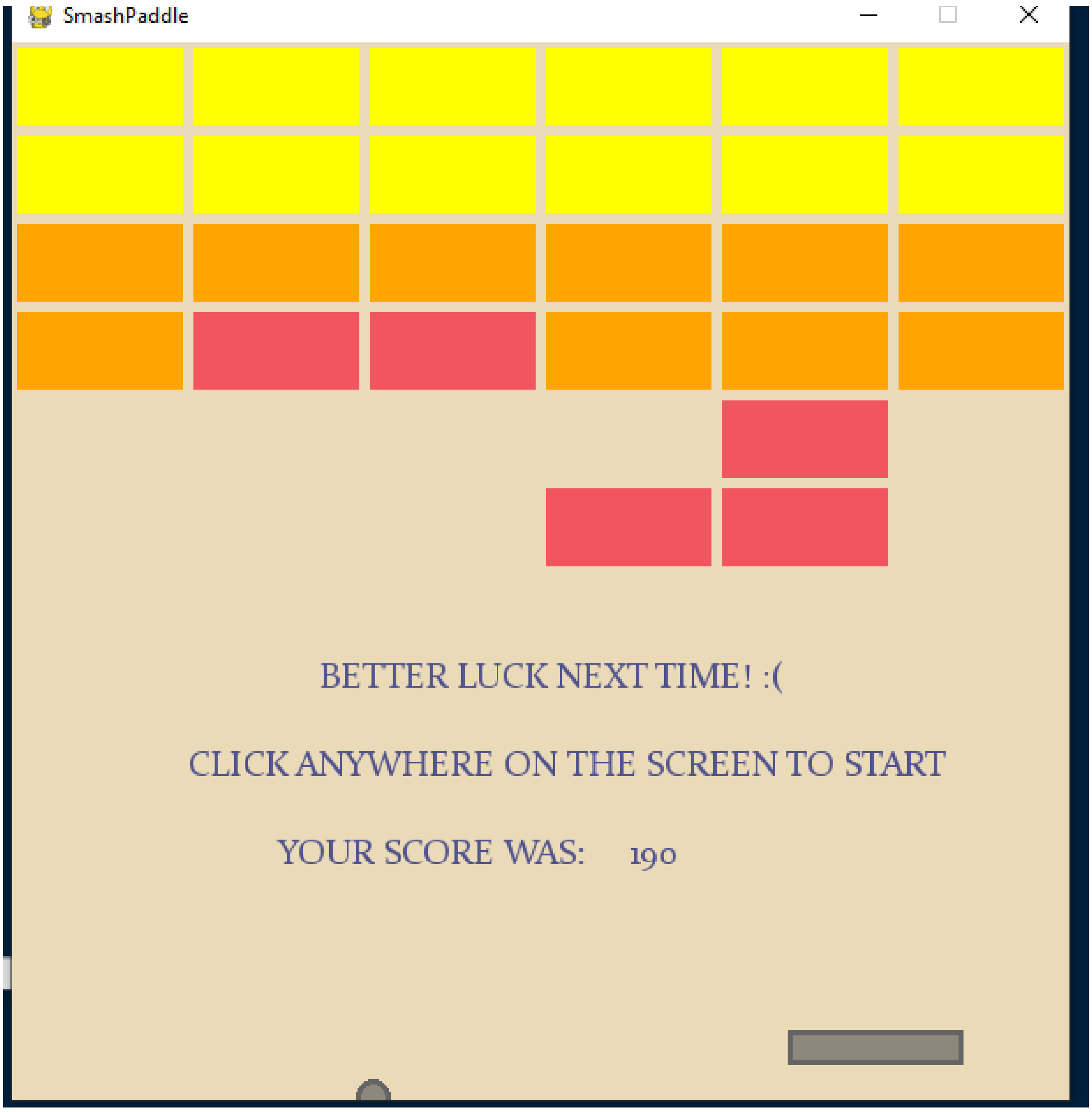
```
#ball class
class ball():

    def __init__(self, x, y):

        #set score
        score = 0
        self.score = score


        self.ball_rad = 10
        self.x = x - self.ball_rad
        self.y = y
        self.rect = Rect(self.x, self.y, self.ball_rad * 2, self.ball_rad * 2)
        self.speed_x = 4
        self.speed_y = -4
        self.speed_max = 5
        self.game_over = 0
```

In this screenshot, the score variable has been set at the beginning of the ball class and has become an attribute. This then allows the score to be easily updated throughout the ball class whenever the ball comes into contact with any of the blocks.



```
   if game_ovr == 1:
       draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
       draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
   if game_ovr == -1:
       draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
       draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
```

In this screenshot, the score is displayed at the end of the game when either the player wins or loses. It is displayed through text by being converted into a string and is drawn onto the screen under the final message. It is the same size and colour as the other text that is drawn onto the screen throughout stages of the game.



BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

YOUR SCORE WAS: 190

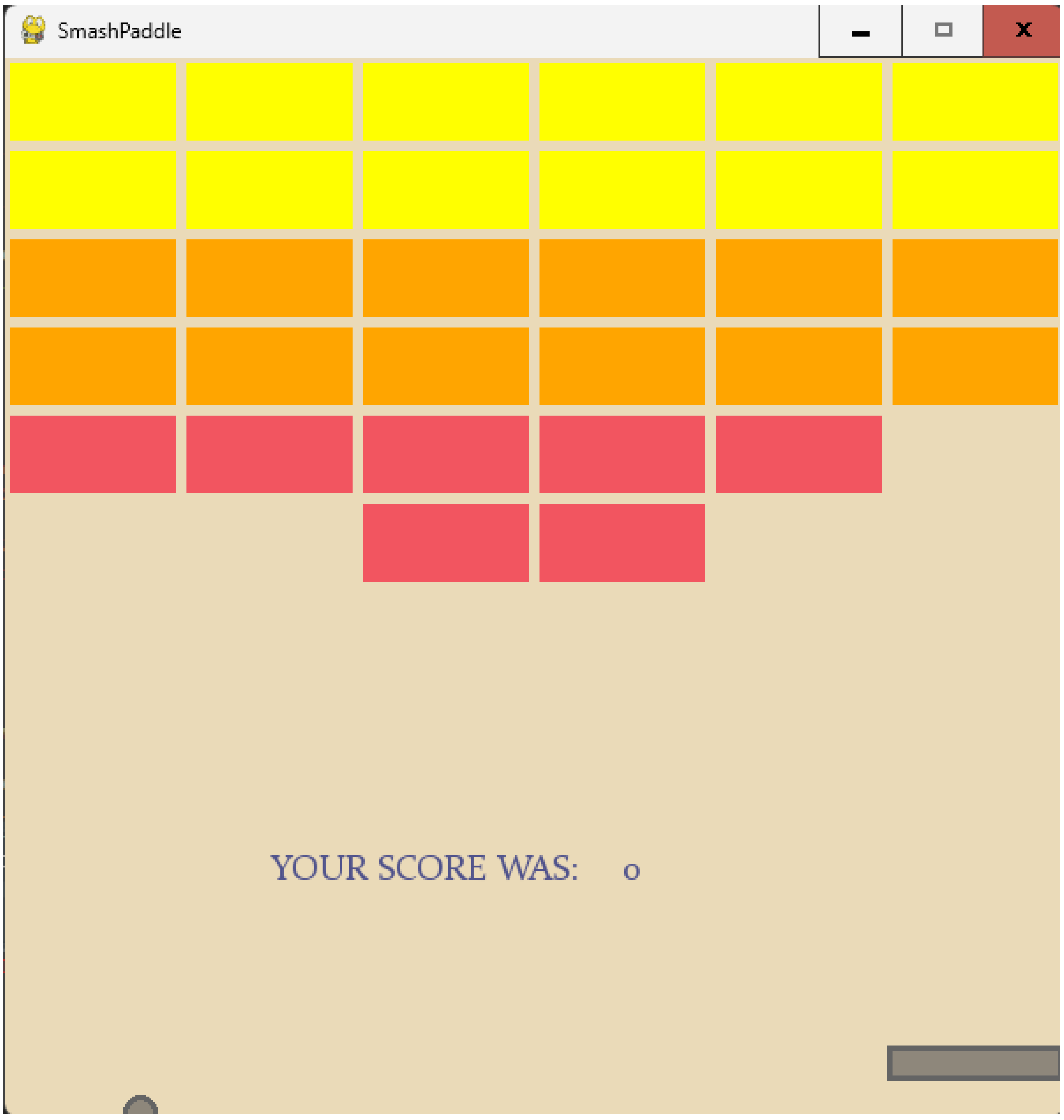
This is what is outputted when the player loses the game. As you can see, the score displayed is the same colour as the other text being displayed and is drawn onto the screen after being converted into a string value.

Test Data	Type	Justification	Actual Test/Result
Score settings to change so that a different number of points are awarded instead of the original set amount	Valid	Adjusts the amount of score the player receives when hitting or destroying a block	Valid

Score not being converted into a string before being drawn onto the screen as text	Invalid	This would cause an error as the computer would expect a string value and therefore will not display it	Invalid
A word is added to the score when the ball hits a block instead of an integer	Invalid	This would create an error	Invalid

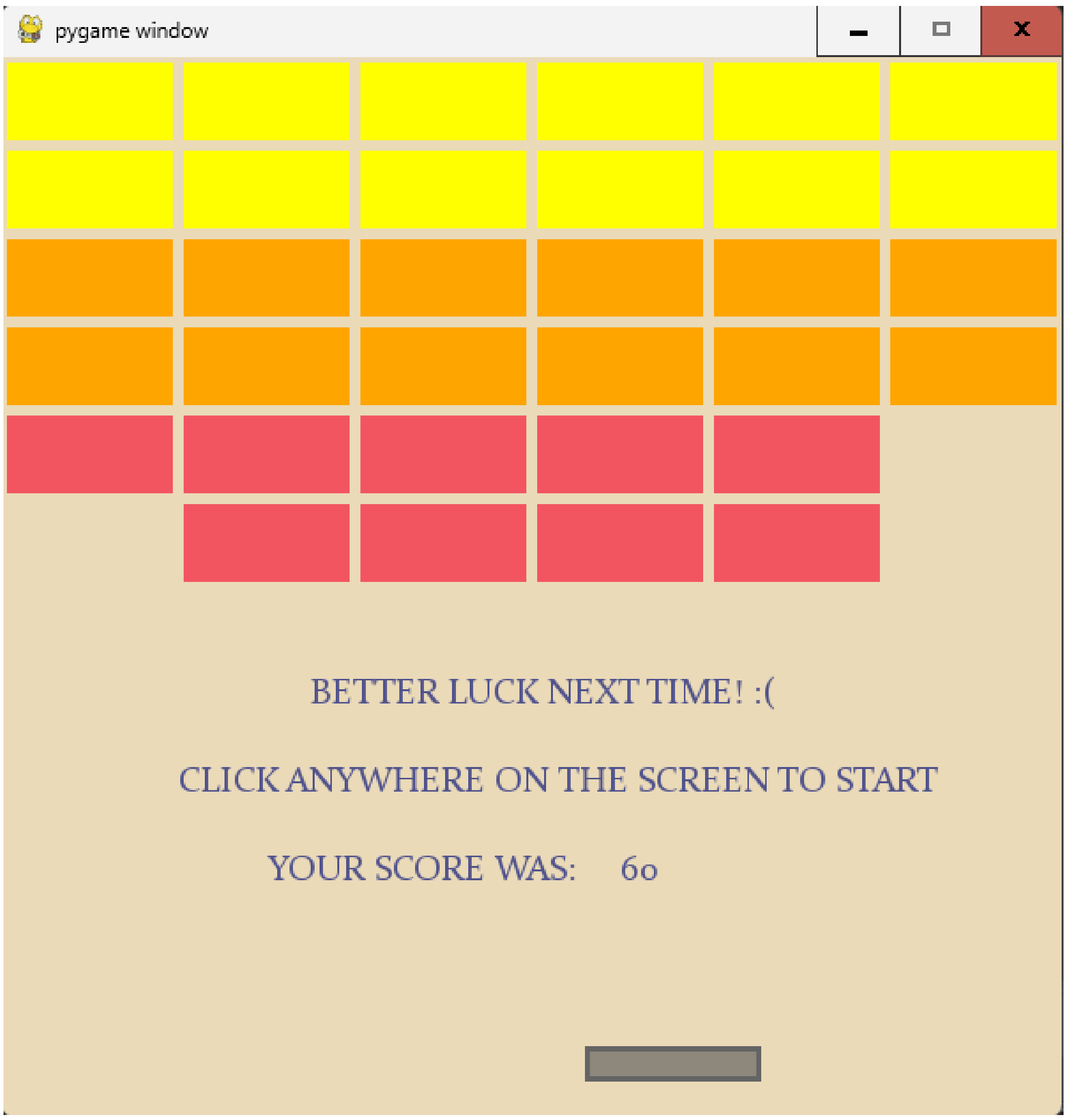
<u>Errors:</u>

When creating the score system, I encountered several errors that mean the score system didn't perform as well as I expected. One of the errors was that the score, no matter how many blocks the player destroyed, printed out as zero. The output below shows this error.



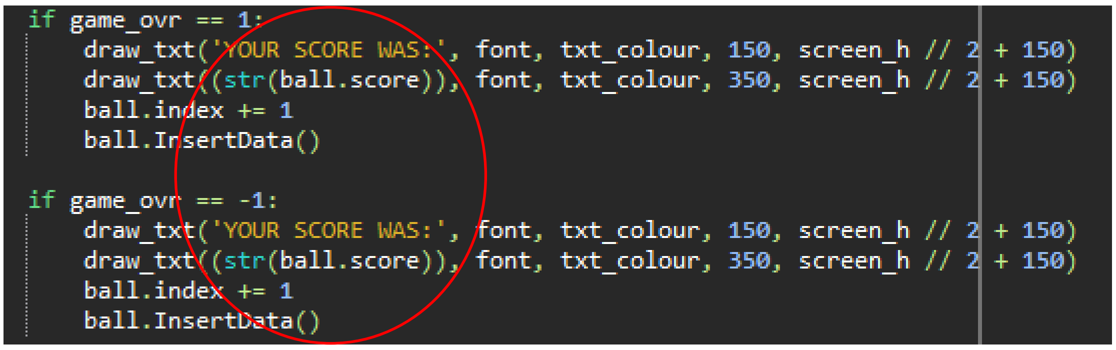
YOUR SCORE WAS: 0

This error occurred because the score variable was set to zero and no matter where I seemed to add a score increase into the code, the score would still print out as zero. To fix this, I made score a variable inside the ball class so that whenever the ball hits a block, it will increase as part of the ball class, and then made the variable global so it will print out the true score. This is the fixed output below.



```
BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

YOUR SCORE WAS:     60
```

This is the code I wrote to fix the error.



```
if game_ovr == 1:
    draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
    draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
    ball.index += 1
    ball.InsertData()

if game_ovr == -1:
    draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
    draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
    ball.index += 1
    ball.InsertData()
```

It draws the score value onto the screen and the values is inside the ball class and so it is constantly checking if it needs to be updated and resets when the game is over. I changed this code so that the correct score would be displayed at the end of the game and also so that the correct score is stored in the database because otherwise it would display and store every value as 0 every game.

Score system inside Ball Class:

```
//checks if collision was from above
          If collision was from greater y value than ball then
             self.speed_y *= -1



          //checks if collision was from below
          If collision was from smaller y value than ball then
             self.speed_y *= -1



          //checks if collision was from the left
          If collision was from smaller x value than ball then
             self.speed_x *= -1



          //checks if collision was from the right
          If collision was from greater x value than ball then
             self.speed_x *= -1



          //lowers the blocks strength if it has been hit by the ball
          if block strength > 1 then
               block strength -= 1


             score = score + 5
```

I have written the pseudocode for the collision check between the ball and blocks and the score updates. Following this pseudocode has allowed me to create an easy-to-use score system that can easily be changed in case the score values want to be changed.

```
else:
   block is destroyed



   score = score + 20
```

Prototype 4 Review:
Success Criteria:

Requirements	**Justification**	☑	**Reference**
Score to update based on how many/what blocks are hit or destroyed	This allows the player to gain a score based on their progress in the game so they feel a sense of achievement and will have something to work to improve on	☑	Stakeholder interview 1/Feature of "Atari: Breakout"
Score to be displayed at the bottom of the screen at the end of the game	This lets the player know how well they have done at the end of each game and again gives them something to try and improve on	☑	Stakeholder Interview/Succes Criteria
Score to be saved into an SQL database so that a leaderboard can be created	This allows multiple players to try and compete against each other to see who can get the highest score which makes the game more exciting as it creates a sense of immersion and competition	☐	Stakeholder/Feature of "Atari: Breakout."

Unfortunately, I was not able to complete all of the objectives set out for score at the beginning of this project however I will work on the database as part of Prototype 6. This then means that I have successfully met all of the other objectives set out for the score system.

Firstly, the score system updates every time a block is hit by the ball. 5 points are awarded if it is a higher strength block that has been hit, and 20 points are awarded to the player if they successfully destroy a block.

Lastly, the score is displayed in text form at the bottom of the screen after the game has ended. It is displayed underneath a message either congratulating the player or encouraging them to have another go as they did not complete the game. The score is converted into a string value and then is drawn as text in the same colour and size as the other text being drawn.

Prototype 4 Interview Plan:

- Should the points awarded be changed to be more or less?
- Should points be awarded for lowering a block's strength or is it too generous?
- Should the score be displayed somewhere else on the screen after the game has ended?

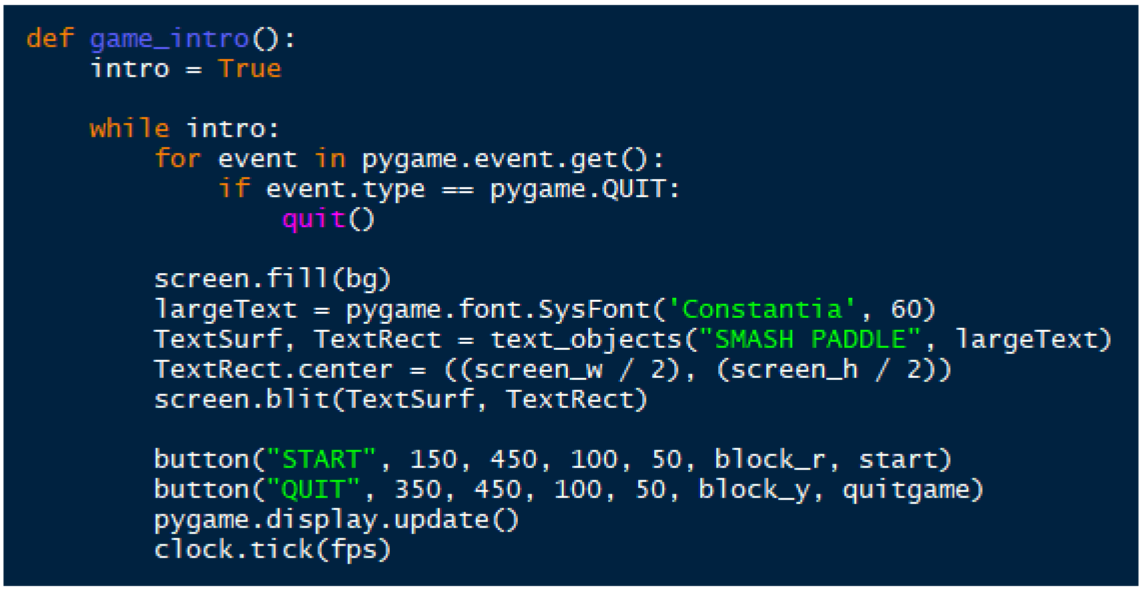
Prototype 4 Interview with Finlay Baker:

- Should the points awarded be changed to be more or less?
  "No, the current number of points being awarded is fair and sufficient enough."
- Should points be awarded for lowering a block's strength or is it too generous?
  "No, I think that awarding points for lowering a block's strength makes the competition closer for those that progress further in the game as it is the small amounts that can add up."
- Should the score be displayed somewhere else on the screen after the game has ended?
  "The score being displayed at the bottom of the screen is a perfect place as it blends in with the message being displayed and the player will already be looking at the message at the bottom of the screen anyways."

Conclusion:
The score system is currently fair and balanced, awarding the same number of points for any blocks no matter what their strength, and the blocks that have a higher strength, the points gained for lowering their strength are equal no matter what their strength as to make the game fairer for everyone. I think that the score system is vital to the game as it creates a sense of competition and immersion to the game which makes it exciting for players of all ages.

Prototype 5

Creating the menu system (03/10/2024):

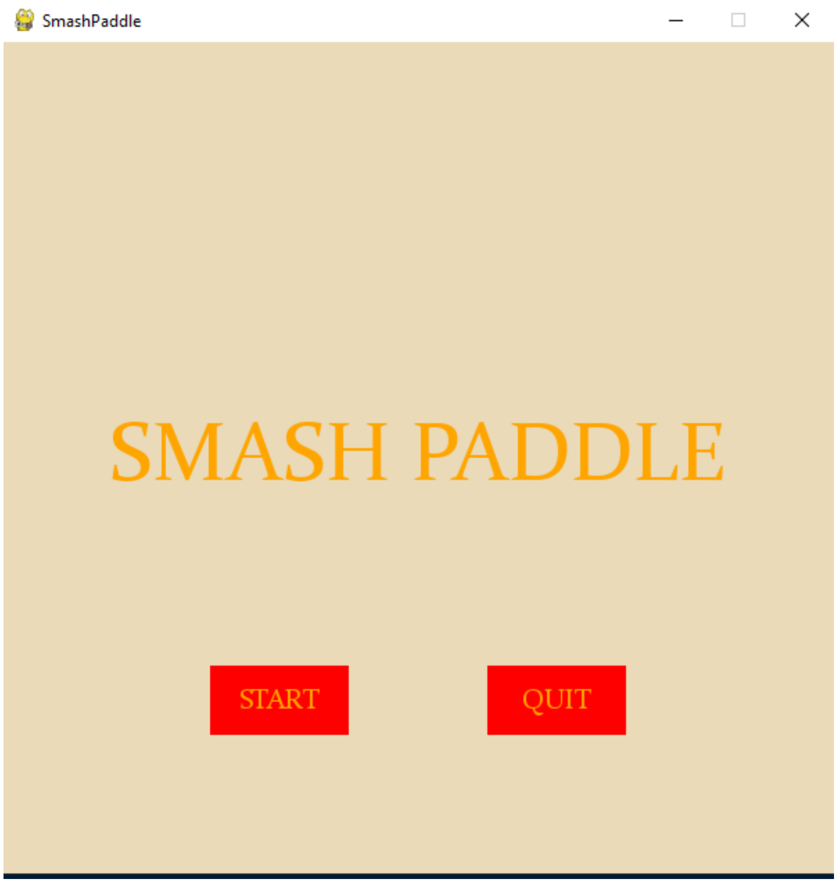


```python
def game_intro():
    intro = True

    while intro:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                quit()

        screen.fill(bg)
        largeText = pygame.font.SysFont('Constantia', 60)
        TextSurf, TextRect = text_objects("SMASH PADDLE", largeText)
        TextRect.center = ((screen_w / 2), (screen_h / 2))
        screen.blit(TextSurf, TextRect)

        button("START", 150, 450, 100, 50, block_r, start)
        button("QUIT", 350, 450, 100, 50, block_y, quitgame)
        pygame.display.update()
        clock.tick(fps)
```

This screenshot of code contains the code for the opening menu. The menu appears as two buttons, one to start the game, and the other to exit the game. Currently, I am working on a fix for the start button as when it is clicked, nothing happens. The quit button on the other hand works as intended and swiftly closes the game when clicked.

Testing:

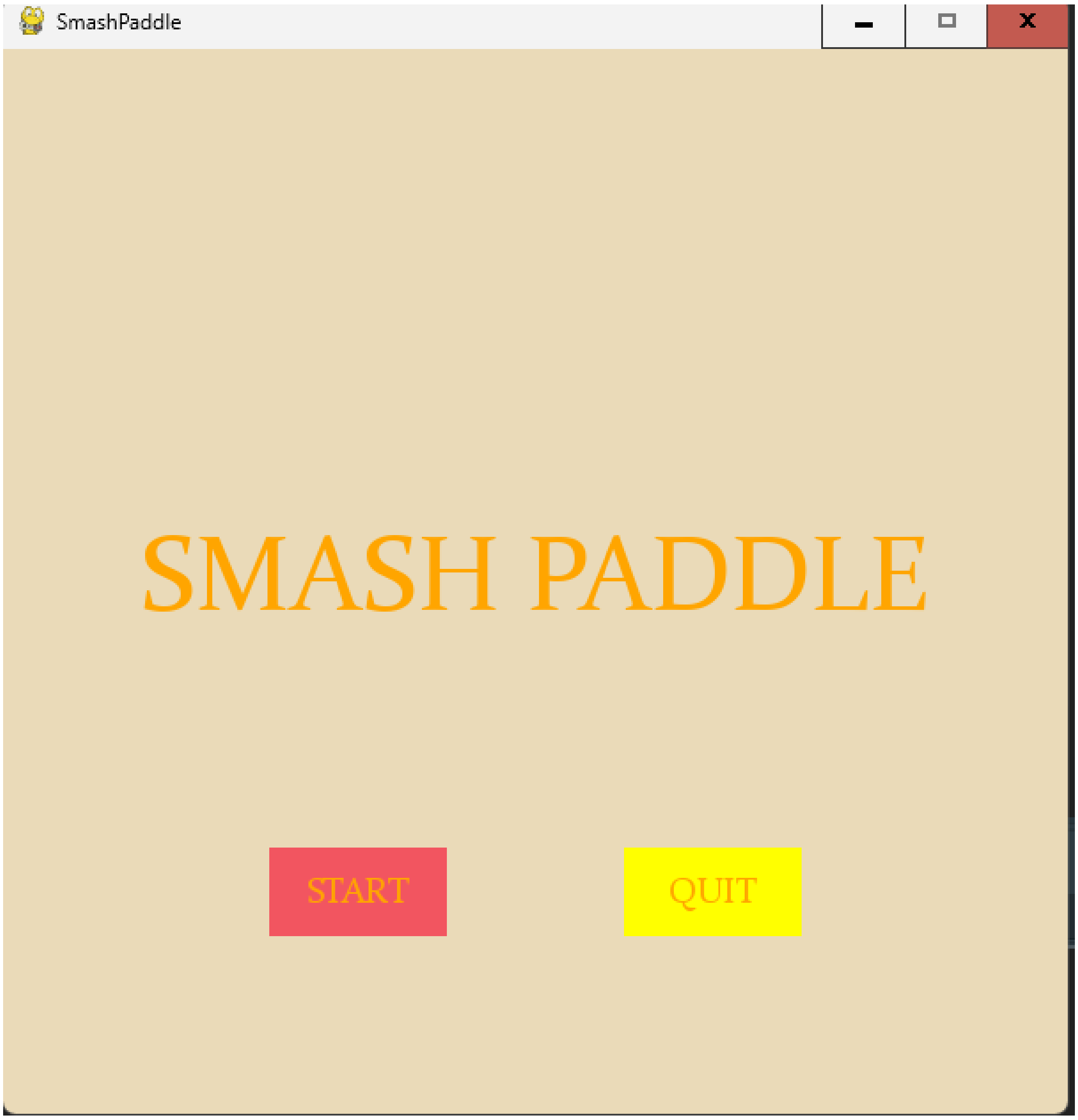
What is being tested?	**Input**	**Justification**	**Expected Outcome**	**Actual Outcome**
Start button being clicked	Mouse being pressed over the area of the start button	To see if the game will start when the player requests	The menu will disappear, and the game will begin as usual	Nothing – error already explained and currently working on a fix
Quit button being clicked	Mouse button being pressed over the area of the quit button	To see if the game will close itself when the player requests	The game will exit when the quit button is pressed	The game closes and the player quit the game
Anywhere else on the opening menu being clicked	Mouse button being clicked anywhere else on the screen other than the buttons	To make sure that the program won't do anything not intended when anything else is clicked	Nothing will happen as none of the buttons have been clicked so no action is taken	Nothing happens as expected



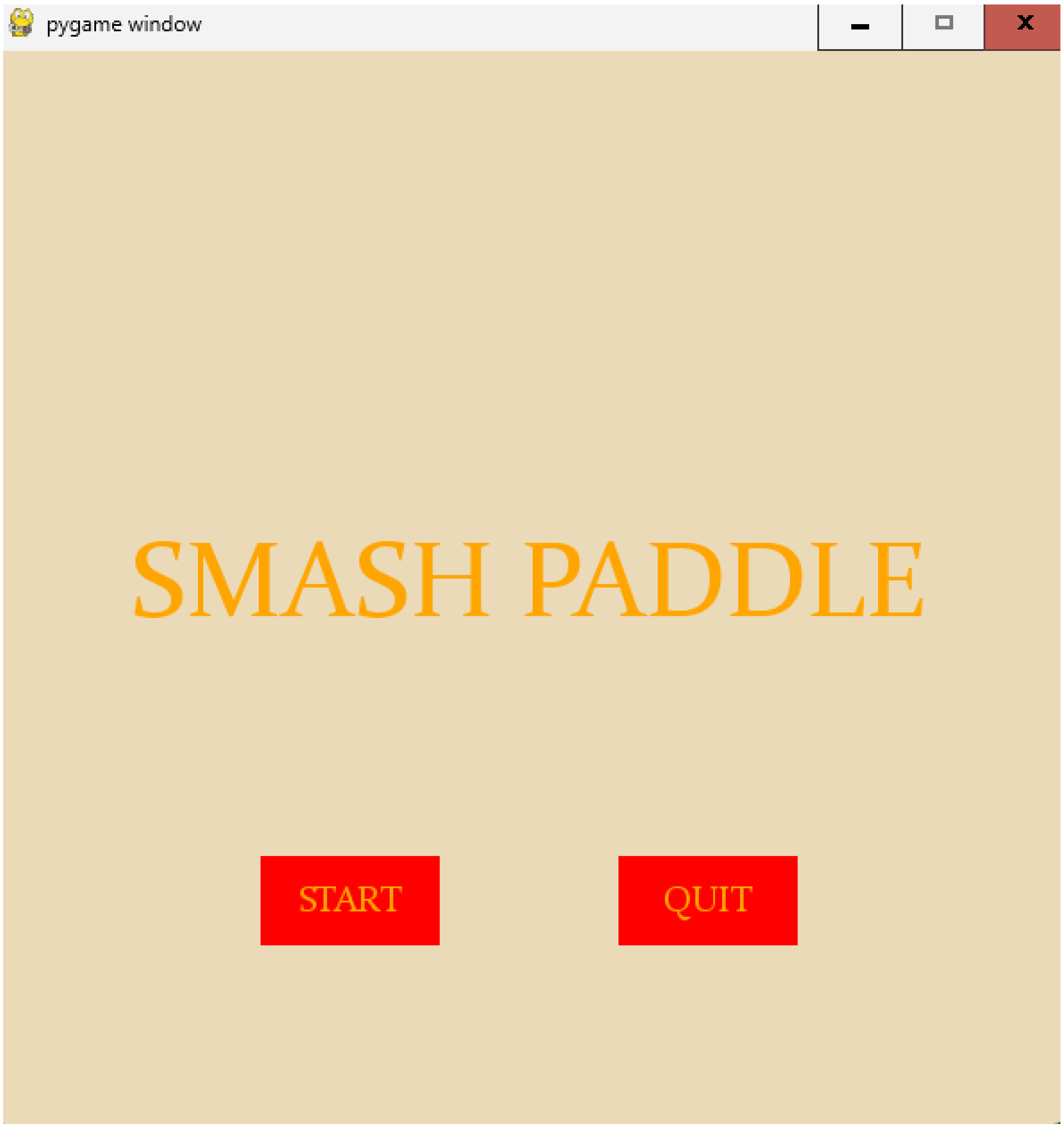
This is what the main menu looks like when the code is run. The title of the game is displayed in a large and easy to read text, with both buttons being a vibrant red colour so that the player can easily see them. The text on the buttons is the same colour as the title so that it is easy to read against the colour of the buttons.

Original Error:

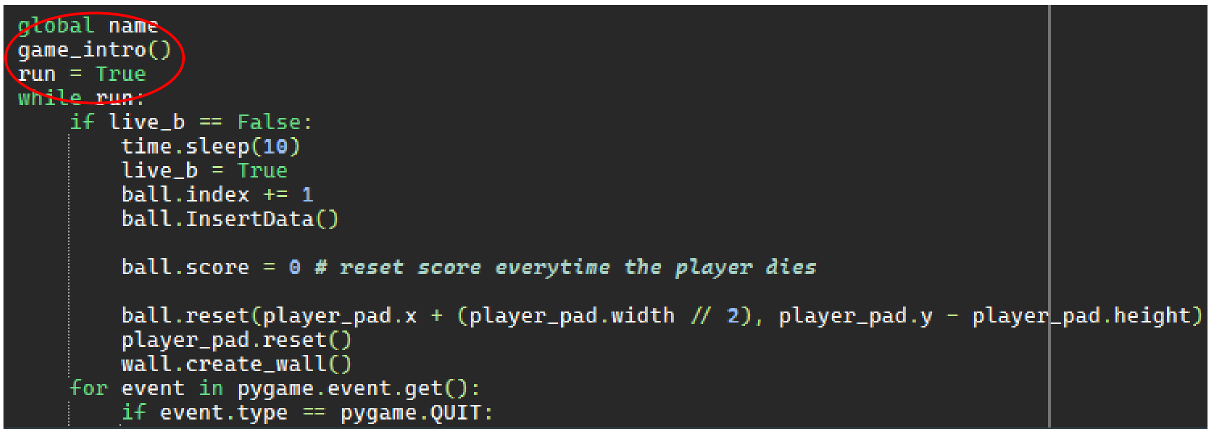
The original error with the menu was that the game would begin as I had originally coded it to where no menu would appear, and the player would have to click the screen to start the game. However, because the menu code was already written as well, when the game ended, then the starting menu came up. As well as that, the only button the player was allowed to click was the quit button which remained highlighted despite not hovering the mouse over it. The screenshot below shows this error in action.



This error was caused by the menu code being called in the wrong place. Originally, the game intro function was called when the game over variable was equal to zero, However, what I did not realise was that the game over variable was only equal to zero when the ball was not moving, and since the ball started moving when the game is opened, the menu would only appear when it stopped moving – at the end of the game. To fix this issue, the game intro function was called outside the main loop so that it would be the first thing carried out before the loop ran. This allowed the menu to appear at the start of the game as originally expected. The screenshot below shows the menu appearing when it should do.

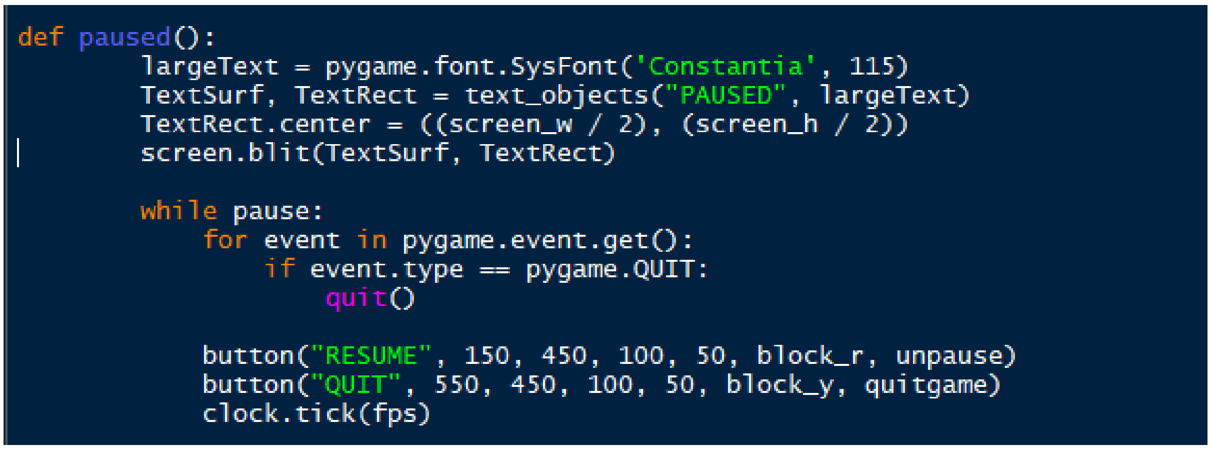


This is the code below that fixed the problem.



```
global name
game_intro()
run = True
while run:
    if live_b == False:
        time.sleep(10)
        live_b = True
        ball.index += 1
        ball.InsertData()

        ball.score = 0 # reset score everytime the player dies

        ball.reset(player_pad.x + (player_pad.width // 2), player_pad.y - player_pad.height)
        player_pad.reset()
        wall.create_wall()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
```

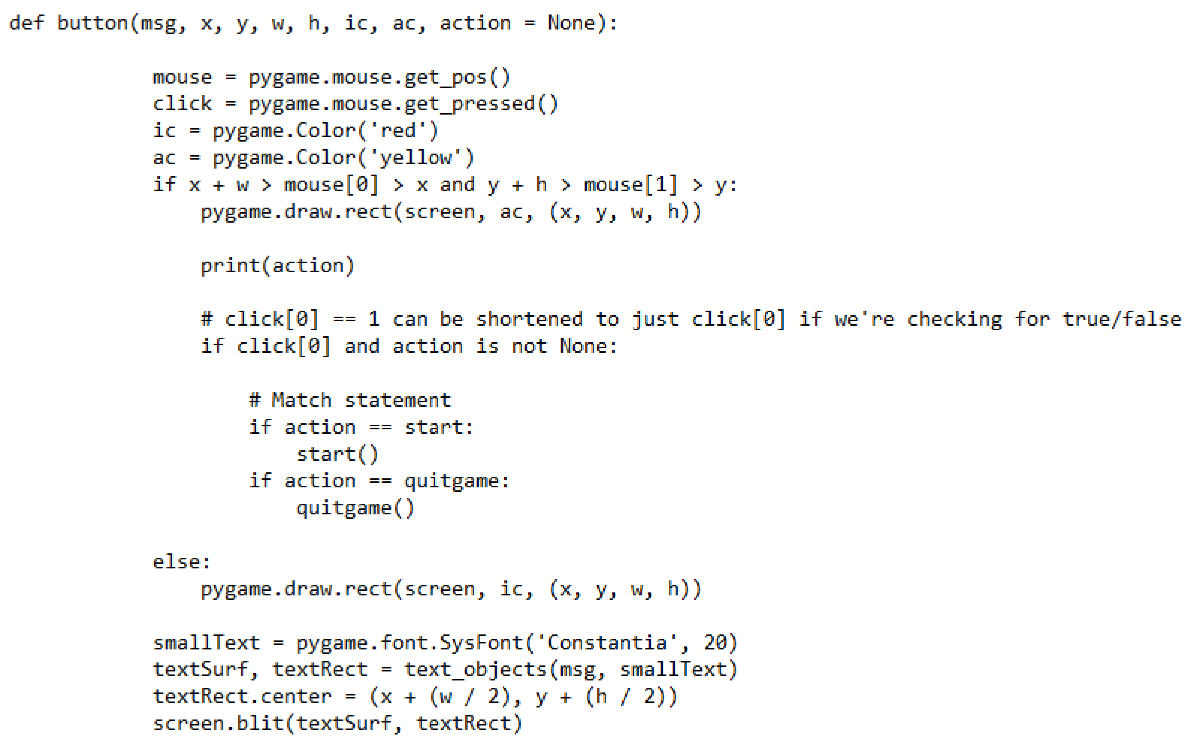
I changed this code so that the main menu would appear at the start of the game and not the end because otherwise the menu would be in the wrong place and it would give the game a bad impression on the player.



```
def paused():
        largeText = pygame.font.SysFont('Constantia', 115)
        TextSurf, TextRect = text_objects("PAUSED", largeText)
        TextRect.center = ((screen_w / 2), (screen_h / 2))
        screen.blit(TextSurf, TextRect)

        while pause:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    quit()

            button("RESUME", 150, 450, 100, 50, block_r, unpause)
            button("QUIT", 550, 450, 100, 50, block_y, quitgame)
            clock.tick(fps)
```

In this screenshot, this is the code for the pause menu. When the player activates the pause menu, it will display text saying 'PAUSED' in the centre of the screen. The player will then be given two buttons to press, one to resume the game, and one to exit the game. Unfortunately, the main menu does not work currently and therefore the rest of the menus and games are locked and therefore I am unable to provide live screenshots of the results when the code is run because the code does not work. However, I am currently putting in a lot of effort to work on a quick fix for this issue.

Creating the buttons (03/10/24):



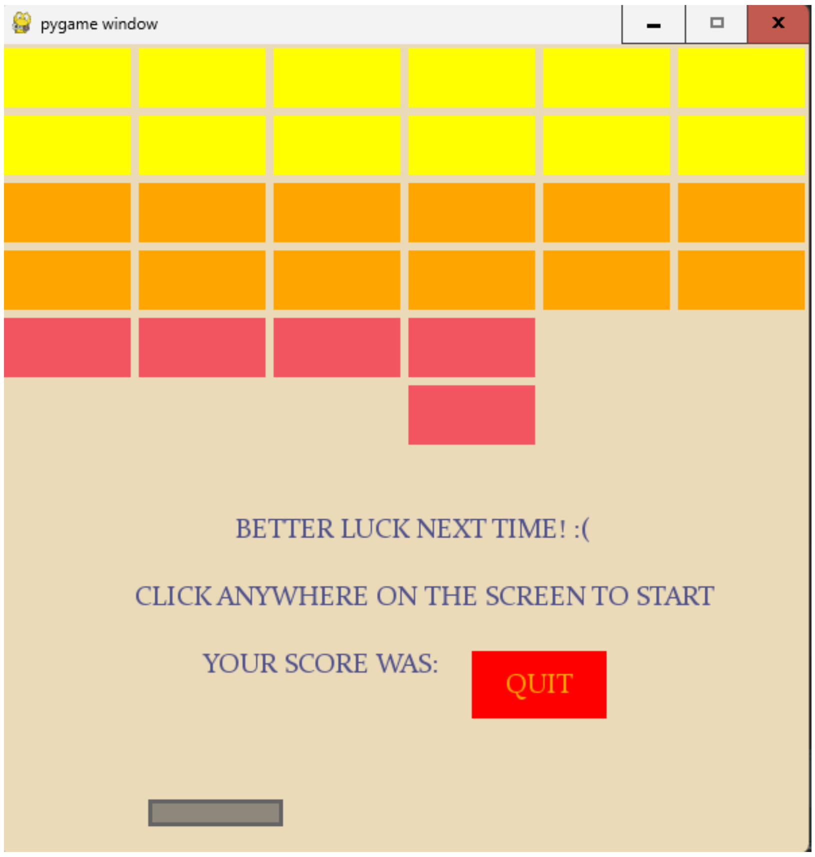
```
def button(msg, x, y, w, h, ic, ac, action = None):

            mouse = pygame.mouse.get_pos()
            click = pygame.mouse.get_pressed()
            ic = pygame.Color('red')
            ac = pygame.Color('yellow')
            if x + w > mouse[0] > x and y + h > mouse[1] > y:
                pygame.draw.rect(screen, ac, (x, y, w, h))

                print(action)

                # click[0] == 1 can be shortened to just click[0] if we're checking for true/false
                if click[0] and action is not None:

                    # Match statement
                    if action == start:
                        start()
                    if action == quitgame:
                        quitgame()

            else:
                pygame.draw.rect(screen, ic, (x, y, w, h))

            smallText = pygame.font.SysFont('Constantia', 20)
            textSurf, textRect = text_objects(msg, smallText)
            textRect.center = (x + (w / 2), y + (h / 2))
            screen.blit(textSurf, textRect)
```

In this screenshot is the code for the two buttons that appear in the main menu. As stated before, one of them can be used to start the game and the other one is used to exit the game. The code above explains what will happen when either of the buttons are clicked by the player. Ideally, when the player clicks the start button, the buttons will clear, and the game will begin as usual. However, there are currently several errors in the code that are preventing the transition from happening. Fortunately, the quit button works as intended so when the player clicks the exit button, a function called quitgame() is called and the game closes.

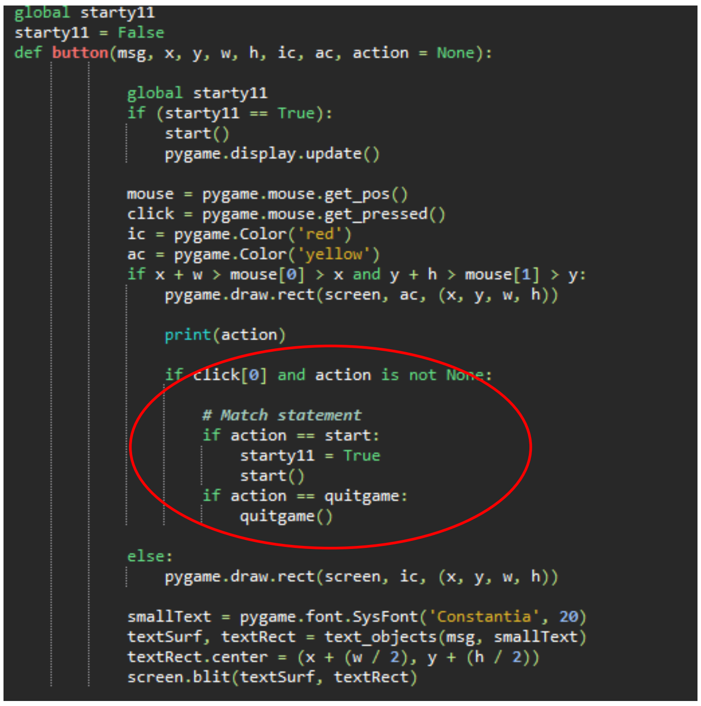
Test Data	Type	Justification	Actual Test/Result
Colour of the text displayed in the menus changed	Valid	Adjusts the colour of the text that the player will see in the menus which may make it easier for the player to read	Valid
Button dimensions being changed to a larger side	Valid	This would make the buttons larger than the original one which could make it easier for the player to see/read.	Valid
Making the size of the text displayed anything other than a positive number e.g. float	Invalid	This would create an error, or it would make the text invisible if it was negative	Invalid

<u>Update on the menu error (6/10/24):</u>

I have reached a new point in solving the error that has risen within the menu system. To begin with, the error was that the start button would not start the game and in fact would do nothing. However, after changing and adding to the code, the start button now does start the game, however the quit button does not disappear along with the start button. This means that while the player plays the game, the quit button is still at the bottom of the screen and is constantly appearing and disappearing every second.


pygame window
BETTER LUCK NEXT TIME! :(
CLICK ANYWHERE ON THE SCREEN TO START
YOUR SCORE WAS:
QUIT

This is what the output currently looks like, and as you can see the quit button is still there and will not disappear when the game is started. This unfortunately means that the player is unable to see their score at the end of the game which is something that is key to the game as the player needs to see their score that they have earned.



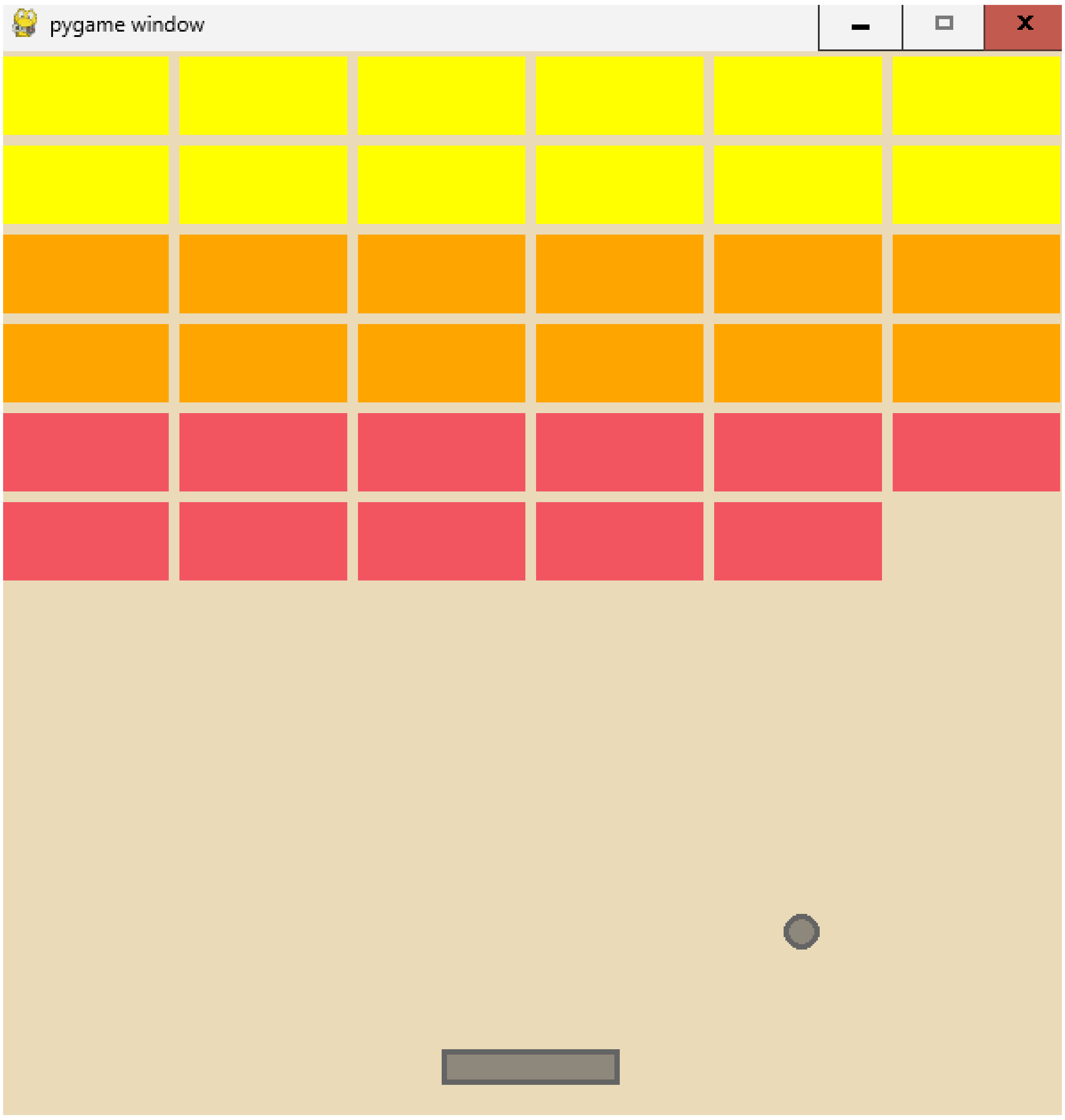
```
global starty11
starty11 = False
def button(msg, x, y, w, h, ic, ac, action = None):

            global starty11
            if (starty11 == True):
                start()
                pygame.display.update()

            mouse = pygame.mouse.get_pos()
            click = pygame.mouse.get_pressed()
            ic = pygame.Color('red')
            ac = pygame.Color('yellow')
            if x + w > mouse[0] > x and y + h > mouse[1] > y:
                pygame.draw.rect(screen, ac, (x, y, w, h))

                print(action)

                if click[0] and action is not None:

                    # Match statement
                    if action == start:
                        starty11 = True
                        start()
                    if action == quitgame:
                        quitgame()

            else:
                pygame.draw.rect(screen, ic, (x, y, w, h))

            smallText = pygame.font.SysFont('Constantia', 20)
            textSurf, textRect = text_objects(msg, smallText)
            textRect.center = (x + (w / 2), y + (h / 2))
            screen.blit(textSurf, textRect)
```

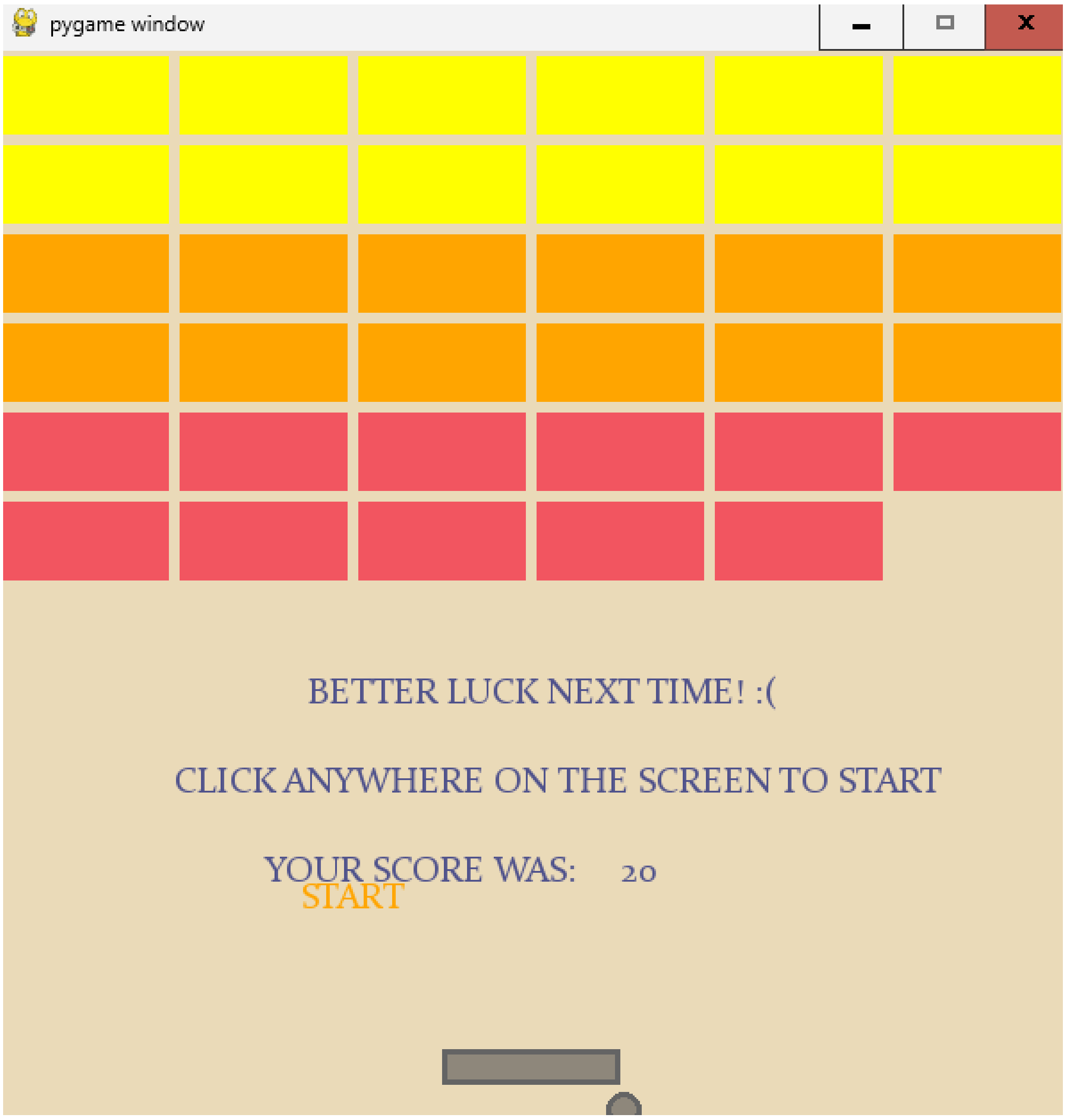
This is the code for the buttons and what actions will occur depending on what buttons have been clicked by the player. As it stands, when the quit button is clicked, the game closes as intended. But when the start button is clicked, the game starts as intended however only the start button disappears and not the quit button. The code uses validation as it needs to check whether the mouse is above the designated areas for the buttons, otherwise when the player clicks somewhere else, nothing happens. It also allows the computer to know when to highlight the buttons as they change colour when the mouse is above them.

I changed this code so that the start button would do what it was supposed to do because originally it would not perform an action and therefore would not allow the player to actually play the game.

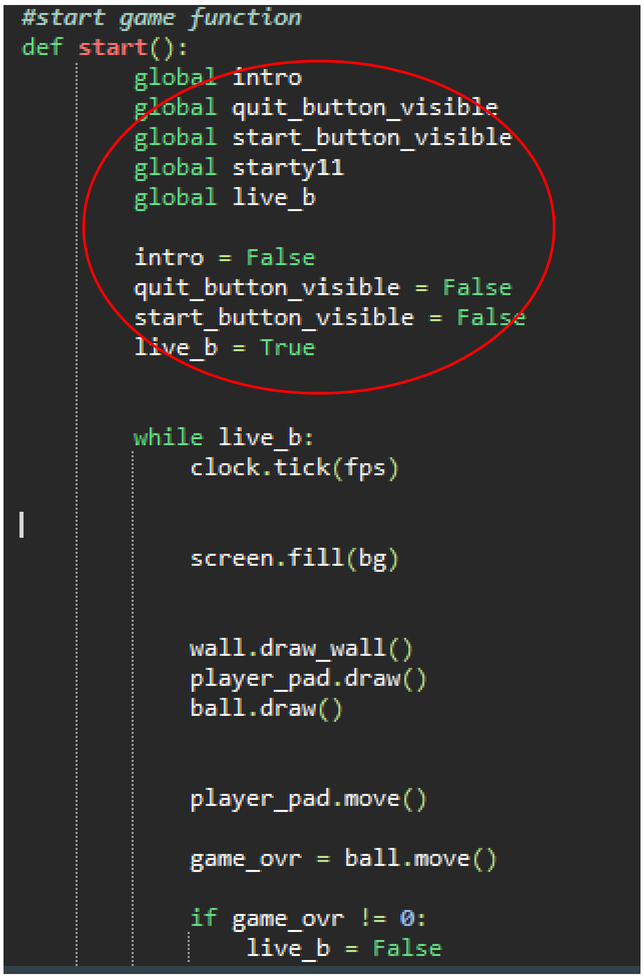
<u>Final update on the menu error:</u>

I have reached the closest point I have come to solving this menu error. After tweaking the code and creating new variables for the visibility of the buttons, where the variables would turn false after the start button was clicked. This has solved the error where when the game started, the start button would not disappear. Therefore, the game now works almost full as intended. However, a small issue that I cannot fix is where the start text on the start button re-appears when the game ends. The screenshots below show the new outputs, the new error, and the code that fixed the main bug.




pygame window
BETTER LUCK NEXT TIME! :(
CLICK ANYWHERE ON THE SCREEN TO START
YOUR SCORE WAS: 20
START

This is the error that I need to fix now that the buttons have been removed when the game starts. When the game ends and the score is displayed, it still draws the text that is written onto the start button at the beginning of the game. After several attempts, I cannot seem to find a fix for this error and therefore due to time constraints, it will unfortunately have to be a part of the final game unless I come across a fix while writing the evaluation of the game.



```
#start game function
def start():
        global intro
        global quit_button_visible
        global start_button_visible
        global starty11
        global live_b

        intro = False
        quit_button_visible = False
        start_button_visible = False
        live_b = True


        while live_b:
            clock.tick(fps)


            screen.fill(bg)


            wall.draw_wall()
            player_pad.draw()
            ball.draw()


            player_pad.move()

            game_ovr = ball.move()

            if game_ovr != 0:
                live_b = False
```

This is the code that I created to solve the start button error. As you can see, I have assigned two new variables to do with the visibility of the buttons. When first assigned, the values of these variables are true so that they appear when called in the game intro function. However, when the start function is called when the start button is clicked, the variables values turn to false therefore hiding them when the game is being played. This is a big step in the code as this error has been a part of the game ever since the menus were implemented. I changed this code because now that I have found a fix that works, the player can play the game without any distractions or things blocking the screen.

Game intro:

```
procedure game_intro()
    intro = True

    while intro = True
        for event in pygame.event.get()
            if event.type == pygame.QUIT then
                quit()

        fill game screen with bg
        largeText = pygame.font.SysFont('Constantia', 60)
        TextSurf, TextRect = text_objects("SMASH PADDLE", largeText)
        TextRect.center = ((screen_w / 2), (screen_h / 2))
        screen.blit(TextSurf, TextRect)
```

This is the current pseudocode for the game intro procedure. I have followed this pseudocode as it allows a simple way to create easy to use buttons that appear clear to the user in the main menu.

```
        button("START", 150, 450, 100, 50, pygame.Color('red'), pygame.Color('yellow'), start)
        button("QUIT", 350, 450, 100, 50, pygame.Color('red'), pygame.Color('yellow'), quitgame)
        pygame.display.update()
        clock.tick(fps)
endprocedure
```

Prototype 5 Review:
Success Criteria:

Requirements	**Justification**	☑	**Reference**
Pause Menu	This allows the user to pause the game if needed without losing their score	☑	Interview – 26/06/24
Easy to use Buttons	Simple buttons for the user to press so it is easier to navigate menus and play the game	☑	Stakeholder and Interview – 14/06/24
How to play menu	To show the user how to play the game if they have never played similar style games before	☐	Interview – 26/06/24

Unfortunately, I was not able to meet all the success criteria set out for the menu system at the beginning of this project. The ‘how to play’ menu has not been included in the game because the current, simpler menu system has come across many errors and therefore I no longer have enough time to add a controls screen to the menu system. On the positive side, 2 of the 3 objectives set out were met with ease.

Firstly, the pause menu code was written well and was simple to code with little to no errors. When the game is running, the player will be able to press the ‘P’ key on their keyboard and the pause menu will come up onto the screen. This will include an option to resume the game to where the player left off and another option to quit the game.

The other requirement was to create easy to use buttons that were clean, simple, and blend in with the retro style of the game. Again, these buttons were simple to code, yet they are a crucial part of the game as they add immersion by making the menu system possible. I have also added a feature so that when the player hovers their mouse over the buttons, the buttons change colour to highlight what the mouse is hovering over. Displayed on the buttons is large, vibrant text clearly and simply explaining what action they will perform if the player clicks them.

Prototype 5 Interview Plan:

- Should the pause menu be accessed in a different way?
- Should the buttons be a different shape or colour in any way?

- Should the title of the game be smaller or a different colour in any way?

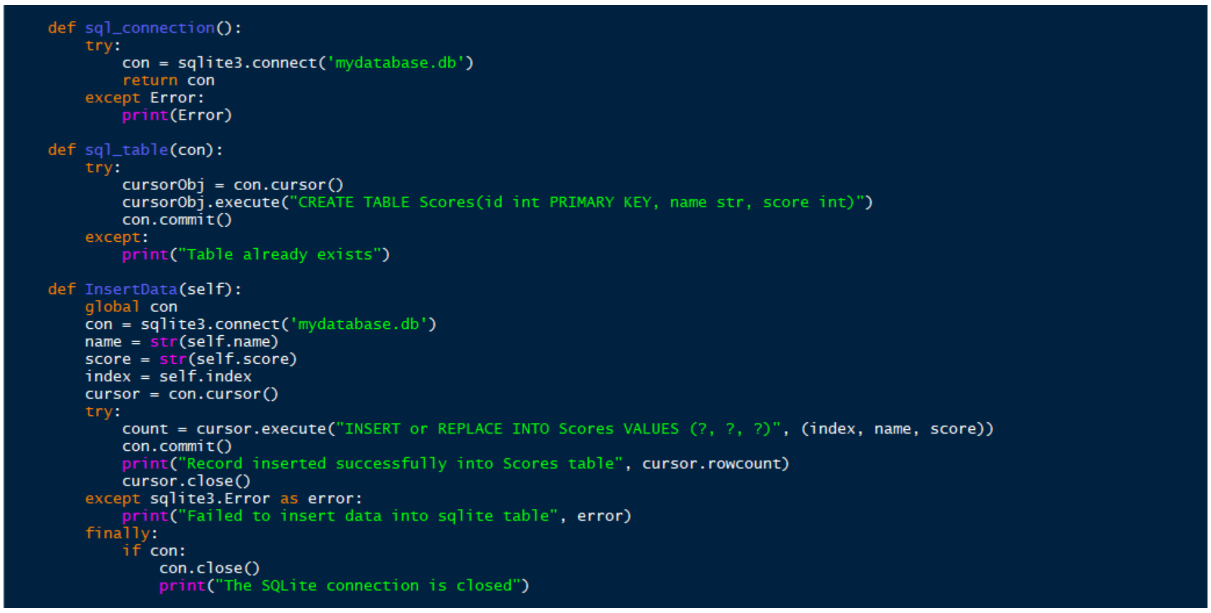
Prototype 5 Interview with Finlay Baker:

- Should the pause menu be accessed in a different way?
  "No, the 'P' key is a good idea as it is easy to see and access."
- Should the buttons be a different shape or colour in any way?
  "I think that the buttons are fine as rectangles, and they are a good colour as they are easy to see for the player."
- Should the title of the game be smaller or a different colour in any way?
  "I think that the size and position of the title is really good although make it brighter so there is no chance of confusion against the background colour."

Conclusion:
The buttons that are drawn onto the screen are very accurate compared to what I set out to create at the beginning of this project. However, my main issue is getting the start button to work as intended and it is currently taking all my attention and time to fix. The addition of highlighting the button when the mouse hovers over it is great as it creates even more immersion and depth to the game. The title being displayed at the top of the screen in large text is an excellent feature as well as it makes the main menu look cleaner and more exciting. Fortunately, towards the end of writing this prototype, I found a fix to my main error with the start button and hiding the buttons when the game begins.

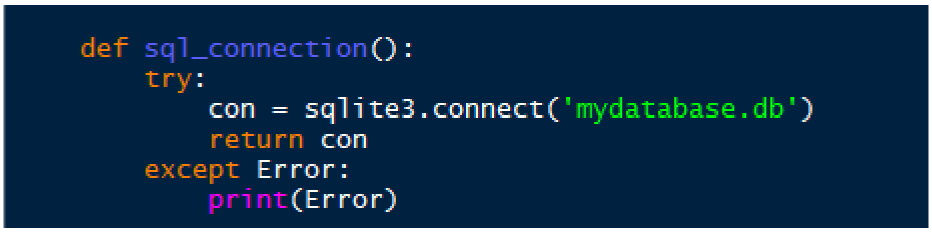
Prototype 6

Creating the database (08/10/24):



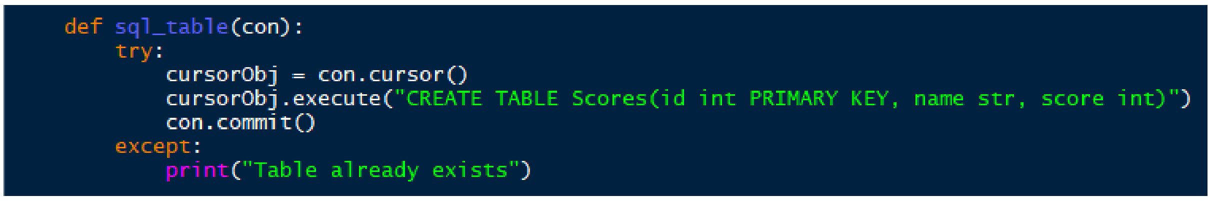
```
def sql_connection():
    try:
        con = sqlite3.connect('mydatabase.db')
        return con
    except Error:
        print(Error)

def sql_table(con):
    try:
        cursorObj = con.cursor()
        cursorObj.execute("CREATE TABLE Scores(id int PRIMARY KEY, name str, score int)")
        con.commit()
    except:
        print("Table already exists")

def InsertData(self):
    global con
    con = sqlite3.connect('mydatabase.db')
    name = str(self.name)
    score = str(self.score)
    index = self.index
    cursor = con.cursor()
    try:
        count = cursor.execute("INSERT or REPLACE INTO Scores VALUES (?, ?, ?)", (index, name, score))
        con.commit()
        print("Record inserted successfully into Scores table", cursor.rowcount)
        cursor.close()
    except sqlite3.Error as error:
        print("Failed to insert data into sqlite table", error)
    finally:
        if con:
            con.close()
            print("The SQLite connection is closed")
```

This is the main code for the SQL database that stores the player's score. The database stores the player's name, which is asked for at the start of the game, and the player's score, which is recorded at the end of the game when the game is over. The code for the database consists of several functions which have uses such as opening and closing the connection with the database, inserting the data into the database, and reading the data recorded.



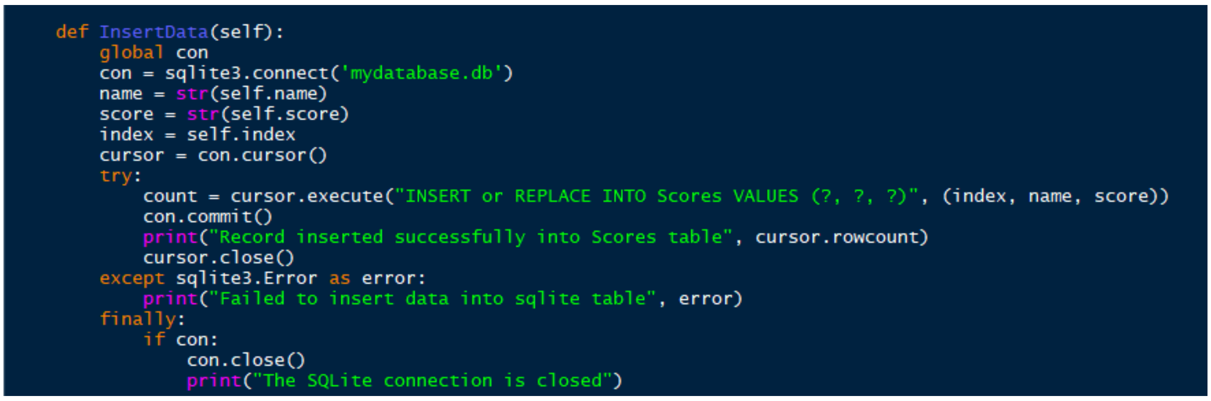
```python
def sql_connection():
    try:
        con = sqlite3.connect('mydatabase.db')
        return con
    except Error:
        print(Error)
```

This code tries to connect to the database called ‘mydatabase.db’. If the computer cannot find the database, it will return an error. The variable ‘con’ is for the connection between the computer and the database. ‘Error’ is imported under the library of ‘sqlite3’.



```python
def sql_table(con):
    try:
        cursorObj = con.cursor()
        cursorObj.execute("CREATE TABLE Scores(id int PRIMARY KEY, name str, score int)")
        con.commit()
    except:
        print("Table already exists")
```

If the computer cannot find the database or the database does not exist yet, it will create a table which will store the player’s name as a string value, and the player’s score as an integer value. If the database and table already exist, it will print out a message saying that the table already exists.



```python
def InsertData(self):
    global con
    con = sqlite3.connect('mydatabase.db')
    name = str(self.name)
    score = str(self.score)
    index = self.index
    cursor = con.cursor()
    try:
        count = cursor.execute("INSERT or REPLACE INTO Scores VALUES (?, ?, ?)", (index, name, score))
        con.commit()
        print("Record inserted successfully into Scores table", cursor.rowcount)
        cursor.close()
    except sqlite3.Error as error:
        print("Failed to insert data into sqlite table", error)
    finally:
        if con:
            con.close()
            print("The SQLite connection is closed")
```

This is the function for inserting data into the actual database. First, the computer establishes a connection with the created database. Then, it will take three values: the player’s index, the player’s score, and the player's name. When the computer gets hold of all three values, it will insert them into the database into their respective columns. If the values have been inserted into the database successfully, the computer will print a message saying that they have been inserted into the database successfully. If not, the computer will display a different message saying that they have not been inserted into the database. Finally, the computer closes the connection with the database after either of the messages have been displayed.

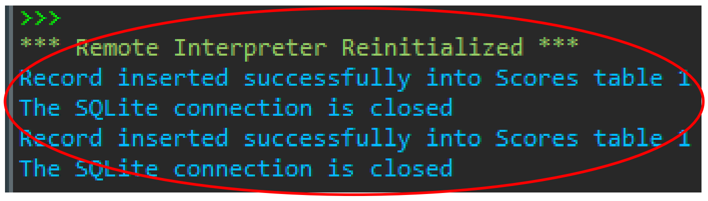
Testing:

What is being tested?	Input	Justification	Expected Outcome	Actual Outcome
Data being inserted into the database	Player name and score after the game ends	To see if the computer inserts the data into the database	The values will be inserted into the database successfully	The values are inserted into the database

Checking if the database exists	Clicking the start button so that the game and the database code runs	To see if the database already exists so that the computer won't make endless databases	The computer will display a message saying if it already exists or if it has been created	Corresponding message displayed successfully
Checking if the messages about the database are printed.	Clicking the start button so that the game and database code runs	To make sure that the computer tells the player anything about the database in case of any errors	The correct message will be printed for each situation	Corresponding message displayed successfully

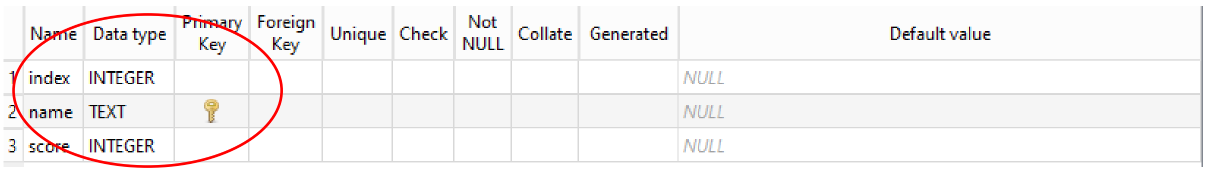
Errors:

At the current moment, there is a logic error with the game code which causes the database to update with the values endlessly, essentially meaning that the database works except it will infinitely add the same data to it until the game is closed. This is a screenshot of the printed messages after the game has finished.

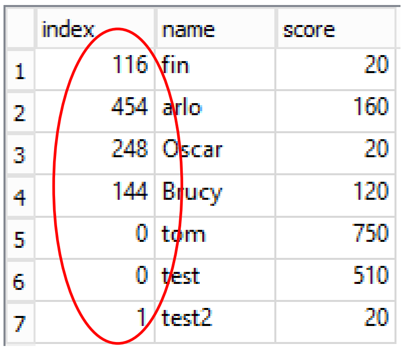


```
>>>
*** Remote Interpreter Reinitialized ***
Record inserted successfully into Scores table 1
The SQLite connection is closed
Record inserted successfully into Scores table 1
The SQLite connection is closed
```

As you can see, the computer repeatedly adds the same value to the database when it should only add one. To help solve this problem, I made the name value in the database the primary key value so that if two scores have the same name tied to them, it will only show the highest score in the database. I changed the code so that the database would not be extremely large and so that it would be kept tidy because otherwise the database would eventually become too large. This is a screenshot of the database and the values inside of it after the modifications so the table values.

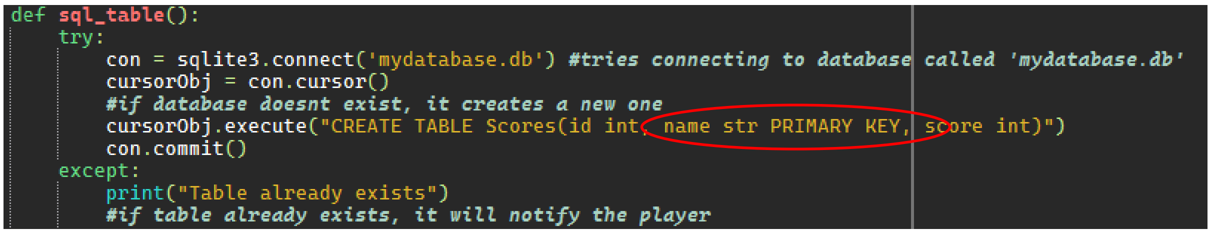


	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	index	INTEGER								NULL
2	name	TEXT	🔑							NULL
3	score	INTEGER								NULL



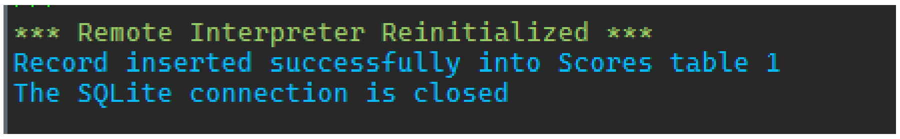
	index	name	score
1	116	fin	20
2	454	arlo	160
3	248	Oscar	20
4	144	Brucy	120
5	0	tom	750
6	0	test	510
7	1	test2	20

As you can see, the database will only show one score for each name as the primary key is the name value in the table. This means that while the loop is still in effect, it will only show one value, hence the large index number as that is how many times it repeated before the game was closed.



```
def sql_table():
    try:
        con = sqlite3.connect('mydatabase.db') #tries connecting to database called 'mydatabase.db'
        cursorObj = con.cursor()
        #if database doesnt exist, it creates a new one
        cursorObj.execute("CREATE TABLE Scores(id int, name str PRIMARY KEY, score int)")
        con.commit()
    except:
        print("Table already exists")
        #if table already exists, it will notify the player
```

This is the new code and as you can see, it creates the 'name' column as the primary key so that it will only take the highest score for a name and wont show two scores for the same name.



```
*** Remote Interpreter Reinitialized ***
Record inserted successfully into Scores table 1
The SQLite connection is closed
```

This is the new printed output and as you can see, it is not repeating as I have added a time.sleep code so that if it does repeat, it will take 10,000 seconds to repeat once.

Acceptance Table:

No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome

1.	Updating the high score in the database.	Valid:
		Insertdata("Arlo," 69)
		Invalid:
		insertdata(9, 69)
			Valid:
			The high score will update in the database
			Invalid:
			The database will be updated, and an error will be printed out
				Works as expected

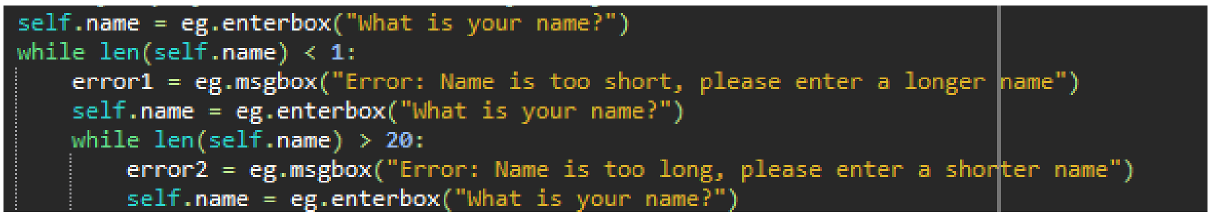
19.	Score will be saved to the database	Valid:
		InsertData("Arlo", 100)
		Invalid:
		InsertData(0)
			Valid:
			Score and name are entered into the database and shown when the table is viewed
			Invalid:
			Data is not inserted into the database.
				Works as expected

Database Pseudocode:

```
function sql_connection():
    try:
      con = sqlite3.connect('mydatabase.db')
      return con
    except Error:
        print(Error)
endfunction
  function sql_table():
    try:
        con = sqlite3.connect('mydatabase.db')
        cursorObj = con.cursor()
        cursorObj.execute("CREATE TABLE Scores(id int PRIMARY KEY, name str, score int)")
        con.commit()
    except:
        print("Table already exists")
endfunction
  function InsertData(self):
    global con
    con = sqlite3.connect('mydatabase.db')
    name = str(self.name)
    score = str(self.score)
    index = self.index
    cursor = con.cursor()
    try:
        count = cursor.execute("INSERT or REPLACE INTO Scores VALUES (?, ?, ?)", (index, name,
score))
        con.commit()
        print("Record inserted successfully into Scores table", cursor.rowcount)
        cursor.close()
    except sqlite3.Error as error:
        print("Failed to insert data into sqlite table", error)
    finally:
        if con:
            con.close()
            print("The SQLite connection is closed")
        endif
```

This is the pseudocode for the database, it will require a few changes after the evaluation has been completed and when I have more time. I have followed this pseudocode because it is a simple yet effective way to create the database for the game and it makes it easier for the user to know what is going on with the database as well endfunction

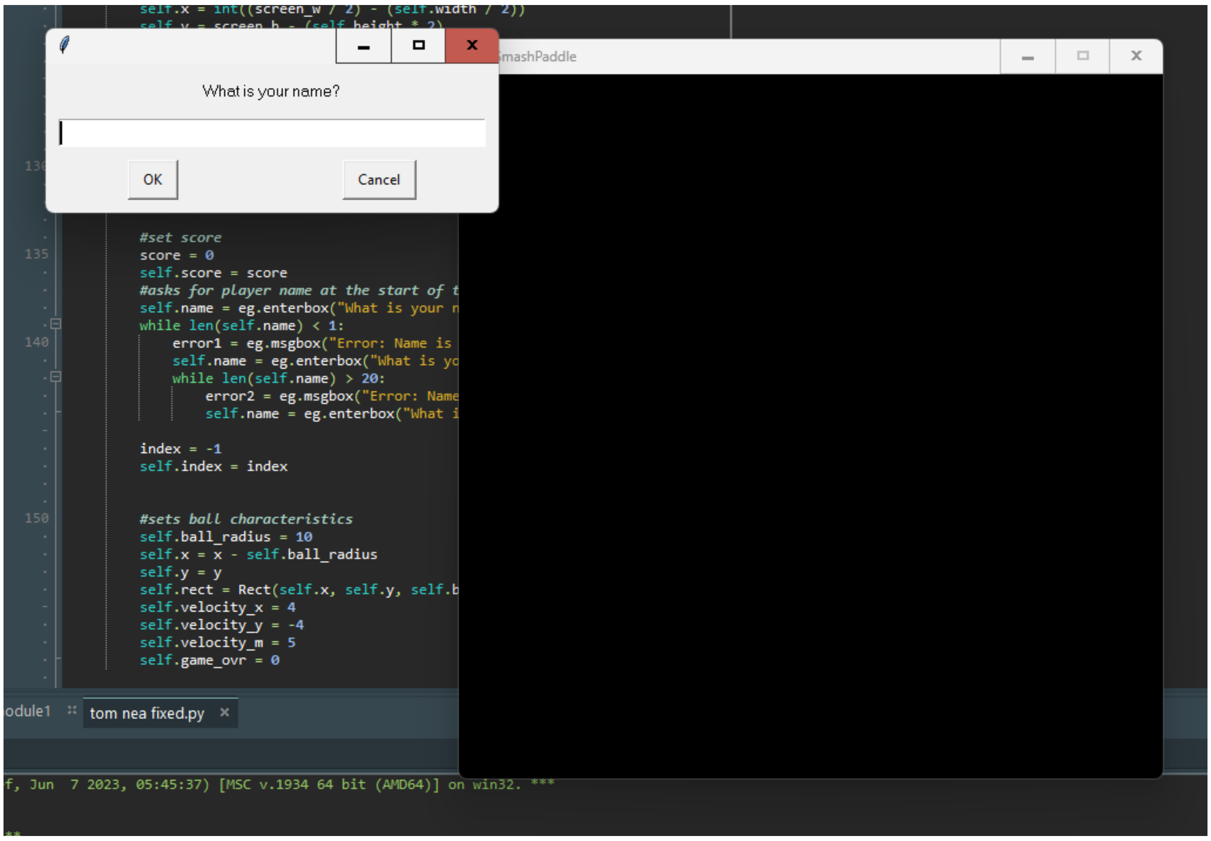
<u>Creating an input box for username (12/10/24):</u>



```
self.name = eg.enterbox("What is your name?")
while len(self.name) < 1:
    error1 = eg.msgbox("Error: Name is too short, please enter a longer name")
    self.name = eg.enterbox("What is your name?")
    while len(self.name) > 20:
        error2 = eg.msgbox("Error: Name is too long, please enter a shorter name")
        self.name = eg.enterbox("What is your name?")
```

This code asks the user to enter a name before the main menu appears. It uses validation to make sure that the user cannot enter a name longer than 20 characters or a name shorter than 1 character. So when the game begins, a box appears that prompts the user to enter a username that will be stored along with the score value that they earn during the game.

The screenshot below shows what the box looks like when the game is started.



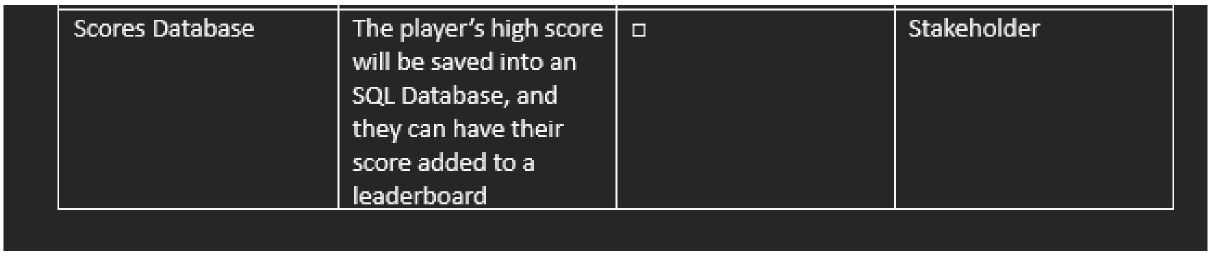
What is your name?

```
        #set score
        score = 0
        self.score = score
        #asks for player name at the start of t
        self.name = eg.enterbox("What is your n
        while len(self.name) < 1:
            error1 = eg.msgbox("Error: Name is
            self.name = eg.enterbox("What is yo
            while len(self.name) > 20:
                error2 = eg.msgbox("Error: Name
                self.name = eg.enterbox("What i

        index = -1
        self.index = index


        #sets ball characteristics
        self.ball_radius = 10
        self.x = x - self.ball_radius
        self.y = y
        self.rect = Rect(self.x, self.y, self.b
        self.velocity_x = 4
        self.velocity_y = -4
        self.velocity_m = 5
        self.game_ovr = 0
```

```
f, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32. ***
```

Prototype 6 Review:

Success criteria:



Scores Database	The player's high score will be saved into an SQL Database, and they can have their score added to a leaderboard	☐	Stakeholder

Unfortunately like several other aspects of the game, the code for the database need work as the database currently does not work as intended. However, personally I am happy with the current state as the database works, and the values can be uploaded into the database. Whilst it may not work as fully intended, a lot of the main objectives set out for coding the database were met including being able to create/open/close a database, store the player's name and to store the player's high score.

Prototype 6 Interview Plan:

- Should the database table have more columns?
- Should the player be asked for their name before or after the game starts?
- Should there be a leaderboard of some sorts displayed as a separate menu so that the player can essentially see the values in the database to see who has the highest scores?

Prototype 6 Interview with Finlay Baker:

- Should the database table have more columns?
  "No, there is not a lot else to store besides the player's name and score."
- Should the player be asked for their name before or after the game starts?
  "I think that it won't matter too much when they are asked for their name but keeping it as it is where the player is asked at the start is fine."
- Should there be a leaderboard of some sorts displayed as a separate menu so that the player can essentially see the values in the database to see who has the highest scores?
  "I think that if you have enough time to create something like that, then it would be an amazing addition to the game for not only immersion, but to make the game look cleaner. However, it will require a lot of time and will most likely cause even more errors than there currently are."

Conclusion:
Overall, I think that the database prototype was one of the trickier codes to write as I have had no previous experience with coding an SQL database and storing new values that aren't set. Whilst there may be a few logic errors with the code, it still does the basic actions and those are the most important things that I wanted to get out of this prototype. I think that if I had more time, I would code a leaderboard section of the menu where the player could view all the high scores earned by other players.

The input box that prompts the player to enter a username was a great addition as it not only adds immersion to the game but it also creates a database column that stores those names so the database is not just a table full of scores with no names attached to them.

As for all the prototypes, I personally believe that they were very successful. Despite there being countless errors throughout creating this project and there still are, this has allowed me to understand the code better as a whole and has deepened my understanding in solving those errors too.

Evaluation:

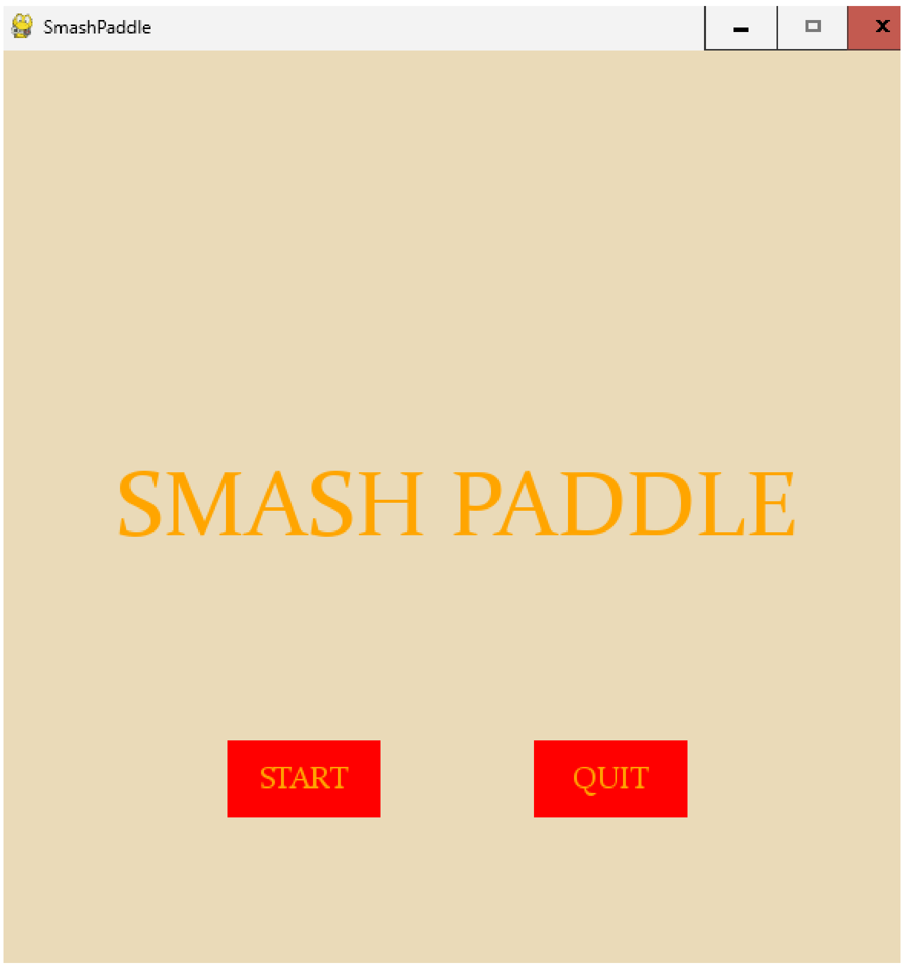
Post Development Testing

Throughout the process of creating this project, I have been using validation so any data that has been marked as 'Invalid' should not be accepted by the game when it is ran.

Main Menu

No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
6.	The button click function	Valid: Left click on the button Invalid: Right click on the button	Valid: Game will start Invalid: Nothing will happen	Works as expected
10.	Exit button	Valid: Left click the exit button Invalid: Right click on the exit button	Valid: The game should shut down. Invalid: Nothing should happen at all	Works as expected
18.	How to play button	Valid: Left click on the button Invalid: Right click on the button	Valid: The player is taken to the how to play section where they are shown the controls Invalid: Nothing should happen	Due to time constraints, this feature will not be implemented into the final game.

Output:

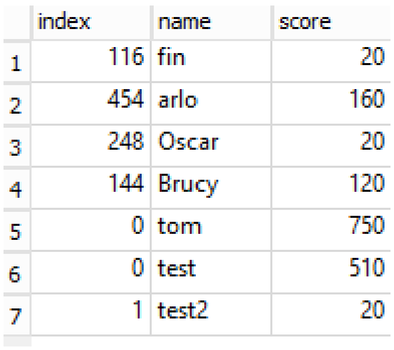


Scoring

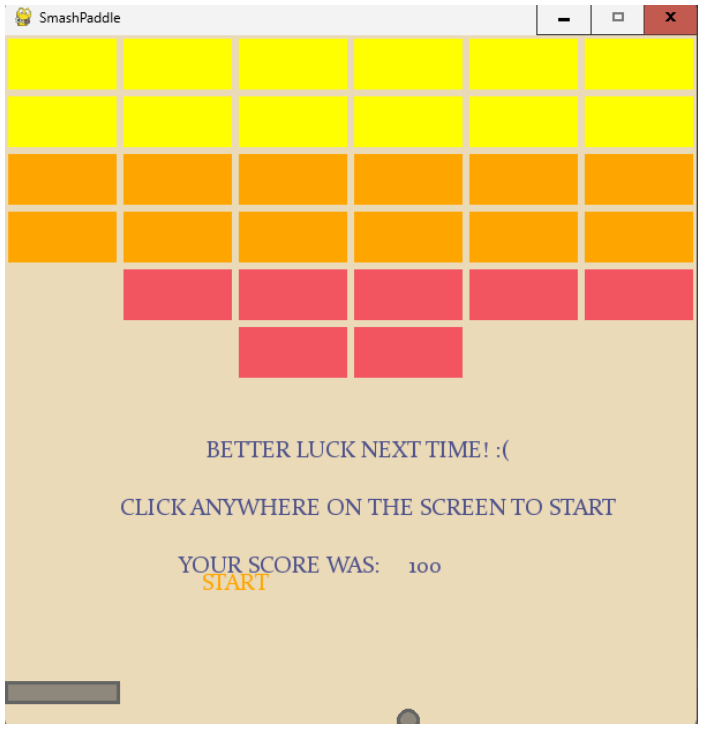
No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
1.	Updating the high score in the database.	Valid: Insertdata("Arlo," 69) Invalid: insertdata(9, 69)	Valid: The high score will update in the database Invalid: The database will be updated, and an error will be printed out	Works as expected
19.	Score will be saved to the database	Valid: InsertData("Arlo", 100) Invalid: InsertData(0)	Valid: Score and name are entered into the database and shown when the table is viewed Invalid: Data is not inserted into the database.	Works as expected

2.	The score to be updated as the player destroys the blocks	Valid: Score = block strength *5 Invalid: The player either hits a wall or has died	Valid: The score to be updated on the screen reflecting the blocks that they have destroyed Invalid: The score to stop being updated and only showing the player's current/last score	Works as expected
3.	The score being displayed at the bottom-middle of the screen	n/a	The score to be towards the lower middle of the screen	Works as expected but only displayed at end of game
4.	The high score being showed to the player when the game is over or in the main menu or the pause menu	n/a	The high score to be displayed to the player when the game is over, in the pause menu and in the main menu also	Due to time constraints this feature was unable to be implemented
14.	Checking if there is a new high score	Valid: High score < score Invalid: High score > score	Valid: The high score variable and the database will be updated. Invalid: The high score variable and the database won't be updated	Works as expected

Outputs:



	index	name	score
1	116	fin	20
2	454	arlo	160
3	248	Oscar	20
4	144	Brucy	120
5	0	tom	750
6	0	test	510
7	1	test2	20


SmashPaddle
BETTER LUCK NEXT TIME! :(
CLICK ANYWHERE ON THE SCREEN TO START
YOUR SCORE WAS: 100
START

Player

No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
5.	The player to stay inside the game screen if not the game is over.	Valid: Player x position is within the game boundaries Invalid: Player x position is outside the game boundaries	Valid: The game to continue until the game is over Invalid: The game will run into a logic error and the paddle will get stuck.	Works as expected

Sound effects

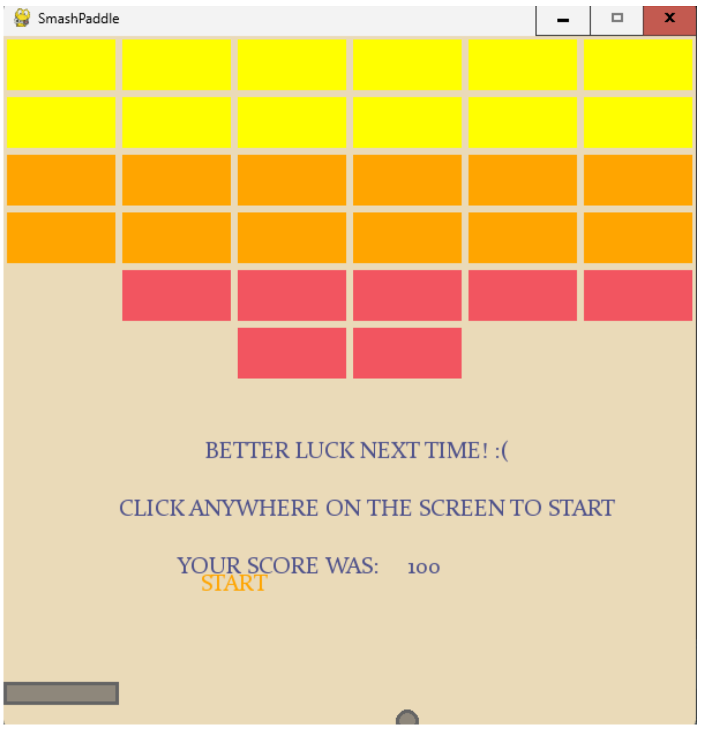
No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
7.	Sound effect when the ball bounces	Valid: If the ball has collided with a wall or a block that won't been destroyed or the paddle	Valid: A sound effect will play after the bounce	Due to time constraints, I was unable to add sound effects to the game.
8.	Sound effect when the ball breaks a block	Valid: If the ball has broken a block when collided	Valid: A sound effect should play when the block breaks	

9.	Sound effect when the ball misses the paddle, and the game is over	Valid: If the ball has missed the paddle and game_ovr = 1	Valid: A sound effect should play after the ball hits the bottom of the game screen	

Obstacles:

No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
15.	Ball destroying a block in the wall	Valid: The ball collides with the block and destroys it Invalid: The ball does not destroy the block	Valid: If the ball hit the block and destroys it, it will disappear from the row. Invalid: The block stays in the row of blocks	Works as expected
16.	All the blocks have been destroyed	Valid: The rows of blocks have disappeared, and the game is over Invalid: Not all the blocks are gone, and the game is still running	Valid: game_ovr = 1 and the player is sent back to the main menu with their score Invalid: Nothing will happen until the game is over	Works as expected however when the game is over, the game stops and a message and score is displayed instead
17.	The ball missing the paddle and going off screen	Valid: the ball hits the bottom of the screen and the game stops Invalid: The game continues to run as normal	Valid: game_ovr = 1 and the player is sent back to the main menu Invalid: Nothing should happen as the game should still be running as usual	Works as expected but again, when game is over, the game stops and a message is displayed with the player's final score.

<u>Outputs:</u>



BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

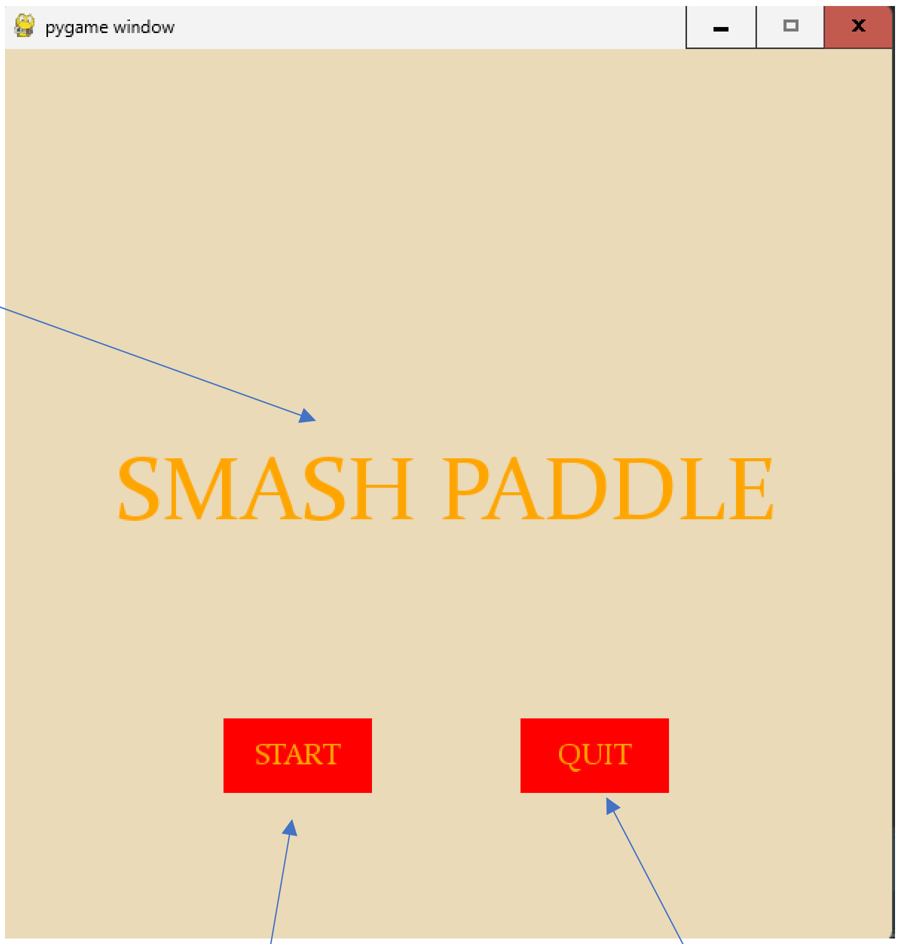
YOUR SCORE WAS: 100

Pause Menu

No.	What is being tested?	Inputs	Expected Outcome	Actual Outcome
11.	Pause menu key pressed	Valid: The 'P' key is pressed Invalid: Any other letter key is pressed	Valid: The pause menu will appear. Invalid: Nothing should happen	Due to time constraints and ongoing errors with the main menu system, the pause menu will not be implemented despite having been able to project a code for it.
12.	Resume button	Valid: The 'P' key is pressed again Invalid: Any other letter key is pressed	Valid: The game should resume as normal. Invalid: Nothing should happen	
4.	The high score being showed to the player when the game is over or in the main menu or the pause menu	n/a	The high score to be displayed to the player when the game is over, in the pause menu and in the main menu also	

Usability Features

Main menu



Name of the game displayed on the screen

By clicking on this button, it starts the game.

By clicking on this button, it closes the entire game.

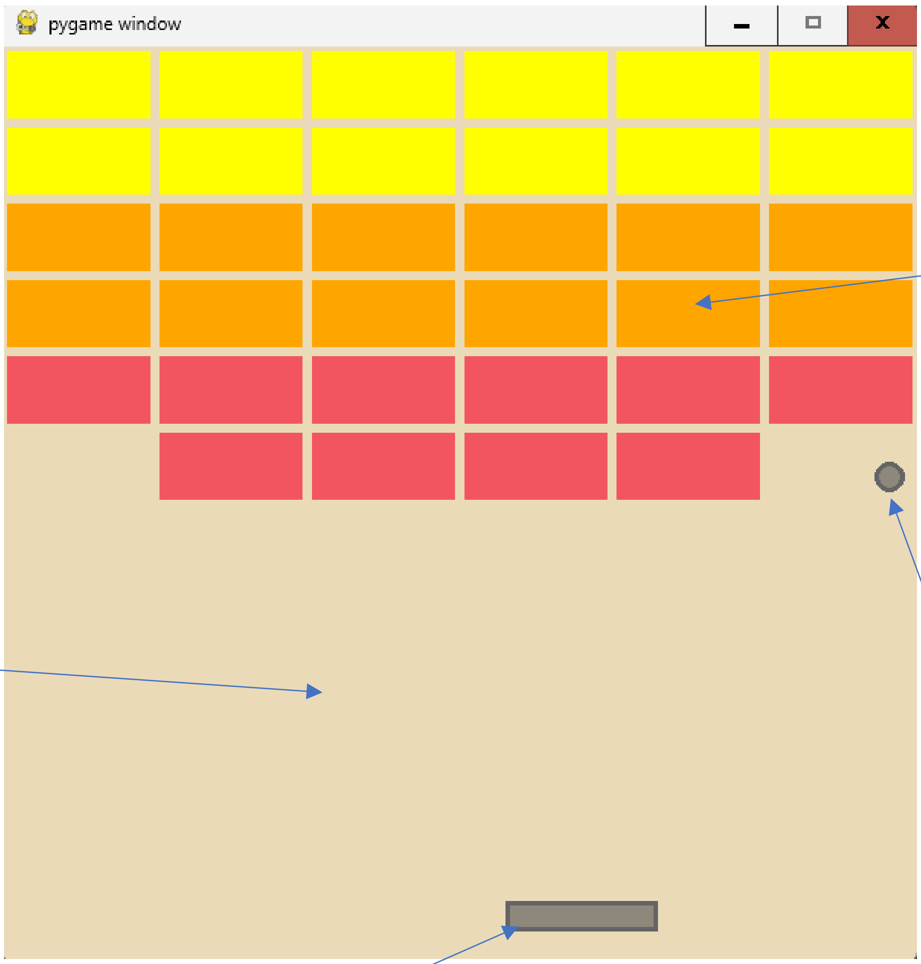
Interview with Finlay Baker:

From researching games for ideas at the start of this project, I have found that creating usable buttons heavily improves accessibility. I have created two working buttons that allows the player to start or exit the game. At the beginning of this project, I set out to create a how to play button and a whole other menu along with a separate menu for seeing the high scores. However, due to time constraints, I was unable to add these features into the final game.

Client's response:

"I am very fond of the minimalist style of the main menu. It makes the game look clean, and the buttons are easy to see and read. The title displayed in the middle of the screen is a great feature as well."

Game state



The background is filled with a simple colour so that the drawn objects that the player uses are easy to see against the background. It also allows the text that will appear at the end of the game to be easier to read as well.

Here you can see the wall of blocks as well as the ball. Each two rows of blocks are different colours so that the player can tell which are higher strengths than the others. They are vibrant colours, so they are easy to see. The ball is again a different colour to the background and blocks so that it is also easy to see.

This is the paddle that the player uses to hit the ball towards the wall of blocks. It is a different colour to the background so that the player won't get them confused with each other.

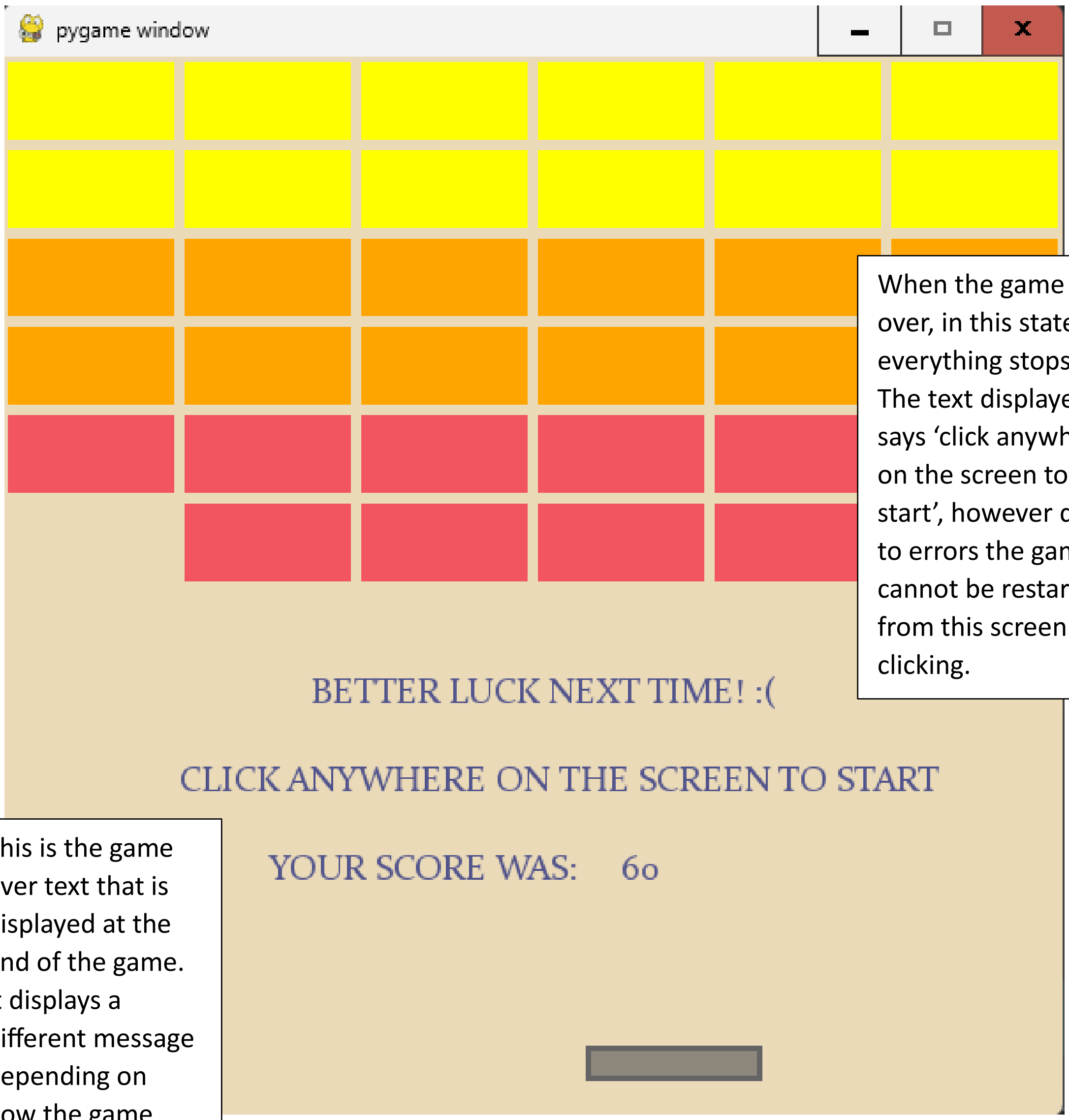
Interview with Finlay Baker:

This is the game state, the game runs until all the blocks have been destroyed or the ball hits the bottom of the screen. This state includes all the features of the main game such as the paddle, ball and the wall of blocks. Each object has contrasting colours so that they are easy to tell what each object is and how large it is. This also makes the game look more pleasing and easier to use.

Client's response:

"I like the idea of the contrasting colours for the objects as it fits with the style of retro and colourful, but it also still fits with the minimalist style too. I obviously think that the game state would be much better had the error of the start button not disappearing was fixed, however ignoring errors the game state works great."

Game over screen



BETTER LUCK NEXT TIME! :(

CLICK ANYWHERE ON THE SCREEN TO START

YOUR SCORE WAS: 60

When the game is over, in this state everything stops. The text displayed says 'click anywhere on the screen to start', however due to errors the game cannot be restarted from this screen by clicking.

This is the game over text that is displayed at the end of the game. It displays a different message depending on how the game ended along with the player's score.

Interview with Finlay Baker:

This is the game over screen. It appears once the game has ended either by the player destroying all the blocks in the wall or the player letting the ball hit the bottom of the screen. It displays the game over message along with the score that the player had earned that game. This screen is crucial as it allows the player to see their score so they know what they need to beat, which therefore will make them want to play again and therefore increase time spent playing this game.

Client's response:

"I like the messages that are displayed at the end of each game depending on how the game ended as it adds immersion. I also think it's crucial that the score was displayed at the end so that the player knows what they have achieved."

Success Criteria Evaluation

Menus

Requirements	Justification	☑	Reference
Main Menu	This allows the player to be able to play the game.	☑	Interview – 14/06/24
Pause Menu	This allows the user to pause the game if needed without losing their score	☐	Interview – 26/06/24
How to play menu	To show the user how to play the game if they have never played similar style games before	☐	Interview – 26/06/24
Easy to use Buttons	Simple buttons for the user to press so it is easier to navigate menus and play the game	☑	Stakeholder and Interview – 14/06/24
Pixel Art/Minimalist Texture for each menu screen	To make the game look more appealing and retro	☑	Stakeholder

I have implemented a main menu that is simple to use and contains two buttons that are easily accessible as well as the game title being displayed in the middle of the screen. The buttons highlight when the mouse hovers over them and they change colour. When a button is clicked, an action is performed and the game either closes, or the game starts. The menu is a minimalistic style as with the rest of the game and the buttons are simple yet clean. If I had more time on this project, I would implement the how to play and pause menus as well as adding a section to see the current high scores.

Game mechanics

Requirements	Justification	☑	Reference
A score based on how many blocks/what blocks they destroyed	This allows the user to see their progress and how well they are doing	☑	Feature of "Atari Breakout" and "Space Invaders"
Ball Bouncing Mechanic	This is what allows the player to destroy the blocks and effectively play the game	☑	Feature of "Atari Breakout"
Realistic Ball Bounces	This makes the ball realistic and able to predict where it will bounce	☑	Feature of "Atari Breakout"
Smooth animations and gameplay	This will make the game more satisfying to play but also it will make the game look cleaner and well made	☑	Stakeholder
Different levels of blocks	Makes the game more immersive as it adds levels of depth and challenge	☑	Interview – 14/06/24
Scores Database	The player's high score will be saved into an SQL Database, and they can have their score added to a leaderboard	☑	Stakeholder

I have successfully completed all the main objectives I set out for the game mechanics at the beginning of this project. I have implemented a working score system after many trials and errors and in the final game it works as well as I hoped it would. The ball bouncing mechanic works smoothly, and the player can easily anticipate where the ball would bounce and can put their trust in it to bounce like it would in real life. The gameplay itself is smooth and polished ignoring errors and runs comfortably at 120 fps. The wall of blocks works as three groups of two rows in a grid of 6x6 blocks. The bottom two rows are the weakest blocks, the middle two rows require two hits and the top row requiring three hits as they are the strongest blocks. The difference in colours for the blocks to signify their strength helps to add immersion into the game. Finally, the database system works as expected where the player is asked for a name at the beginning of the game, and at the end their score along with the name is stored into a database where only the highest scores for each name are stored. Whilst the success criteria met is simple and there are several additions that I would make if I had more time on this project, I still believe that this part of the project was very successful.

Appearance and sounds

Requirements	Justification	☑	Reference
Game over screen	This will display the user's final score and then direct them to the main menu	☑	Interview – 26/06/24
Relevant sound effects for the game	This is needed to make the game more immersive and fun to play	☐	Interview – 14/06/24
Game over sound effect	To notify the user that the game is now over and that they can quit or restart	☐	Interview – 26/06/24
Square Screen	To make the window fit with the way that the game is drawn and orientated	☑	Feature of "Atari Breakout" and "Space Invaders"
Ball Bouncing Sound	To make the game more immersive and responsive	☐	Feature of "Atari Breakout"
Block Breaking Sound	To make the game more fun and immersive	☐	Feature of "Atari Breakout"
Paddle contact sound	Overall to make the game more satisfying by adding contact sound effects	☐	Feature of "Atari Breakout"
Level Cleared Screen	Congratulates the player when they have cleared the screen of blocks, so they have as sense of achievement	☑	Interview – 26/06/24

These are the features that tie the game together and create a lot of the immersion that the player will feel when playing the game. Unfortunately, I was unable to add any sound effects to the final game due to time constraints. If I had any more time to complete the project, I would go back and add sound effects as I feel this would 'complete' the game and would completely immerse the player when they are playing the game. The separate screens for when the game is over, and the player has won gives the player a sense of achievement but also an inspiration to maybe beat their score if they are able to improve on it. These were also the features that were most important to my client as these are what gives the game a good impression so that people would want to play the game without seeing any gameplay beforehand and therefore, I have worked very closely with my client when deciding what would be the best next step to take when it came to working on these features.

Controls

Requirements	Justification	☑	Reference
Only Keyboard Controls	This will make the game more accessible so that people of all ages and abilities can play the game	☑ *+ mouse controls for menus*	Interview – 14/06/24

I have successfully implemented both mouse and keyboard controls so that the player can navigate the menus and press buttons using the mouse, and when playing the game, they can use the keyboard which makes the game accessible for any ages.

Maintenance

Here I will suggest how further maintenance could be carried out and how missing features of the game could be added in.

Pause Menu	This allows the user to pause the game if needed without losing their score	☐	Interview – 26/06/24
How to play menu	To show the user how to play the game if they have never played similar style games before	☐	Interview – 26/06/24
Leaderboard menu	To show the user the current leaderboard and other player's scores.	☐	Interview – 26/06/24

These menus could be easily implemented had there been a larger time frame to complete this project. It would simply require almost the exact same code for my main menu except you would need to just change the text written onto the buttons and you would also need to create a pause key to access the pause menu – I had the 'P' key on the keyboard planned as the pause button in case the menu was implemented. The how to play menu would follow the same code and when the button is clicked it needs to take the player to a new screen. I think that a leaderboard menu could also be a great addition as well as a how to play menu screen. You could create a leaderboard menu by following the same code for the main menu and buttons, while obviously changing the text to say leaderboard, and then on the new screen you would have to write a code in the sql functions that calls the values from the database and displays them onto the screen by drawing text as with the title and text everywhere else in the game.

Relevant sound effects for the game	This is needed to make the game more immersive and fun to play	☐	Interview – 14/06/24

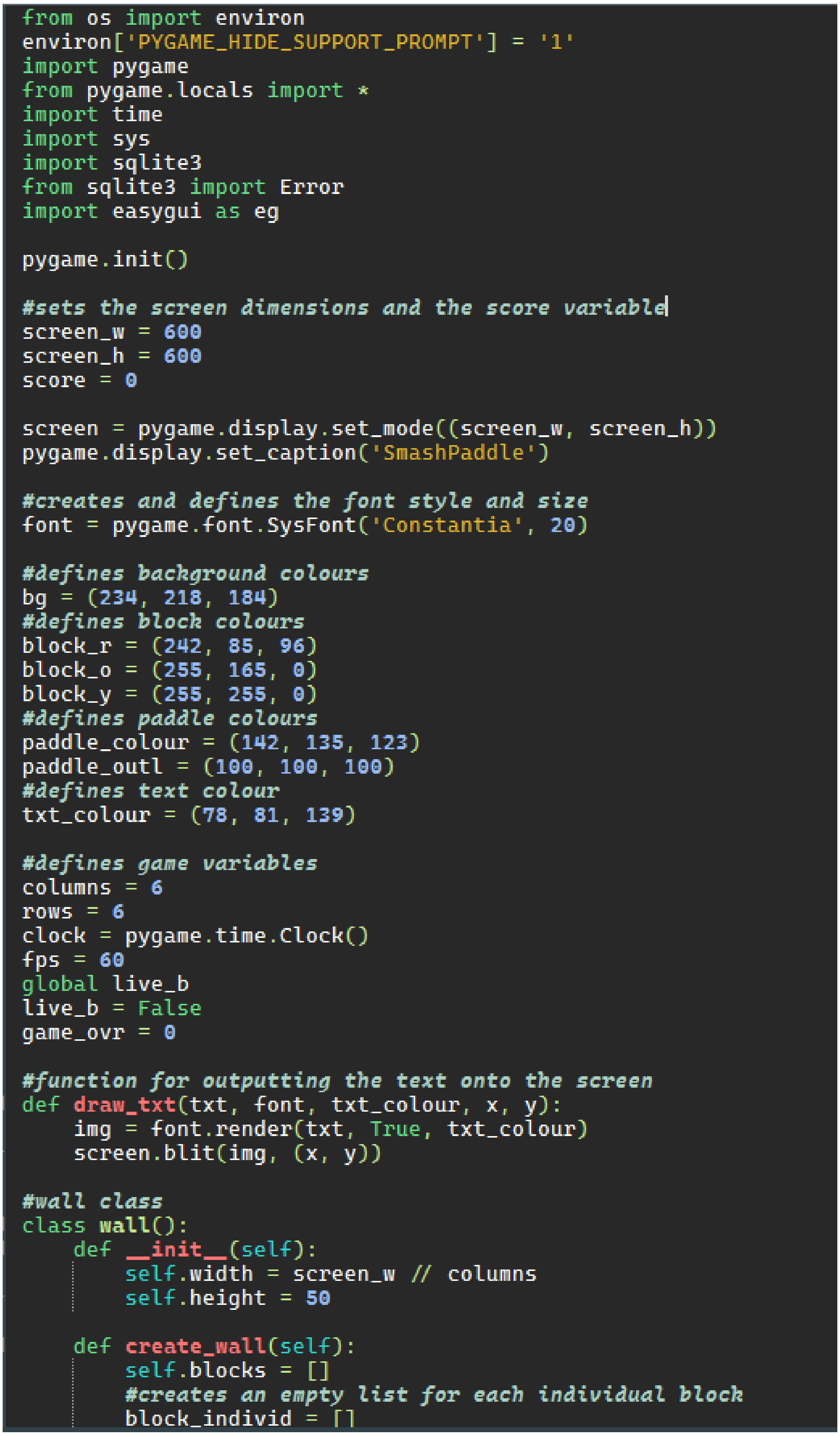
Game over sound effect	To notify the user that the game is now over and that they can quit or restart	☐	Interview – 26/06/24
Ball Bouncing Sound	To make the game more immersive and responsive	☐	Feature of “Atari Breakout”
Block Breaking Sound	To make the game more fun and immersive	☐	Feature of “Atari Breakout”
Paddle contact sound	Overall to make the game more satisfying by adding contact sound effects	☐	Feature of “Atari Breakout”

To add these sound effects, you would need to use the ‘mixer’ library to allow you to play music, and then you would need to source the music you wanted to play or create it yourself. Then it would be a simple case of calling the sound to play every time an action occurs such as when the ball hits or destroys a block.

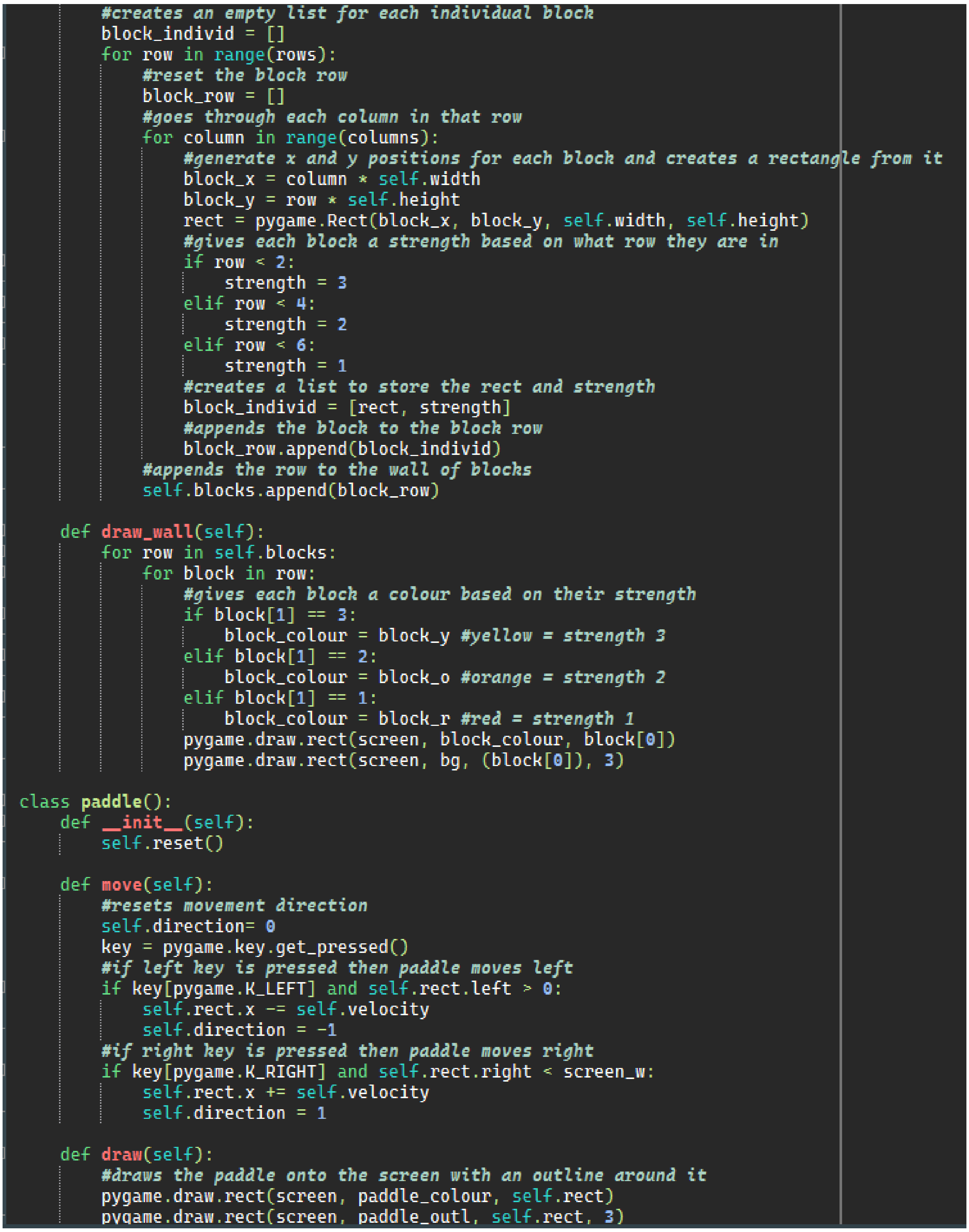
Once all these main features have been implemented, there would still be several bugs to fix that have been occurring since the beginning of the project. These would be needed to be fixed by myself, and to do this I would personally use a website that you can send proof of the bug and the code where the error occurs, and then check the website later to see if there are any possible fixes to the bugs. Once fixed, I will be able to send out an updated version that no longer contains these errors. Another possibility is to have someone else fix the errors for me, and that is why I have added comments to my code so that it is clear to see where everything is so that they can quickly find the areas that need to be looked at. If after the updates players still find errors, I would create a forum online so that players can share their problems with the game and then I would be to test them for myself and fix any errors.

Project Appendixes

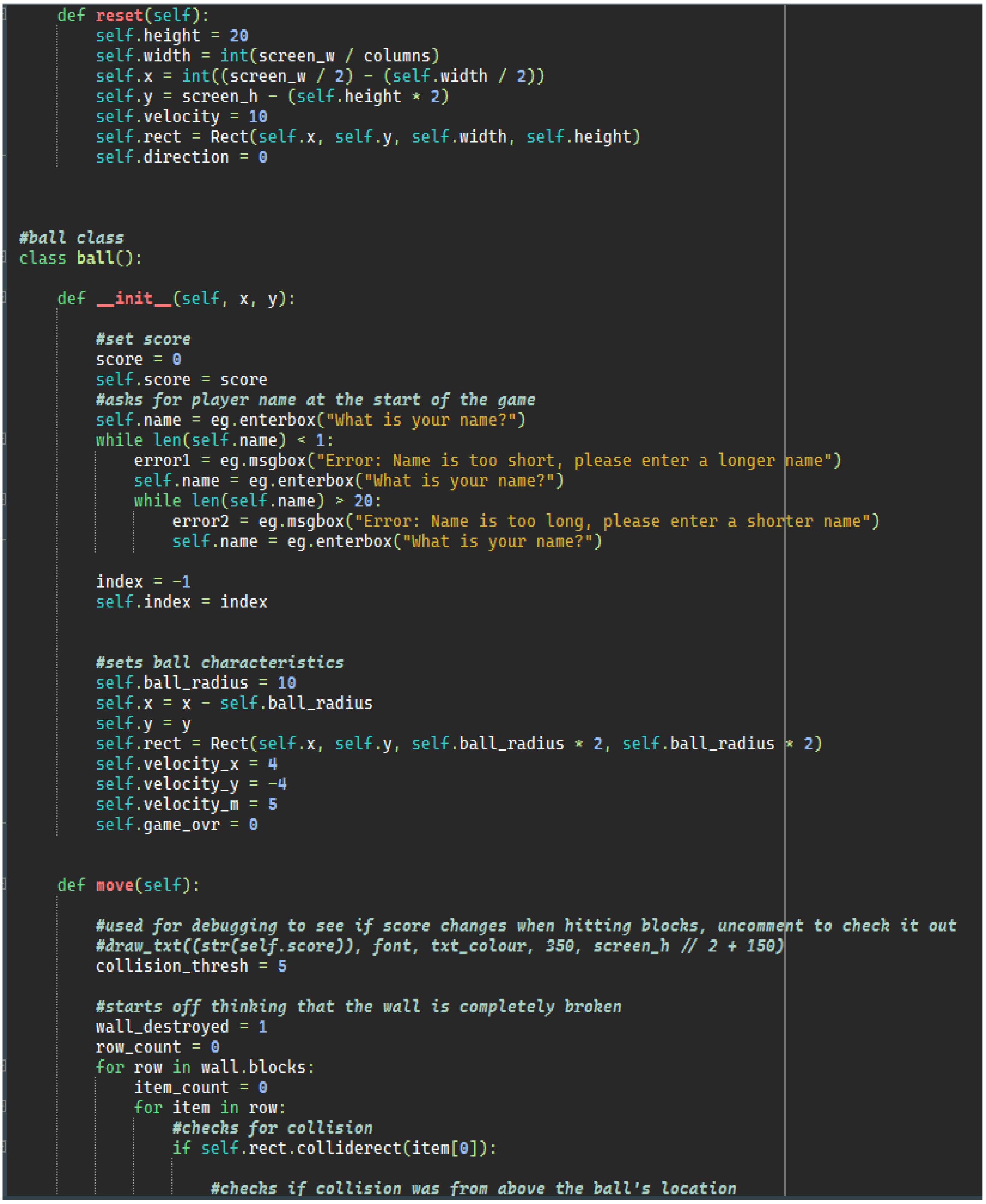
Code listings



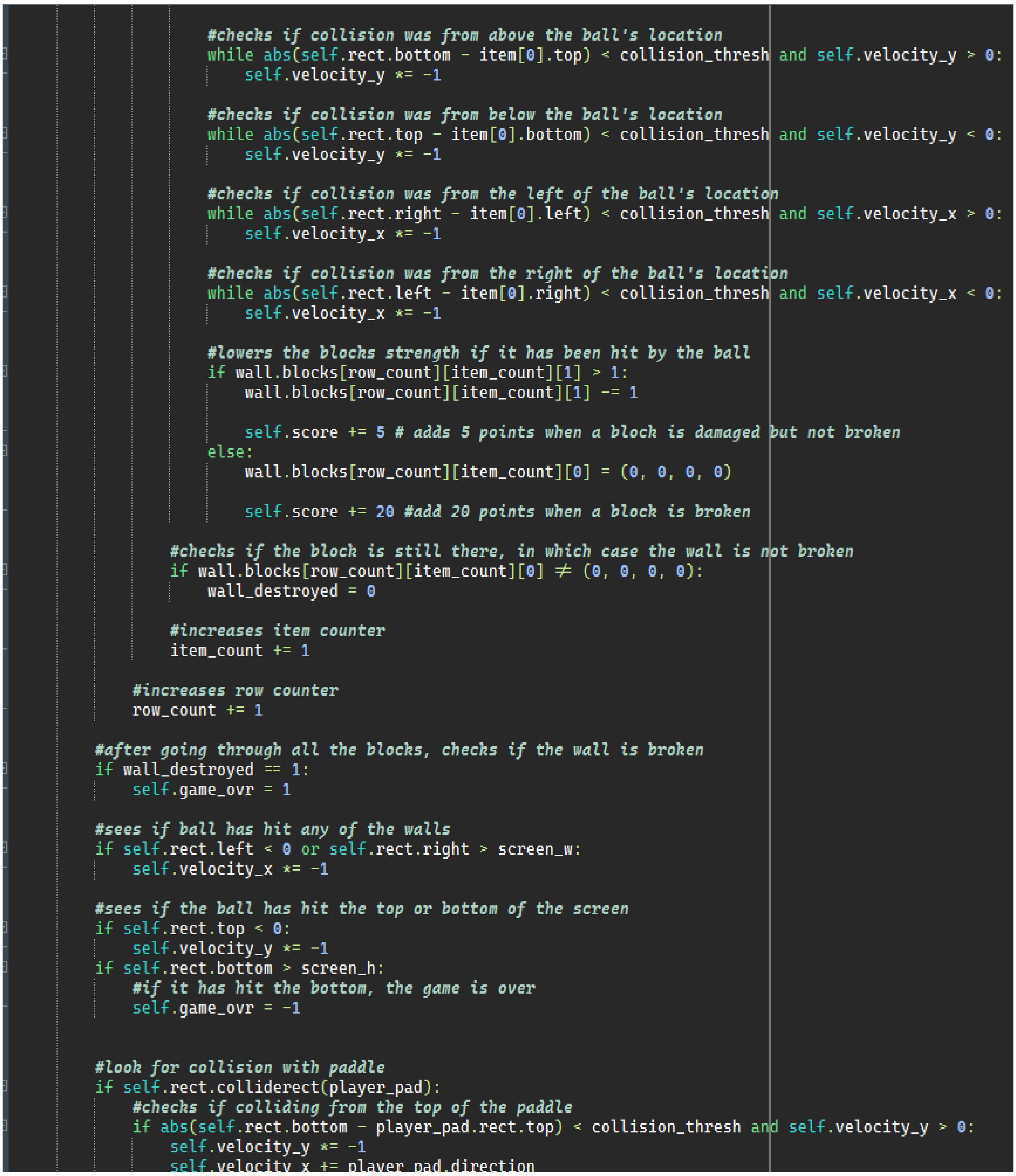
```
from os import environ
environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1'
import pygame
from pygame.locals import *
import time
import sys
import sqlite3
from sqlite3 import Error
import easygui as eg

pygame.init()

#sets the screen dimensions and the score variable
screen_w = 600
screen_h = 600
score = 0

screen = pygame.display.set_mode((screen_w, screen_h))
pygame.display.set_caption('SmashPaddle')

#creates and defines the font style and size
font = pygame.font.SysFont('Constantia', 20)

#defines background colours
bg = (234, 218, 184)
#defines block colours
block_r = (242, 85, 96)
block_o = (255, 165, 0)
block_y = (255, 255, 0)
#defines paddle colours
paddle_colour = (142, 135, 123)
paddle_outl = (100, 100, 100)
#defines text colour
txt_colour = (78, 81, 139)

#defines game variables
columns = 6
rows = 6
clock = pygame.time.Clock()
fps = 60
global live_b
live_b = False
game_ovr = 0

#function for outputting the text onto the screen
def draw_txt(txt, font, txt_colour, x, y):
    img = font.render(txt, True, txt_colour)
    screen.blit(img, (x, y))

#wall class
class wall():
    def __init__(self):
        self.width = screen_w // columns
        self.height = 50

    def create_wall(self):
        self.blocks = []
        #creates an empty list for each individual block
        block_individ = []
```



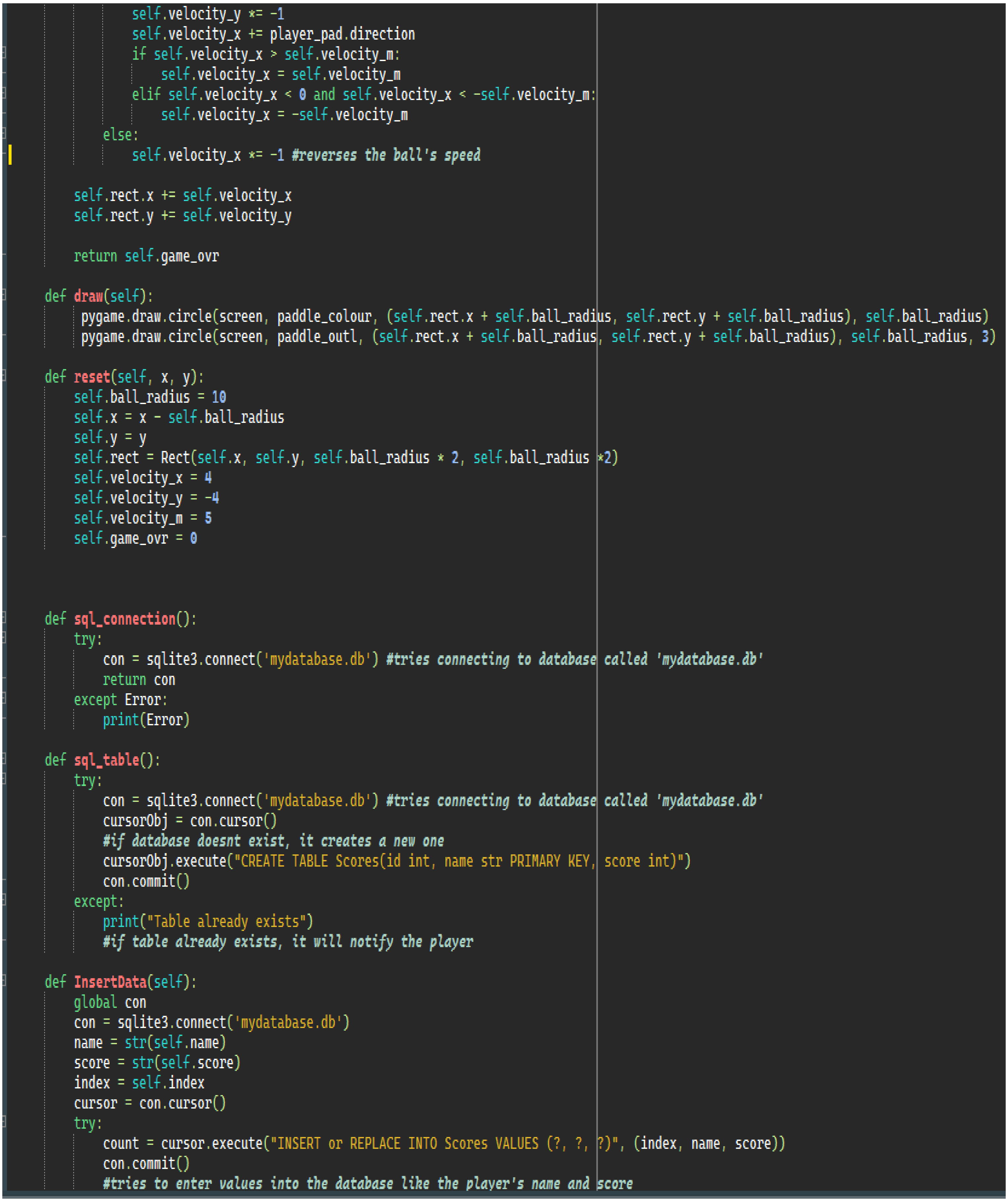
```
        #creates an empty list for each individual block
        block_individ = []
        for row in range(rows):
            #reset the block row
            block_row = []
            #goes through each column in that row
            for column in range(columns):
                #generate x and y positions for each block and creates a rectangle from it
                block_x = column * self.width
                block_y = row * self.height
                rect = pygame.Rect(block_x, block_y, self.width, self.height)
                #gives each block a strength based on what row they are in
                if row < 2:
                    strength = 3
                elif row < 4:
                    strength = 2
                elif row < 6:
                    strength = 1
                #creates a list to store the rect and strength
                block_individ = [rect, strength]
                #appends the block to the block row
                block_row.append(block_individ)
            #appends the row to the wall of blocks
            self.blocks.append(block_row)

    def draw_wall(self):
        for row in self.blocks:
            for block in row:
                #gives each block a colour based on their strength
                if block[1] == 3:
                    block_colour = block_y #yellow = strength 3
                elif block[1] == 2:
                    block_colour = block_o #orange = strength 2
                elif block[1] == 1:
                    block_colour = block_r #red = strength 1
                pygame.draw.rect(screen, block_colour, block[0])
                pygame.draw.rect(screen, bg, (block[0]), 3)

class paddle():
    def __init__(self):
        self.reset()

    def move(self):
        #resets movement direction
        self.direction= 0
        key = pygame.key.get_pressed()
        #if left key is pressed then paddle moves left
        if key[pygame.K_LEFT] and self.rect.left > 0:
            self.rect.x -= self.velocity
            self.direction = -1
        #if right key is pressed then paddle moves right
        if key[pygame.K_RIGHT] and self.rect.right < screen_w:
            self.rect.x += self.velocity
            self.direction = 1

    def draw(self):
        #draws the paddle onto the screen with an outline around it
        pygame.draw.rect(screen, paddle_colour, self.rect)
        pygame.draw.rect(screen, paddle_outl, self.rect, 3)
```



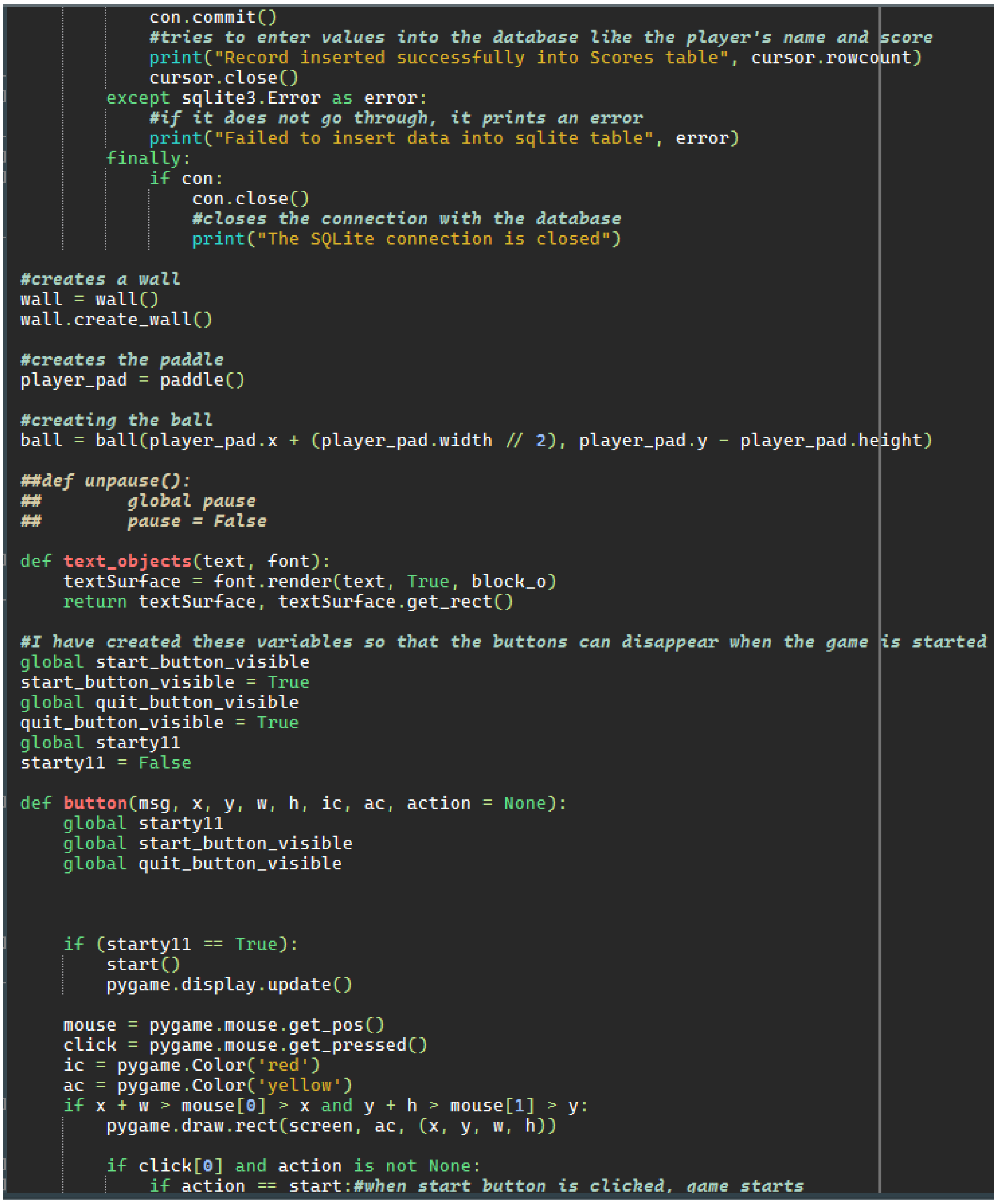
```
    def reset(self):
        self.height = 20
        self.width = int(screen_w / columns)
        self.x = int((screen_w / 2) - (self.width / 2))
        self.y = screen_h - (self.height * 2)
        self.velocity = 10
        self.rect = Rect(self.x, self.y, self.width, self.height)
        self.direction = 0



#ball class
class ball():

    def __init__(self, x, y):

        #set score
        score = 0
        self.score = score
        #asks for player name at the start of the game
        self.name = eg.enterbox("What is your name?")
        while len(self.name) < 1:
            error1 = eg.msgbox("Error: Name is too short, please enter a longer name")
            self.name = eg.enterbox("What is your name?")
            while len(self.name) > 20:
                error2 = eg.msgbox("Error: Name is too long, please enter a shorter name")
                self.name = eg.enterbox("What is your name?")

        index = -1
        self.index = index


        #sets ball characteristics
        self.ball_radius = 10
        self.x = x - self.ball_radius
        self.y = y
        self.rect = Rect(self.x, self.y, self.ball_radius * 2, self.ball_radius * 2)
        self.velocity_x = 4
        self.velocity_y = -4
        self.velocity_m = 5
        self.game_ovr = 0


    def move(self):

        #used for debugging to see if score changes when hitting blocks, uncomment to check it out
        #draw_txt((str(self.score)), font, txt_colour, 350, screen_h // 2 + 150)
        collision_thresh = 5

        #starts off thinking that the wall is completely broken
        wall_destroyed = 1
        row_count = 0
        for row in wall.blocks:
            item_count = 0
            for item in row:
                #checks for collision
                if self.rect.colliderect(item[0]):

                    #checks if collision was from above the ball's location
```



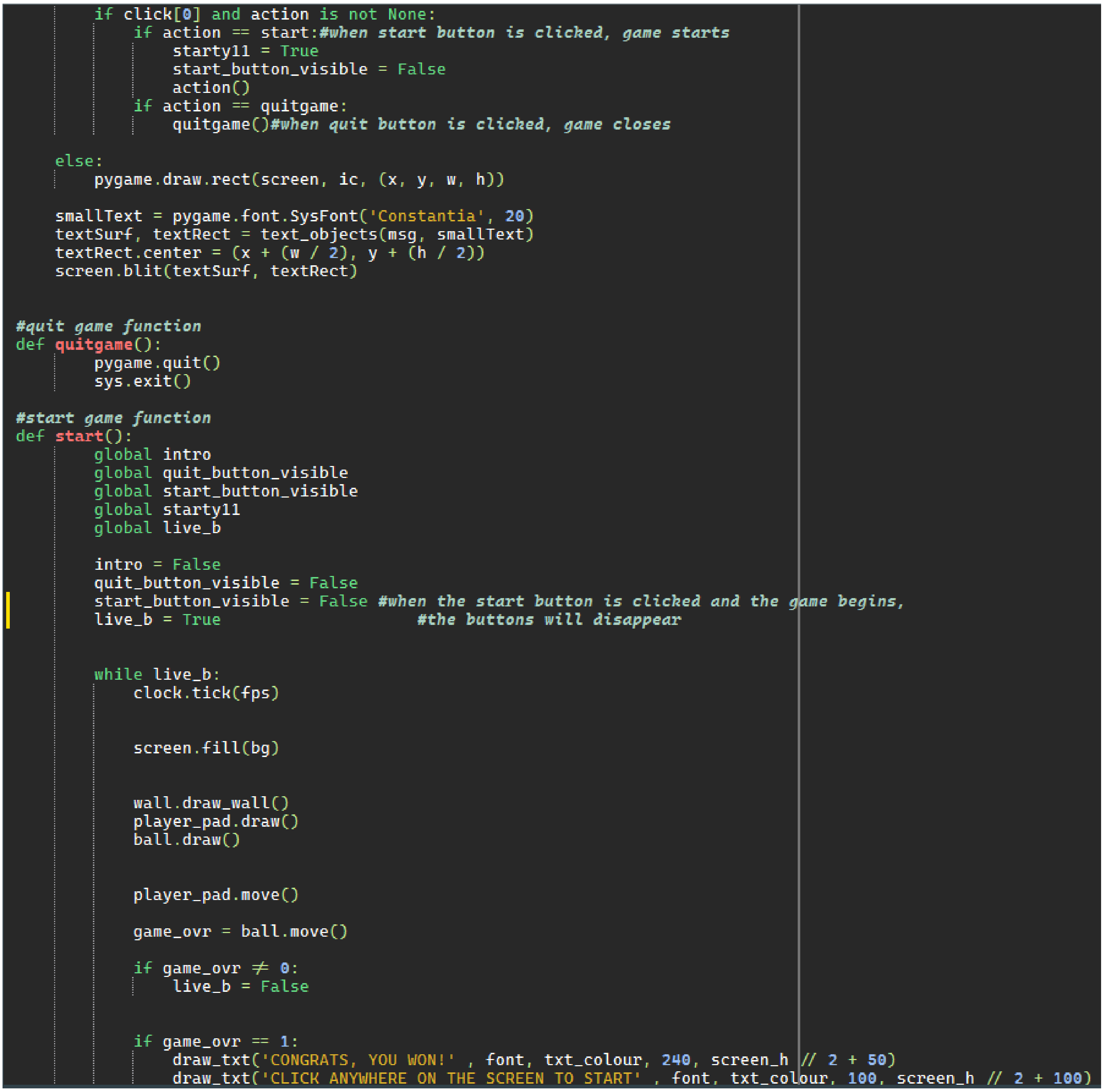
```
                    #checks if collision was from above the ball's location
                    while abs(self.rect.bottom - item[0].top) < collision_thresh and self.velocity_y > 0:
                        self.velocity_y *= -1

                    #checks if collision was from below the ball's location
                    while abs(self.rect.top - item[0].bottom) < collision_thresh and self.velocity_y < 0:
                        self.velocity_y *= -1

                    #checks if collision was from the left of the ball's location
                    while abs(self.rect.right - item[0].left) < collision_thresh and self.velocity_x > 0:
                        self.velocity_x *= -1

                    #checks if collision was from the right of the ball's location
                    while abs(self.rect.left - item[0].right) < collision_thresh and self.velocity_x < 0:
                        self.velocity_x *= -1

                    #lowers the blocks strength if it has been hit by the ball
                    if wall.blocks[row_count][item_count][1] > 1:
                        wall.blocks[row_count][item_count][1] -= 1

                        self.score += 5 # adds 5 points when a block is damaged but not broken
                    else:
                        wall.blocks[row_count][item_count][0] = (0, 0, 0, 0)

                        self.score += 20 #add 20 points when a block is broken

                #checks if the block is still there, in which case the wall is not broken
                if wall.blocks[row_count][item_count][0] != (0, 0, 0, 0):
                    wall_destroyed = 0

                #increases item counter
                item_count += 1

            #increases row counter
            row_count += 1

        #after going through all the blocks, checks if the wall is broken
        if wall_destroyed == 1:
            self.game_ovr = 1

        #sees if ball has hit any of the walls
        if self.rect.left < 0 or self.rect.right > screen_w:
            self.velocity_x *= -1

        #sees if the ball has hit the top or bottom of the screen
        if self.rect.top < 0:
            self.velocity_y *= -1
        if self.rect.bottom > screen_h:
            #if it has hit the bottom, the game is over
            self.game_ovr = -1


        #look for collision with paddle
        if self.rect.colliderect(player_pad):
            #checks if colliding from the top of the paddle
            if abs(self.rect.bottom - player_pad.rect.top) < collision_thresh and self.velocity_y > 0:
                self.velocity_y *= -1
                self.velocity_x += player_pad.direction
```



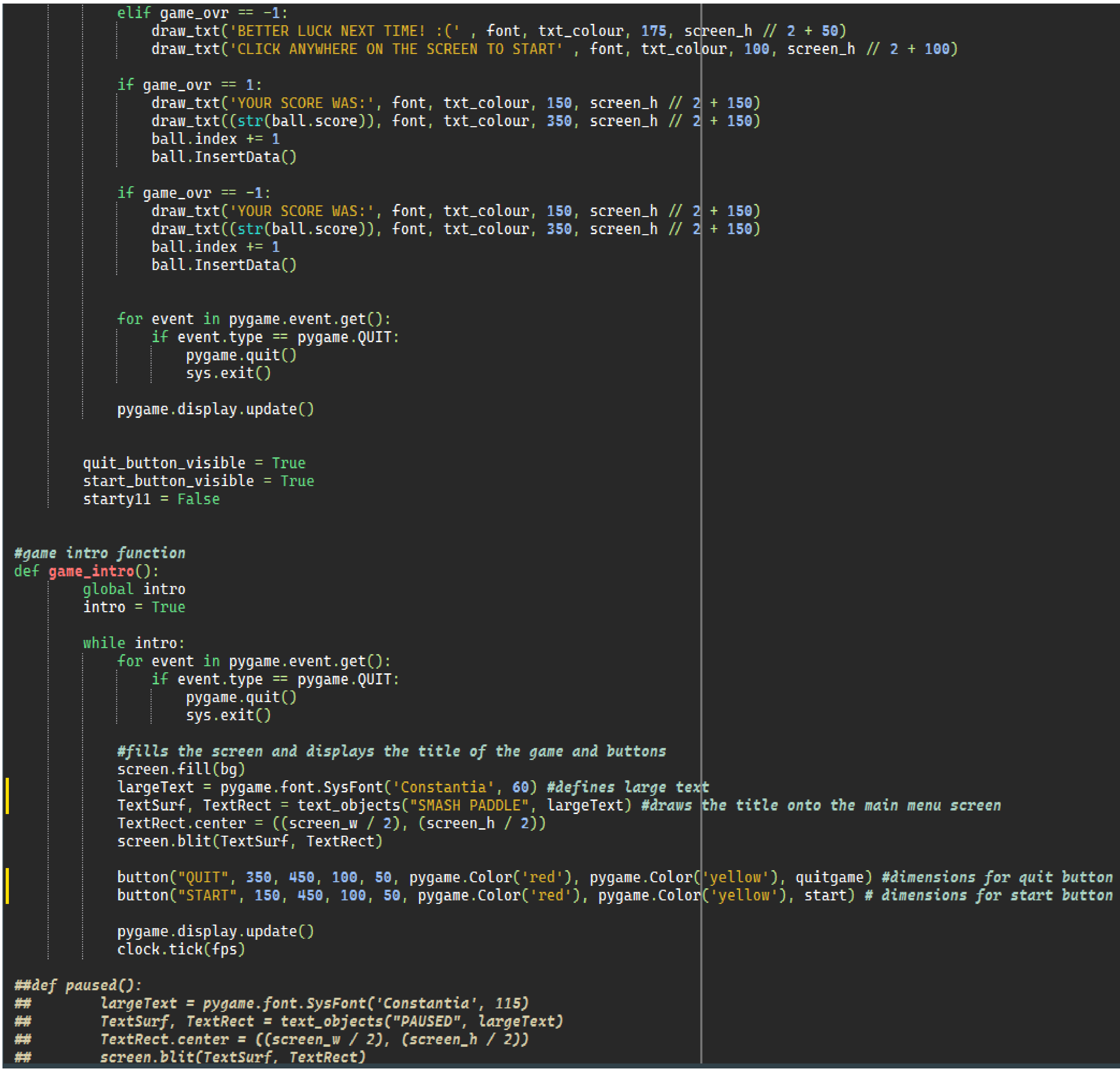
```
                self.velocity_y *= -1
                self.velocity_x += player_pad.direction
                if self.velocity_x > self.velocity_m:
                    self.velocity_x = self.velocity_m
                elif self.velocity_x < 0 and self.velocity_x < -self.velocity_m:
                    self.velocity_x = -self.velocity_m
            else:
                self.velocity_x *= -1 #reverses the ball's speed

        self.rect.x += self.velocity_x
        self.rect.y += self.velocity_y

        return self.game_ovr

    def draw(self):
        pygame.draw.circle(screen, paddle_colour, (self.rect.x + self.ball_radius, self.rect.y + self.ball_radius), self.ball_radius)
        pygame.draw.circle(screen, paddle_outl, (self.rect.x + self.ball_radius, self.rect.y + self.ball_radius), self.ball_radius, 3)

    def reset(self, x, y):
        self.ball_radius = 10
        self.x = x - self.ball_radius
        self.y = y
        self.rect = Rect(self.x, self.y, self.ball_radius * 2, self.ball_radius *2)
        self.velocity_x = 4
        self.velocity_y = -4
        self.velocity_m = 5
        self.game_ovr = 0



    def sql_connection():
        try:
            con = sqlite3.connect('mydatabase.db') #tries connecting to database called 'mydatabase.db'
            return con
        except Error:
            print(Error)

    def sql_table():
        try:
            con = sqlite3.connect('mydatabase.db') #tries connecting to database called 'mydatabase.db'
            cursorObj = con.cursor()
            #if database doesnt exist, it creates a new one
            cursorObj.execute("CREATE TABLE Scores(id int, name str PRIMARY KEY, score int)")
            con.commit()
        except:
            print("Table already exists")
            #if table already exists, it will notify the player

    def InsertData(self):
        global con
        con = sqlite3.connect('mydatabase.db')
        name = str(self.name)
        score = str(self.score)
        index = self.index
        cursor = con.cursor()
        try:
            count = cursor.execute("INSERT or REPLACE INTO Scores VALUES (?, ?, ?)", (index, name, score))
            con.commit()
            #tries to enter values into the database like the player's name and score
```



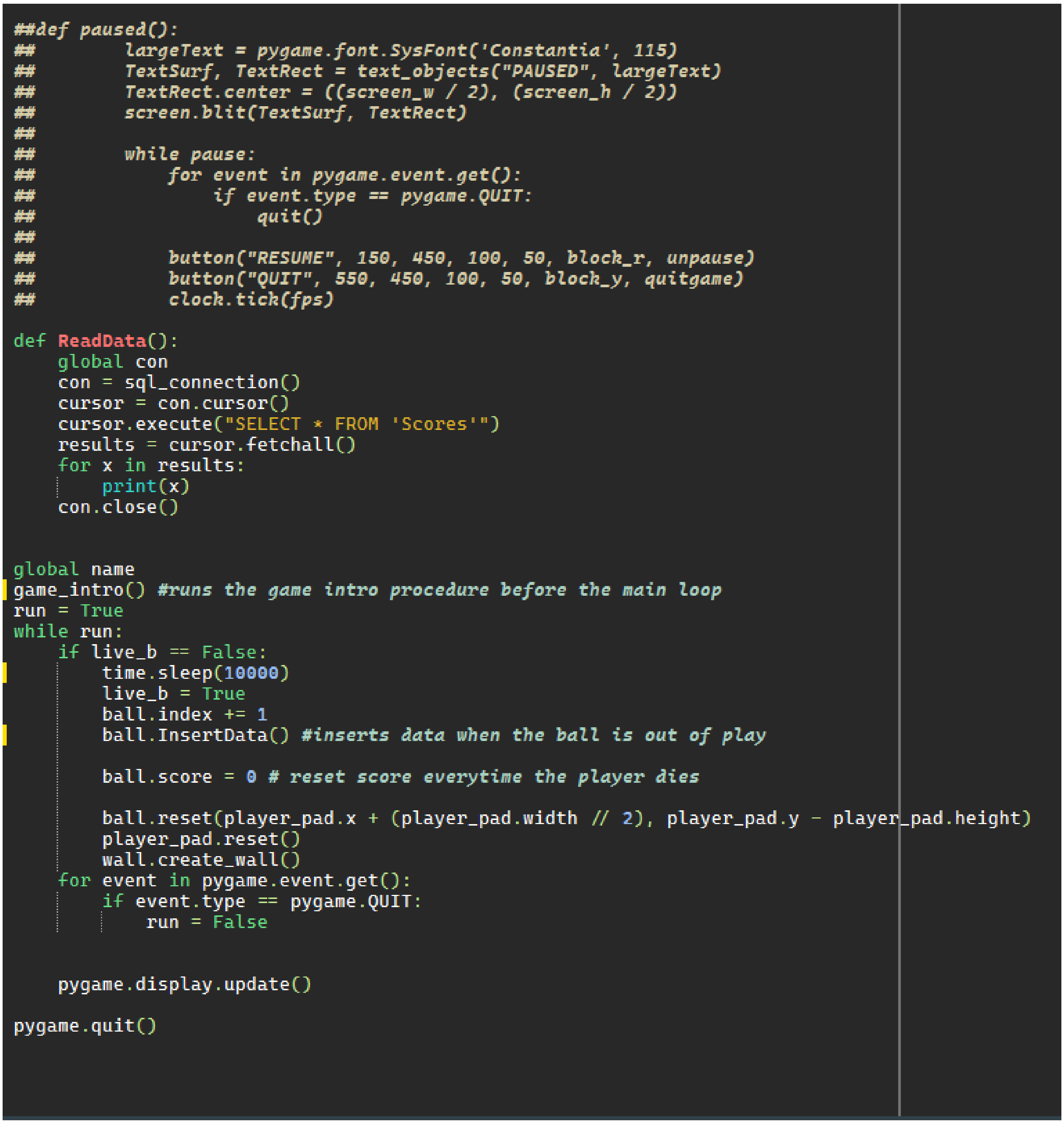
```
            con.commit()
            #tries to enter values into the database like the player's name and score
            print("Record inserted successfully into Scores table", cursor.rowcount)
            cursor.close()
        except sqlite3.Error as error:
            #if it does not go through, it prints an error
            print("Failed to insert data into sqlite table", error)
        finally:
            if con:
                con.close()
                #closes the connection with the database
                print("The SQLite connection is closed")

#creates a wall
wall = wall()
wall.create_wall()

#creates the paddle
player_pad = paddle()

#creating the ball
ball = ball(player_pad.x + (player_pad.width // 2), player_pad.y - player_pad.height)

##def unpause():
##          global pause
##          pause = False

def text_objects(text, font):
    textSurface = font.render(text, True, block_o)
    return textSurface, textSurface.get_rect()

#I have created these variables so that the buttons can disappear when the game is started
global start_button_visible
start_button_visible = True
global quit_button_visible
quit_button_visible = True
global starty11
starty11 = False

def button(msg, x, y, w, h, ic, ac, action = None):
    global starty11
    global start_button_visible
    global quit_button_visible



    if (starty11 == True):
        start()
        pygame.display.update()

    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()
    ic = pygame.Color('red')
    ac = pygame.Color('yellow')
    if x + w > mouse[0] > x and y + h > mouse[1] > y:
        pygame.draw.rect(screen, ac, (x, y, w, h))

        if click[0] and action is not None:
            if action == start:#when start button is clicked, game starts
```



```
        if click[0] and action is not None:
            if action == start:#when start button is clicked, game starts
                starty11 = True
                start_button_visible = False
                action()
            if action == quitgame:
                quitgame()#when quit button is clicked, game closes

    else:
        pygame.draw.rect(screen, ic, (x, y, w, h))

    smallText = pygame.font.SysFont('Constantia', 20)
    textSurf, textRect = text_objects(msg, smallText)
    textRect.center = (x + (w / 2), y + (h / 2))
    screen.blit(textSurf, textRect)


#quit game function
def quitgame():
        pygame.quit()
        sys.exit()

#start game function
def start():
        global intro
        global quit_button_visible
        global start_button_visible
        global starty11
        global live_b

        intro = False
        quit_button_visible = False
        start_button_visible = False #when the start button is clicked and the game begins,
        live_b = True                           #the buttons will disappear


        while live_b:
            clock.tick(fps)


            screen.fill(bg)


            wall.draw_wall()
            player_pad.draw()
            ball.draw()


            player_pad.move()

            game_ovr = ball.move()

            if game_ovr ≠ 0:
                live_b = False


            if game_ovr == 1:
                draw_txt('CONGRATS, YOU WON!' , font, txt_colour, 240, screen_h // 2 + 50)
                draw_txt('CLICK ANYWHERE ON THE SCREEN TO START' , font, txt_colour, 100, screen_h // 2 + 100)
```



```
            elif game_ovr == -1:
                draw_txt('BETTER LUCK NEXT TIME! :(' , font, txt_colour, 175, screen_h // 2 + 50)
                draw_txt('CLICK ANYWHERE ON THE SCREEN TO START' , font, txt_colour, 100, screen_h // 2 + 100)

            if game_ovr == 1:
                draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
                draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
                ball.index += 1
                ball.InsertData()

            if game_ovr == -1:
                draw_txt('YOUR SCORE WAS:', font, txt_colour, 150, screen_h // 2 + 150)
                draw_txt((str(ball.score)), font, txt_colour, 350, screen_h // 2 + 150)
                ball.index += 1
                ball.InsertData()


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

            pygame.display.update()


        quit_button_visible = True
        start_button_visible = True
        starty11 = False


#game intro function
def game_intro():
        global intro
        intro = True

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

            #fills the screen and displays the title of the game and buttons
            screen.fill(bg)
            largeText = pygame.font.SysFont('Constantia', 60) #defines large text
            TextSurf, TextRect = text_objects("SMASH PADDLE", largeText) #draws the title onto the main menu screen
            TextRect.center = ((screen_w / 2), (screen_h / 2))
            screen.blit(TextSurf, TextRect)

            button("QUIT", 350, 450, 100, 50, pygame.Color('red'), pygame.Color('yellow'), quitgame) #dimensions for quit button
            button("START", 150, 450, 100, 50, pygame.Color('red'), pygame.Color('yellow'), start) # dimensions for start button

            pygame.display.update()
            clock.tick(fps)

##def paused():
##        largeText = pygame.font.SysFont('Constantia', 115)
##        TextSurf, TextRect = text_objects("PAUSED", largeText)
##        TextRect.center = ((screen_w / 2), (screen_h / 2))
##        screen.blit(TextSurf, TextRect)
```



```
##def paused():
##        largeText = pygame.font.SysFont('Constantia', 115)
##        TextSurf, TextRect = text_objects("PAUSED", largeText)
##        TextRect.center = ((screen_w / 2), (screen_h / 2))
##        screen.blit(TextSurf, TextRect)
##
##        while pause:
##            for event in pygame.event.get():
##                if event.type == pygame.QUIT:
##                    quit()
##
##            button("RESUME", 150, 450, 100, 50, block_r, unpause)
##            button("QUIT", 550, 450, 100, 50, block_y, quitgame)
##            clock.tick(fps)

def ReadData():
    global con
    con = sql_connection()
    cursor = con.cursor()
    cursor.execute("SELECT * FROM 'Scores'")
    results = cursor.fetchall()
    for x in results:
        print(x)
    con.close()


global name
game_intro() #runs the game intro procedure before the main loop
run = True
while run:
    if live_b == False:
        time.sleep(10000)
        live_b = True
        ball.index += 1
        ball.InsertData() #inserts data when the ball is out of play

        ball.score = 0 # reset score everytime the player dies

        ball.reset(player_pad.x + (player_pad.width // 2), player_pad.y - player_pad.height)
        player_pad.reset()
        wall.create_wall()
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False


    pygame.display.update()

pygame.quit()
```

Bibliography

- Tutorial for base game mechanics: https://www.youtube.com/watch?v=NIfkaOF3Hjs&list=PLjcN1EyupaQlbAyHLsuFIp0n6i_p8XWaO
- Help with creating/fixing new code: https://stackoverflow.com/