# Computer science project

**H446-03**

**Atif Abdur-Rahman**

**Snake With Friends**

## Contents Page

### Table of contents

[Computer science project 1](#_Toc4138640)

[Contents Page 2](#_Toc4138641)

[Analysis 3](#_Toc4138642)

[The problem and stakeholders 3](#_Toc4138643)

[How the problem can be solved using computational methods 4](#_Toc4138644)

[Research 6](#_Toc4138645)

[Features of proposed solution 8](#_Toc4138646)

[Hardware and software requirements 11](#_Toc4138647)

[Success criteria 12](#_Toc4138648)

[Design 15](#_Toc4138649)

[Systems diagram 15](#_Toc4138650)

[Decomposition of problem 16](#_Toc4138651)

[Key variables and structures 19](#_Toc4138652)

[Flowchart 22](#_Toc4138653)

[Pseudocode for flowcharts 27](#_Toc4138654)

[Usability features 32](#_Toc4138655)

[Test plan during iterative development 34](#_Toc4138656)

[Test plan for alpha testing 37](#_Toc4138657)

[Development of coded solution 41](#_Toc4138658)

[Iterative development 41](#_Toc4138659)

[Annotated code 111](#_Toc4138660)

[Evaluation 143](#_Toc4138661)

[Stake holder alpha testing 143](#_Toc4138662)

[Comparing with success criteria 163](#_Toc4138663)

[Evaluation on usability features 165](#_Toc4138664)

## Analysis

### The problem and stakeholders

My game is a variation of the popular game snake that was popularised by the Nokia phone series from. The game follows a snake that is made from blocks with the first block being the head followed by the body that follows the exact path the snake goes on. The objective of the game is to see how big the snake can get without eating itself. To get bigger the snake must eat food usually in the form of apples that randomly appear around the screen. The food is represented by one block and when it is eaten the snake adds another block to its body. The snake eats everything in its path so when the head of the snake touches any part of its body it will eat itself and the player will get a game over. There are two main variation of snake: when the snake dies when it touches the edge of the playable area and one where the snake wraps round to the other side of the area when it reaches the edge.

I plan to improve the original snake by adding a multi-player mode where players compete with one another to try and beat the players and see who can survive for the longest. I also plan to add a time trial mode that would be focused on improving you score on snake in the fastest possible time. During the time trial there would be an option to view a previous trial as a benchmark to see how fast the player can be. These improvement to snake would bring more people in to play it with the nostalgia of snake and the new game mode.

The game appeals to all ages and at the time of release on the Nokia phones everyone was playing it from children to adults. Due to the simplicity of the game everyone was able to learn it quickly including seniors and toddlers. With the new version of snake that I am creating everyone that owns a computer will be able to play classic snake. Before the only people that could play snake were the people that owned the Nokia phone. More people will be able to play snake and people will be engaged on it for much longer with the added features.

The stakeholders for my game are young people who are new to snake and want to get much better and an older audience that want to feel the nostalgia again. In addition to these main stakeholders many students who would like a game to play in their free time would be very interested in this game. The people who I will be focusing on are the people that have played snake and now want to play another version of it together instead of by themselves.

My first stakeholder is Venkata Murali who is an avid gamer and has played the original snake game on the Nokia. He is disappointed that there is no way to play snake without having to be connected to the internet or buying an old Nokia. He wants to be able to play and have a mode where he can practice and have a high score leader board to keep track of his scores and to see how much he improves.

My second stakeholder is Abdiaziz Ahmed who has never played the original snake but still wants to learn it. He also wants to be able to play the game with his friends, so he can have more fun while playing the game for the first few times. With the addition of the time trial mode it will become much easier to see how much better he has gotten when he started and how good his friends are compared to him. I am helping him by creating the leader board system and the time trial system to help him improve and creating a fresh multiplayer version of snake which he can enjoy.

### How the problem can be solved using computational methods

#### ABSTRACTION AND VISUALISATION

* To help refine the problem and simplify it so that it will be easier to code I will need to use abstraction and visualisation. Using abstraction, I can need to remove everything that is not needed and keep everything that important to the problem. In games abstraction is used to get the core concept of the game removing all features that are not needed in the development of the game. Anything that isn’t important to the game such as the way a character moves in real life can be simplified so that the mechanics of the game is easier to code and control.
* In the snake game I am creating the snake’s movement will be abstracted from real life so that the each of the snakes’ body will move one unit at a time. The movement for the snake in my game will use the wasd movement set (w for up, a for left, s for down and d for right). In real life the snake will uses its surrounds and its muscles to move but in my game the snake will just move in that ever direction they are going or the direction that it the user commands it to move. The body of the snake will be separated into one unit each. At the start the whole snake will only be 3 units long. Every time the snake eats it gains one unit. The head of the snake is the only thing being controlled by the user. The body moves forward in the direction that the part in front is moving. This simplified movement will be much easier to code and much easier for the player to manage.
* The graphics of the game will be extremely simple. The snake just be a long rectangle until it changes the direction where the snakes’ body will appear to be bent. The area itself will be a plain black background with a white border representing the playable area. The snake itself will be bright green so that’s its body will stand out and the food will be red blocks. In the multiplayer modes there will be power ups that will be purple. The other snakes in multiplayer mode will be green, blue, red and yellow. Each snake will need to have a white outline to stop its body appearing to merge with itself.
* When the snake eats the food, the snake will not get longer at first but after the last body until moves another unit will appear where the last unit was. This gives the illusion of the snake growing is just the extra unit that is left behind when it eats the food.

#### THINKING AHEAD

* Thinking ahead is where you think about all the inputs and outputs that the game will and give and thinking about all the problems that the game will face. I will need to know all the game breaking possibilities and how I will solve these problems.
* The area in which the snake can move is limited and the whole screen is the limit to where the snake can move. Unlike real life where a snake can go wherever it wants here the snake will only be allowed to stay in the area. If the user tries to leave the area, then the snake will die, and the user will be given a game over.

#### THINKING PROCEDURALLY

* Thinking procedurally is where you take the problem and you break it down into smaller parts. These smaller parts are then broken down again until each part is manageable and can be solved by themselves.
* I will need to add subroutines into my game, so I can reuse parts of the code that will be repeated several times which will also make my code more efficient.
* I will need to create the code needed for the menu selection of each mode and when they view the high score board.
* The game states I need to have is: when the snake changes direction, when the snake eats the food, when the snake hits the wall and when the snake hits itself of another player.
* If the player is put into a tight square by rotating very fast the snake will not die as I will make the code so that the snakes tail move first before the head moves.
* When I am creating the multiplayer mode every instance, I will need to check the inputs so that each snake can move accordingly.

#### THINKING LOGICALLY

* Thinking logically is where you decide the points that need branching (if statements), iteration (for and while loops) and recursion (when a subroutine is called within itself to complete a small task until the whole task is completed).
* The main point of branching within my program is the selection of the modes. I will use an if statement will launch a subroutine used for each different mode.
* The main point of iteration is the body parts of the snake moving in the same direction as each part would need to move where the other part just was.
* The main point of recursion would be when the snake eats the food.

#### THINKING CONCURRENTLY

* This is where different parts of the code run side by side to help be more efficient by seeing what parts of the code can be done at the same time.
* In multiplayer snake each snake is being controlled separately with inputs coming in at the same time. For multiplayer snake to work I must be thinking concurrently and have each snake move simultaneously.
* In game the at every event I need to be moving the snake in the correct direction and taking inputs for the direction the snake is facing.
* I need to be able change the direction for multiple snakes at the same time and by thinking concurrently I can take multiple inputs and change data so that every snake movement in in sync.

### Research

#### Image result for snake nokia 6110SNAKE ON THE NOKIA

Snake was first coded in 1997 by Taneli Armanto and was put into the Nokia 6110. The graphics were extremely basic with black squares and a green background. The only mode was the classic single player mode and the aim of the game was to get as big as possible. In this very old version of snake there was no way to know how long you snake got and what score you got. The original version of snake also had a border to, so the snake would die it its head hit the edge. The controls of the game were very simple, and the head of the snake was controlled by the up, down, left, right buttons on the phone.

#### MODERN SNAKE GAMES

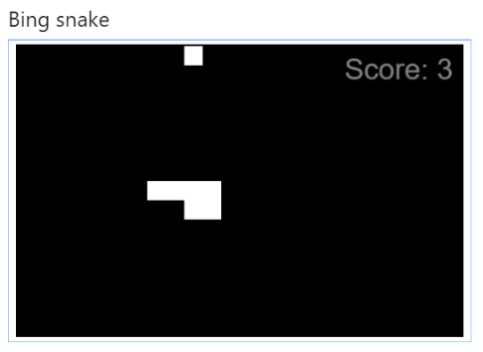
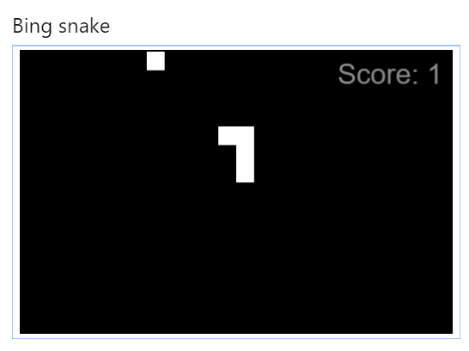
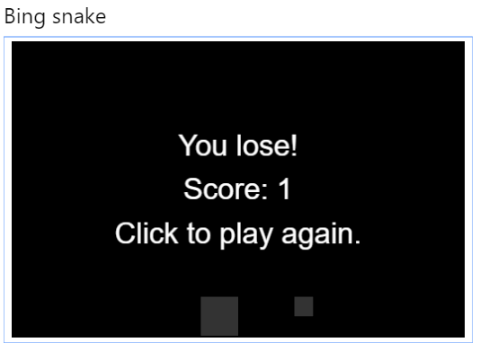
Today, there are still extremely simplified versions of snake that thousands of people use. Bing snake is an example of a modern snake game that is easy to understand. The graphics are so basic that if one part of the snake touches the other part there is no outline to separate them and they look as if they have merged as shown in the picture below. This version of snake does have the score of the player in the top right corner, so the user knows how well they have done compared to other times. Bing snake is inbuilt to the Bing search engine and just searching for “Bing snake” will give you the game to play.

PROS

* The graphics are very simplified and easy to see, the white snake and food stand out heavily from the black background.
* The player can keep track of their score.
* The game is extremely accessible as just searching “Bing snake” on Bing will bring up the game.
* The game can be paused if the player is going to leave and comeback.

CONS

* There is no outline to the snake so when the snake touches its own body without killing itself it’s hard to keep track of the head as the body merges together.
* Sometimes if the food spawns on the top right corner it can block the score and the snake can block the score as well.
* There is no set border to the game so when you first start you don’t know if you can go through the edges of the area.
* The head moves forward before the tail so when the snake is four units long and it forms a square it can eat itself even though it should not be possible.
* You need to have internet connection to play this game because it is inbuilt in the search engine.
* The only mode is the single player mode and the game doesn’t store the high score of the player.



#### MULTIPLAYER SNAKE

There are very few multiplayer snake games and I took me a while to find one. The multiplayer snake game that I found also had very simple graphics which was extremely close Bing snake. The snake itself was green and the food was white with a black background. The mechanics of the game were very similar to Bing snake but there were some major differences. The snake grows by several units when it eats the food. When the snake first spawns it appears to be a single unit but when a directional key is pressed the snake becomes long even though no food has been collected. Another major difference is the speed of the game. The time taken in Bing snake is much slower to move one unit compared to multiplayer snake meaning it is significantly harder to get the right timing to turn before you hit the edge. In this version the snake also dies when its head hits the edge. The last major difference is that when the snake is going one direction and the key opposite to the direction is pressed the snake dies (the snake is going right and the left key is pressed). As for the multiplayer aspect of the game there are 4 modes in which the user can select: 1 player, 2 player, 3 player and 4 players In multiplayer modes if the snake runs into another snake then the first snake will die. The controls for the four player snake is wasd, ijkl, up down left right and 8456 on the numpad. The multiplayer version of snake suffers from many of the problems that Bing snake had.

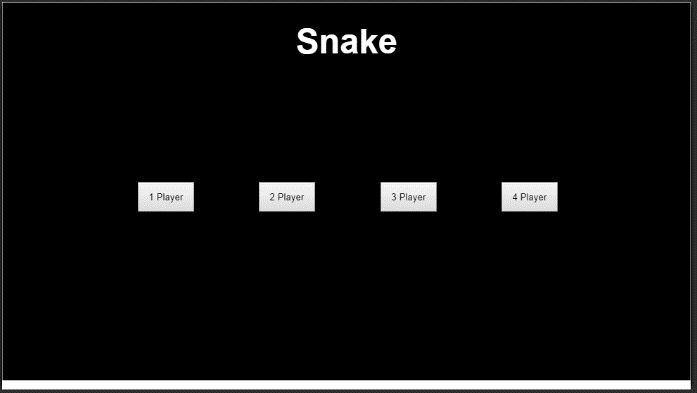
PROS

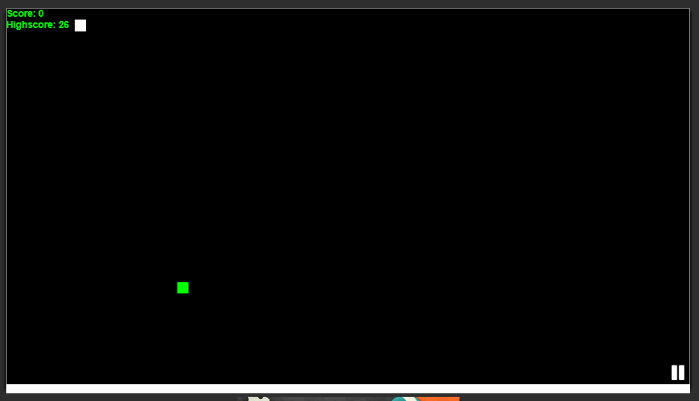
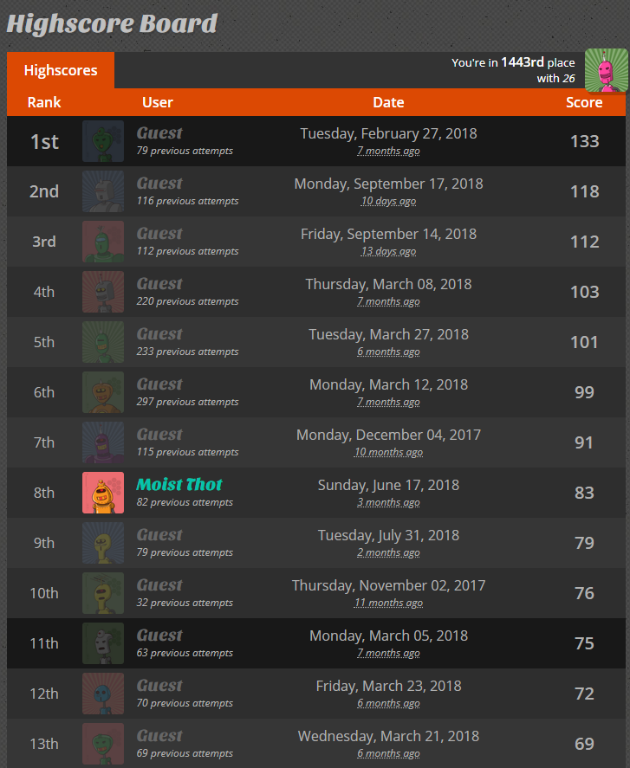
* This version of snake is multiplayer, so it can be played with friends on the pc.
* The user can pick how many players there are with up to 4 players including 1 player, so the user can play the game themselves.
* In 1 player mode the top left of the screen shows score of the players current round and the high score that has ever been achieved.
* Underneath the actual game there is a high score board for every player that has achieved a score in 1 player mode, so people can rank themselves against other people and try to beat them.
* The game can be paused if the player is going to leave and comeback.
* The playable area is much bigger then Bing snake which would allow a much larger snake to be made.
* In multiplayer mode the game keep track of how many rounds each snake has one.

CONS

* There is no outline to the snake so when the snake touches its own body without killing itself it’s hard to keep track of the head as the body merges together.
* Sometimes if the food spawns on the top right corner it can block the score and the snake can block the score as well.
* There is no set border to the game so when you first start you don’t know if you can go through the edges of the area.
* The maximum number of players is four.
* The high score board takes long to update (it said my place was 1443 with a score of 26 even though the scores at places around 1443 were 2 or 3).
* When coming into multiplayer mode it took me a long time to find the control for each player as the game didn’t tell me what the controls were.
* Due to the fast pace of the game and “go backwards and die” mechanic it is extremely hard to do 180 degree turns as most the times the snake won’t turn fast enough, and the snake will end up dying.
* There is a major bug in the game where in multiplayer modes if the last snakes run into each other and both of their heads touch then they will both die simultaneously. Because the round doesn’t end until there is a winner the game never ends and the only way to start again is to refresh the page. There needs to be a way to tie the game.
* Once you select a mode there is no way to get out of that mode unless you refresh the page.
* You need internet connection to play the game as it is on a website.
* The text on the resume button is so small it cannot be read.

#### WHAT I WANT IN MY GAME FROM MY RESEARCH

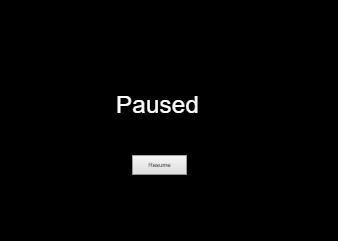
From Bing snake I would like the simplistic style of the white snake against the black background. Also, the food is a simple square and the snake is a rectangle which makes the snake easy to see and understand how to control it. The game also has a pause feature that I would like to implement as it lets the user take a break from the game. From the multiplayer snake game, I would like the user to be able to select either 1 ,2, 3 or 4 players. I liked the fact that the game shows how many round each player has won and how long the snake is now. I also liked the fact that the playing area of the game is much bigger, it allows for longer game and more moves. Also, I also I found that the controls for multiplayer snake were very good because they allowed for lots of hand space.



+



<- the snakes collide head on which causes the game to never end so you would need to restart the game.



### Features of proposed solution

#### FEATURES

|  |  |
| --- | --- |
| **GAME FEATURE** | **JUSTIFYING IF IT SHOULD BE IN MY GAME** |
| Graphics | The graphics of multiplayer snake is extremely basic which I am also aiming for. The colours used standout from the background and the score is shown clearly. I do not like the score being on the same area that the players are also on as it can be blocked by the snakes and the food. I also think I should improve the graphics of the snake should have an outline and not have the whole colour for the entire snake because then parts of the snakes’ body touches other part they appear to merge. I think that the large area for the snake to be in should also be in my game as it increases how big the snake can grow and the maximum high score. |
| Pausing | I do think that pausing should be allowed in my game. Some people think that pausing shouldn’t be in the game as people would abuse it so that they can try to plan so that they get a better score. I will still add pausing to my game as most of the time people need to use it to take breaks, so they can do things in the real world instead of having to constantly play until they die. In multiplayer mode pausing is needed a lot so that people who have died can move away and they can take group breaks without the game constantly running. |
| High score board | I will have a high score board because it is needed to set benchmarks and for people to see how much they are improving. I can store the high score in the game files and an option to view the high scores in game so that they will instantly update every time a score is set. |
| On the internet | I have decided not to put my game on the internet as I feel that sometime internet connection will not always be available for example traveling with a laptop. If I make it into a downloadable game, then people won’t need internet connection to play it. |
| Pace | The pace of multiplayer snake is very fast due to the increased movement speed. This makes it much harder than it should be, so I will stick to the speed of Bing snake. |
| Rate of growth | In multiplayer snake the snake becomes several units longer whenever it eats the food. This means that the maximum high score will be lowered as less food is needed to grow. I will only increase the length of the snake by one every time the snake eats food so that the maximum score is higher so people who get extremely good at the good will have more to time to play and a higher score to beat. |
| Controls | The controls (the buttons you need to press to play) of multiplayer snake are very good as it allows each players hand to have as much space as possible. However, there is no way of checking what each players controls are so I will add a way for users to see what the controls for each player are. I will also add an option for the users to change what each players controls are by binding them to different keys on the keyboard. |
| Methods of losing | In multiplayer snake one method of losing is for you snake to press the opposite directional button to what direction the snake is going on the screen. I think that this shouldn’t be in my game as it can punish players very easily for accidently pressing the wrong button. The game I am making should be fun with a challenge but having one mis press being the end of the game could be extremely irritating causing some players to hate the game because of it. |
| Account | When the game starts, I will need to ask the user to log in to an account. If they don’t have an account, then there should be a button to make one. I need this in my game so that a player cannot imitate another player. In the high score board, the owner of the account must be the one with the score and it shouldn’t have been done by another player. Each player should be making high scores themselves. |

When the game first starts the user will have an option to: log into and account or create an account. Once they have logged in, they can: play classic single player mode, play multiplayer mode, high score board and options.

Classic single player mode: this will be simple snake and due to my research of the original games and the games that I found I will not allow the user to teleport to the other side as most snake games do not allow it, so the snake will die if its head hits the wall. I have also decided that the best background colour is black as all the light colours standout. I will also have a dedicated line underneath the playable area with the players score and the high score of that session so that the player cannot block the score.

Multiplayer mode: this will have 2-4 different players that are all trying to become as big as possible and survive the longest. Before the user starts the play, they must choose how many players they are playing with. The round finishes when all the players lose and the last player to lose is the winner. If both the last multiple snakes die simultaneously then there will be a tie between the last snakes that die. In the multiplayer snake game that I played in my research each snake spawned as a single dot that grew when first moved. For my game I will have the snakes each spawn across the area from each other as their full beginning lengths. There will also be a separate area underneath the playable area that shows the scores of each player and how many rounds each player has won.

Practice mode: instead of having a time trial mode I decided to make it a practice mode as the players would try and get as big as possible not try to get bigger as quick as possible. I will also add the time they played for before they lost as well just to let the user know. There will be two options for the user to pick: with high score ghost, without high score ghost. With the high score ghost the player who got the previous high score will be on the side of the screen in a separate area repeating the exact moves that they did.

High score board: the high score board will show the score of the top twenty players and the score they got and the time they did it in. I will need to store the exact moves the players makes and where the food spawns each time to help with practice mode.

In options I will allow the player to change the movement buttons depending on the players preferences. Each of the four players will be able to change their controls in the options.

In every playable mode there will be an option to pause the game at the bottom in the area underneath the playable area.

The exact size of the playable area is still to be decided, it will be based of a grid with each square being one unit, so it is very easy for the user to see how much the snake has grown and the number of units between itself and the wall.

The snake will only be able to grow one unit at a time so that the maximum possible score is large enough to never reach unless it is a perfect run.

The snake itself will one solid colour but the outline will be white, so you can see the shape of the snake without it appearing to be merged together. The white outline will be put into snake so that it doesn’t cover up anymore then exactly one unit wide.

I will have the snake starting off at three units long, but I will have the score as zero as the snake hasn’t got any food yet, but it still needs to have a body

I will create an account system where players log in to and play so that their high scores will be saved to that person. To make these passwords secure I will need to hash them so that they will not be decrypted at all.

I am going to create a database for my high scores and accounts so that I can add any new high scores very easily using relational databases.

Sound effects will be played whenever something happens

#### LIMITATIONS AND SCOPE

Unfortunately, I will not be able to complete everything I want to do due to the constraints in time this is a list of what wouldn’t be feasible in my time frame.

I will have to abstract the graphics so that they are as basic as possible. In snake games such as google snake the snake is represented as a snake looking sprite and the food is an apple. I will have to make the snake just a rectangular block as it would take too long to create realistic graphics for the snake and the food.

I was thinking about having the number of players to be more than four, but I decided against it as it would cause the people to be close together and unable to play properly. The players would be lacking hand space which would be uncomfortable.

I would have liked to add a mode where the player could verse an AI but I would be unable to the time restraint of the project. If I can finish the rest of my code within a reasonable amount of time, then I can try to create an AI to verse.

I wanted to have a mode where you could verse other players online, but I would need a dedicated server for that which I would not be able to set up.

Another idea was to allow the user to customise the colours of each snake and change the background, but this was not needed, and it was not necessary to add as I could spend my time working more on the actual gameplay.

I am using python to code this and it is extremely bad at running games and using the graphics card, so I will have to time the snake’s movement with the FPS of game so that it seems fluid.

I would have liked to have added a game mode where there are powerups available to grab in the form of different coloured food. The ideas for these foods were things such as a gold fruit that would make you invisible for some time by automatically changing directions when the snake is about to die. A blue fruit that would increase the speed at which the snake moved and a bad purple fruit that would cause your snake to become shorter and lose some points.

### Hardware and software requirements

#### HARDWARE REQUIRMENTS

I am coding my game in python so the minimum hardware and software requirements for my game will be the minimum hardware and software requirements for python. The list of python requirements are listed below.

|  |  |
| --- | --- |
| **HARDWARE REQUIRMENTS** | **JUSTIFICATION** |
| Processor: 1.6GHz | The minimum speed the processor needs to have to run the game. |
| RAM: 1GB | I will need to store the game states inside the RAM while running it. |
| HDD: 500MB of available space | The game files will be stored on the HDD and 1GB is required to store it. |
| Keyboard | This is needed to control the snakes. |
| Mouse | This is needed to select different modes and navigate the game. |
| Monitor | This is needed to view the game so that the player can see it and play it. |

#### SOFTWARE REQUIRMENTS

The software requirements are the items the user will have to install to run the game.

|  |  |
| --- | --- |
| **SOFTWARE REQUIREMENTS** | **JUSTIFICATION** |
| Windows XP or later | This will be able to run the pygame engine. |
| Must have python and pygame | I am coding in python so that is needed for the game to run |

### Success criteria

#### ESSENTIAL

Three different options – when you first start the game you will have three different options to choose from. The classic single player mode, the new multiplayer mode and the high score board for my game.

Classic single player mode – the controls must be given to the player, so they know how to play the game. The snake must be obvious and the it should standout against the background. The snake must die if its head hits its body or its head hits the wall. I need to have an area underneath the playable area is where there will be the score of the game currently in progress and the high score which is the highest score that the player has achieved in their entire play session. By default, the player will move using wasd, w for up, a for left, s for down and d for right. In classic single mode the player can click how hard they want the game to be, the smaller the size of one unit the harder it is to reach the max score.

Multiplayer mode – the user should be able to pick how many players there are either 2, 3 or 4. A player will win when it is the last snake alive, the player will keep playing until the last snake dies and then when it does a new round starts. A high score is kept for each snake to show the maximum score each player has gotten and the number of rounds that each player has won. If the last snakes die simultaneously then a tie will be given, and the last snakes will be given a point each. Each player spawn with three units and when any player moves all the snakes move forward automatically. I want to add a multiplayer mode to my snake game as that will make it unique to other snake games and can bring people together playing a game they used to enjoy extremely.

High score board – the high score board should show the top twenty players and the score they got and the time they did it in. the high score board needs to update each time a new top twenty score is set.

The player(s) should be able to pause the game at any time.

The player will have a top down 2d view of the snakes.

There should be sound effects when certain events happen. I would like to add sound effects for events that happen while playing the game. When a button is clicked in the main menu a click noise is made, when the snake dies a game over noise is made, when the food is eaten another noise is made and when a high score is made a congratulations noise is made. These sound effects help immerse the user into the game and feel more compelled to keep playing it.

An account system where different players create an account that only they can access so that they can set their own personal high score.

A database containing the accounts of each player and the high scores they have obtained, I will need to use this database so that I can check that each players password is correct and when making an account there aren’t any duplicate usernames.

After a few games in single player mode you can be asked to make the game harder or easier. If get very good scores three times in a row the game will ask you if you would like to play in hard mode which would be at a faster pace compared to normal mode. If a player gets three low scores in a row, then they would be asked if they would like to play on easy mode which is much slower than normal mode.

#### OPTIONAL

Practice mode – the user will try to get as big as possible in this game mode but there will be a “ghost” of the high score run running on the side. The screen will be split in half with the player on top and the high score ghost at the bottom doing the exact same run as when it got the high score. For this to work I need to store the move of player in single player mode because if they get the high score then I will need to change the ghost, so it is the latest high score.

Options – here I will add the feature to change the key binds so that the player doesn’t have to use the pre-set key binds and can customise it. The key binds can also be changed for multiplayer mode, so all the players have different controls.

Allow the users to customise the colours of the snake and the colour of the background in the settings.

If the player is doing extremely well then, I will make the area that the snake plays in much bigger and make the size of unit smaller so that the player will have more room to grow. If the player is doing quite bad, then I will make the size of unit much bigger, so the player has a higher chance of filling the whole screen. When the size of one unit is changed the size of the playable are doesn’t change but the maximum score does as the snake can be many more units long in the same space. This widens the skill range and it can get to the point where even extremely skilful players are able to get a higher score.

#### UNNECESSARY

Graphics – more realistic snake graphics so that the player looks like a snake instead of a long rectangle that only moves in 90 degree turns. I might also model the snake in 2.5d where the game seems like it has been rendered in 3D, but the camera is at a fixed angle

Creating a new multiplayer custom mode that will have power ups spawn as well as food, so the players can have a newer experience to snake. The player can have another button that will be used to use the power up that they obtained. The powerup they have stored will be displayed in the heads-up display at the bottom of the screen.

More players in multiplayer mode – I could do this, but it could be a waste as it can be extremely difficult having more than four players at the same computer.

Online multiplayer mode – I can try to do this but without having a server running all the time the online multiplayer mode might not work all the time.

Creating a mode where the player can verse a computer so that they don’t need to have anyone with them to play multiplayer mode

## Design

### Systems diagram

### Decomposition of problem

#### EXPLAINATION OF FUNCTIONS, PROCEDURES AND METHODS REQUIRED

**Functions**

* SinglePlayer function

This function is needed to start the single player game mode, when the user clicks the single player button this function should be run. First the controls for player1 for that user should appear and underneath the controls the player will be told to press space to start. The snake will then be made by creating a snake object and the grid list. Using object-oriented programming for the snake allows me to create several snakes without having to re write code. The grid list is used to hold a grid of the entire playable area as well as everything inside it. I will use characters to represent different items currently in the area.

“1” – the snake

“A” – fruit

When starting the round itself the snake will be loaded in the middle left to give the player the most time to react before they die automatically. The snake will be loaded from the grid list and each unit of it will be 20x20 pixels. The entire area is 520x960 pixels so the grid list will be 30 by 45 blocks with each block representing a 20x20 pixels. The fruit will be randomly generated using the grid list and the fruit can be anywhere that isn’t taken up by the snake. The current high score, player length and back button is also loaded at the bottom of the screen. In the SinglePlayer function other functions will be run in different game states.

* Movement function is run every 160 milliseconds to move the snake in a certain direction.
* Grow function is run whenever the snakes head collides with the fruit.
* Dead function is run whenever the snakes head collides with its body or the wall.
* NewHighScore function is run when the player beats the high score.
* Movement function

This function is needed to be able to move inside the game, every 160 milliseconds the snake will move based on the direction it is facing. Having the snake move every 160 milliseconds allows time for the program to process inputs and sends outputs, this ensures a high frame rate and there won’t be any choppy movement. The function takes the direction that the snake is facing and moves the head block in that direction and moves the rest of the body one block forward. The movement function also makes sure that if the snake is facing one way then it cannot do a 180 degree turn as that would make the snake go back on itself. Every time the snake moves the grid list is updated to make sure that the game keeps track of where the snake is. When updating the grid list a variable called temp is used to move every block of the body one space forward, I will use this variable to temporarily store the space that the block is at so that each part in the grid list can be overwritten without any data lost. When the grid list is updated if there is something already in the spot that the snake is going to move into then the game will check what is in that space. If there is the food in that space, then the grow function will run but if there is a snake in that space or a wall then the dead function will run. Having most of the game inside functions allow for testing of each function independently and greatly speeds up how fast bugs are found.

* Multiplayer function

This is the function that will be ran when the play selects multiplayer mode. When this function is the ran the first thing that will happen is that the user will be asked how many players are playing. When the player chooses the number of players their controls for each of the players appears on screen. There will be lots new players playing in single mode so telling them the controls will be a necessity so that players don’t start without having to press all the buttons to find the controls. Underneath the controls it will say press space to start and when the player does the game will start.

When the game starts the first thing that will happen is snake objects will be made depending on how many players there are. The snake objects will be loaded on screen and they will all begin the move. Every time the players move the grid list update, if the snake runs into another snake then it will die. In this game when the snake dies the game will check to see how many snakes are left. If there are more than one snakes left, then the snake that died will simply be removed and the game will carry on. If there are is only one snake left, then the game will end and the last snake alive will get a point for winning a round. If there are no snakes left due to multiple snakes dying at the same time, then there will be a tie and no snake gains and point and the game ends. When the game ends the result of the game (“PlayerX wins” or “It’s a tie!”) is shown and the game will automatically restart. The only way for the player exits the game mode is if they pause the game and then exit to main menu. The number of rounds that each player has won is displayed at the bottom of the screen. In the grid list there are multiple snakes and they will be represented differently. The grid I am using is going to be a list because using a text file would take up disk space that doesn’t need to be taken up. Also, it is much faster to read from a list then a text file. Having a text file as my grid could also cause the game to become laggy as it takes significant time to save and load data every time the snake moves.

* Player1 – “1”
* Player2 – “2”
* Player3 – “3”
* Player4 – “4”
* Grow function

This function is run whenever a snake’s head collides with the food. When the food is eaten the player, score increases by one and the length of the snake is increased by one. To increase the size on the snake inside the game when the snake moves forward the last block won’t move forward. This gives the illusion that the snake as grown. When the snake eats the food there will be an ascending noise made. This noise gives the player a sense of satisfaction that they have achieved something which will get the player to play more and immerse them in the game.

* Dead function

When a snake dies if the snake is removed from the game and a descending noise is made. The descending noise makes the user feel bad that they died so that they will try to get a better score next time and will play more. If the mode was single player, then there will be a game over and the game will start again on the controls screen. If the player got a high score in that single player run, then the NewHighScore function is run. In multiplayer mode if there is more than one snake then the dead snake is just removed. If there is one or no snakes, then a game over is given.

* NewHighScore function

This function is run when the score in single player mode is higher than the lowest score on the high score board then a linear search is done to find what place the new high score goes into. When the new high score and username is then inserted to the table and the previous bottom high score is removed. When a player creates a new high score, they are congratulated, and a success noise is made. This will make the player try to get a higher score which makes them keep playing.

* LoadHighScores function

In the main menu when the user wants to view the high score board then I will read the table that the high scores have stored them and then show them on the screen. The top twenty scores from the high score board is displayed on screen but all the scores are stored inside the database.

* EditControls function

When the player tries to change the control, the program will ask them what player they want to change them for and control they want to change. Then the program will ask the player to press a key to change it to.

* EditColour function

When the play tries to change the colour, the program will ask them what object they want to change (background, snake1 or snake2 ect). I will give them a drop-down list of the different colours they can pick but when they change a colour to.

* Register function

This function is run when the user clicks the button to register a new account. When the user wants to register a new account then they will need to create a username and enter their password twice. First the username entered is checked against all the existing username and if the username is taken then it cannot be used, and the user will be asked to use a different username. Then the password is checked against the second entered password and if they are the same the password will be hashed, and the username and hashed password is stored in the database.

* LogIn function

When the user enters their username and password the username will be checked against all the stored usernames in the database. If there is a match between the inputted username and a stored username then the entered password will be hashed. The new hashed password will be checked against the stored hashed password and if they are the same the user will gain access.

* ChangeDifficulty function

If the user has played single player mode and gets above a certain score three times in a row, then the user will be asked if they want to play hard mode. If they say yes, then the speed of the snake will increase. If the user below a certain score three times in a row, then the user will be asked if they want to play easy mode. If they say yes, then the speed of the snake will decrease.

* ChangeColour function

When the user tries to change the colour, they user will be asked what they want to change the colour of (background, snake1, snake2, snake3 or snake4) and then what they want to change the colour to.

**Database**

I am going to be using a database for my account system and inside my data I am going to be storing the username and hashed versions of passwords. In the database I will also be storing a table full of the top twenty high scores and the usernames of the people that achieved them. I am going to be using a relational database as the user can have many high scores. When someone registers an account, their password will be hashed and stored in the database. I am going to use SQL to create the database and I will update the database using SQL whenever a new account is register. The accounts table will also store the controls for each player on that account

### Key variables and structures

I will need to make a snake class for multiplayer mode with each object being one of the snakes.

This class is used to create the snakes that are going to be used when single mode or multiplayer mode is picked. The snake objects are created when the player starts the game and the number of players in the game will control how many snake objects are created. Inside the snake class there will be the attributes belonging to the snake and the methods that are used on the snake. The “Create” method is used to draw the snake on the screen. I am going to use object orientated programming because I can create objects that are like each other without having repeated code. Object orientated programming also allows me to make changes to software easier.

|  |
| --- |
| Snake |
| Coordinates  Colour  Length  Rounds won  Control-up  Control-down  Control-left  Control-right |
| Win-round  Die  Move  Create  Gain-food  Bind-up  Bind-down  Bind-left  Bind-right  Kill-snake |

This class is used to create the food that is going to be used in single player mode and multiplayer mode. The food object is created and in every game loop I will run the create method to draw the food. When the food is eaten, I will use the spawn function to pick new coordinates for the food.

|  |
| --- |
| Food |
| x-coordinate  y-coordinate  Colour |
| Create  Spawn |

|  |
| --- |
| newText |
| Text  Size  Font  Colour  Xplace  Yplace |
| mainText  showJustText  addText |

This class is used whenever I want to add some text to the screen. the attributes hold the properties of the text that I am creating such as the position, colour and size. The methods I am creating are used to display the text on the screen. the mainText method is responsible for just the display. The showJustText method put the data in a relevant format that the mainText method will understand and run. The showJustText method will be used when I only want to show text on the screen. the addText method will be used when I want to add text to the screen while other processes are happening.

|  |  |  |
| --- | --- | --- |
| Data structure and name | Function | Data type |
| Grid – list | This is used to store the coordinates of the snake and any other objects on the map. | String |
| Position – list | This is used to store the position of each of the units of the snake. | Integer |
| Direction – variable | This variable is used to store the direction that the snake is facing. This used to keep the snake moving in a certain direction. | String |
| Width – variable | This variable stores the width of the game window. | Integer |
| Height – variable | This variable stores the height of the game window. | Integer |
| Leave – variable | This variable stores a true or false value and depending on the contents the game will continue. If the variable becomes false, then the game will end. | Boolean |
| Timer – variable | This variable controls how often the game loop will repeat. Using this variable, I can control the speed of the snake. | Integer |
| Accounts – database table | This table stores the usernames and hashed passwords of the user. This table also contains the controls for each player. | |  |  | | --- | --- | | AccountID | Integer | | Username | Text | | Password | Text | | Player1Up | Text | | Player1Down | Text | | Player1Left | Text | | Player1Right | Text | | Player2Up | Text | | Player2Down | Text | | Player2Left | Text | | Player2Right | Text | | Player3Up | Text | | Player3Down | Text | | Player3Left | Text | | Player3Right | Text | | Player4Up | Text | | Player4Down | Text | | Player4Left | Text | | Player4Right | Text | | Background  Colour | Text | | Player1Colour | Text | | Player2Colour | Text | | Player3Colour | Text | | Player4Colour | Text | |
| High Scores – database table | This table stores the high score and who owns the high score. It is linked with the accounts table because a user can have multiple high scores. | |  |  | | --- | --- | | HighScoreID | Integer | | High score | Integer | | *AccountID* | Integer | |
| Username – Variable | This is used when logging in and this variable is checked against the database for any matches. I will use this variable throughout the game whenever the database is accessed. | String |
| PasswordAttempt – Variable | This is the variable that stores a hashed version of what the user enters as their password when trying to access their account. | String |
| setPass1 – Variable | This the variable that stores what the user enters when trying to create a password. | String |
| setPass2 – Variable | This the variable that stores what the user enters when trying to create a password. This is used against the first password to make sure they typed the password in correctly. | String |

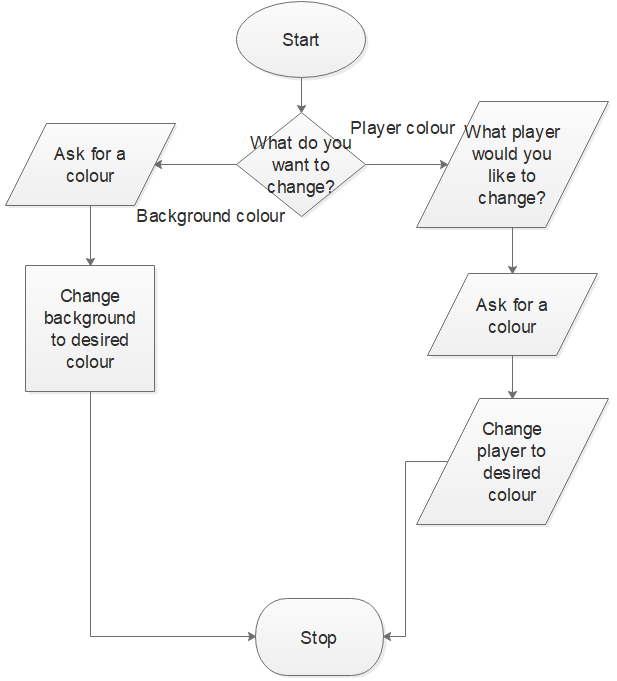
|  |
| --- |
| HighScoreID |
| High score |
| *AccountID* |

**Relationship diagram**

|  |
| --- |
| AccountID |
| Username |
| Password |
| Player controls |
| Player colours |

### Flowchart

#### MAIN MENU



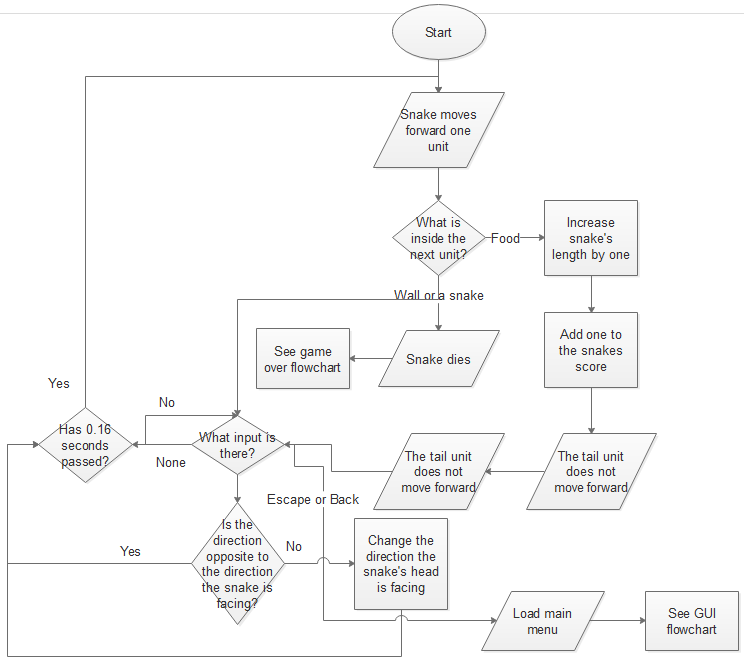
#### CHANGE COLOUR

#### ACCOUNT SYSTEM

#### SINGLE PLAYER START

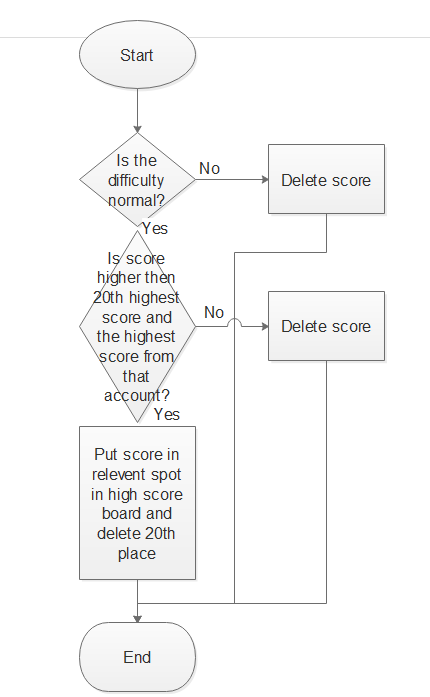
#### MULTIPLAYER START

#### MOVEMENT



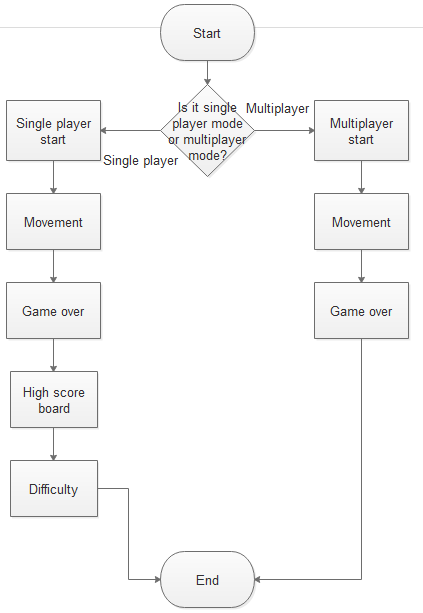
#### GAME OVER

#### DIFFICULTY



#### HIGH SCORE BOARD

#### LINKING FLOWCHART



### Pseudocode for flowcharts

#### MAIN MENU

Draw single player button

Draw multiplayer button

Draw high score board button

Draw options button

Clicked=False

While not clicked then

If multiplayer.clicked = True then

Multiplayer()

elif single\_player.clicked = True then

Single\_player()

elif high\_score\_board.clicked = True then

high\_score\_board()

elif options.clicked = True then

Options()

Endif

Procedure Multiplayer()

numPlayers = input(“how many players are there?”)

for i = 1 to numPlayers

print(player(i).controls)

next i

*see multiplayer*

endprocedure

procedure Single\_player()

print(player1.controls)

*see single player*

endprocedure

procedure high\_score\_board()

scores=20

allScores = SELECT highScores FROM scores WHERE account=user

if len(allScores) < 20 then

scores = len(allScores)

endlif

for i = 1 to scores

print(allScores[i])

next i

endprocedure

procedure options()

choice=input(“Would you like to change keybinds or colour?”)

if choice == “keybind” then

change\_keybind()

elif choice == “colour” then

*see change colour*

endif

endprocedure

procedure change\_keybind()

player=input(“Select a player”)

direction=input(“What direction would you like to change?”)

bind=input(“what button would you like to bind it to?”)

player.direction=bind

INSERT INTO account.player VALUES player.direction

endprocedure

#### CHANGE COLOUR

colour\_change=input(“would you like to change a player colour or background colour?”)

colour=input(“what colour would you like to bind it to?”)

if colour\_change == “player” then

player= input(“which player colour would you like to change”)

INSERT INTO account. colour\_change (player) VALUES colour

Else

INSERT INTO account. colour\_change (“background”)VALUES colour

Endif

#### ACCOUNT SYSTEM

procedure quitGame():

con.commit()

c.close()

con.close()

pygame.quit()

quit()

endprocedure

con=sqlite3.connect("SnakeWithFriends.db")

c=con.cursor()

c.execute("CREATE TABLE IF NOT EXISTS accounts(username TEXT,password TEXT, p1up INTEGAR, p1down INTEGAR, p1left INTEGAR, p1right INTEGAR, p2up INTEGAR, p2down INTEGAR, p2left INTEGAR, p2right INTEGAR, p3up INTEGAR, p3down INTEGAR, p3left INTEGAR, p3right INTEGAR, p4up INTEGAR, p4down INTEGAR, p4left INTEGAR, p4right INTEGAR, p1colour TEXT, p2colour TEXT, p3colour TEXT, p4colour TEXT)")

c.execute("CREATE TABLE IF NOT EXISTS highScores(username TEXT, score INTEGAR)")

displayText(“Snake With Friends”)

choice=input(“Would you like to log in or register?”)

if choice == “log in” then

log\_in()

elif choice == “register” then

register()

endif

procedure log\_in()

account=””

username=input(“What is your username”)

password=input(“What is your password”)

realPassword=SELECT pass FROM accounts WHERE user=username

hashed\_password = hash(“password”)

if realPassword=hashed\_password then

account=username

mainMenu()

else

print(“Your username or your password is incorrect”)

endif

endprocedure

procedure register()

username=input(“Create a username”)

password1=input(“Create a password”)

password2=input(“Re-enter your password”)

if password1==password2 then

INSERT INTO accounts.user, accounts.pass VALUES username, password

Else

Print(“Your passwords do not match”)

endif

endprocedure

#### SINGLE PLAYER

Snake=[[4,6],[3,6],[2,6]]

For i = 1 to len(Snake)

Draw 20x20 square at snake[i,0],snake[I,1]

Next i

Food=[random.int,random.int]

Draw 20x20 square at Food[0],Food[1]

*See movement*

#### MULTIPLAYER START

allPos= [[[0, 23], [0, 24], [0, 25]], [[47, 2], [47, 1], [47, 0]], [[2, 0], [1, 0], [0, 0]], [[45, 25], [46, 25], [47, 25]]]

for i = 1 to len(allPos)

for j = 1 to len(allPos[i])

Draw 20x20 square at snake[i,j,0],snake[i,j,1]

next j

next i

Food=[random.int,random.int]

Draw 20x20 square at Food[0],Food[1]

*See movement*

#### MOVEMENT

If time.elapsed = 0.16 then

If snake[0,0] == food[0] and snake[0,1] == food[1] then

Snake.grow()

Endif

For i = 2 to len(snake)

If snake[0,0] == snake[i,0] and snake[0,1] == snake[i,1] then

gameOver()

endif

next i

if snake[0,0] > 48 or snake[0,0] < 0 or snake[0,1] > 26 or snake[0,1] < 0 them

gameOver()

endif

for i = len(snake) to 1

snake[i][0]=snake[i-1][0]

snake[i][1]= snake[i-1][1]

next i

endprocedure

#### GAME OVER

Procedure gameOver()

Alive=0

Winner=0

If numPlayers == 1 then

Single\_player()

Else

For i = 1 to numPlayers

If allPos[i] == “” then

Alive=alive+1

Winner=i

If alive == 0 then

Print(“it’s a tie”)

Elif alive == 1 then

Print(“player ”+str(winner)+ “ wins”)

Endif

Endif

Endprocedure

#### DIFFICULTY

If len(snake) – 3 < 20 then

Count=count+1

Elif len(snake)-3 > 100 then

Count=count-1

Else

Count=0

Endif

If count = 3

Speed = 200 # This will increase how often the snake will move. The snake will now move every 200 milliseconds making the game easier

Count=0

Elif count = -3

Speed = 120 # This will increase how often the snake will move. The snake will now move every 120 milliseconds making the game harder

Count=0

#### HIGH SCORE BOARD

Place=0

If speed == 160 then

allScores = SELECT scores FROM highScores

for i = 1 to len(allScores)

if snake.length-3 < allScores[i]

place=place+1

endif

next i

if place < 20

INSERT INTO highScores.account, highScores.scores VALUES(account,snake.length-3)

Endif

Display("SELECT \* FROM highScores ORDER BY score DESC LIMIT 20")

endif

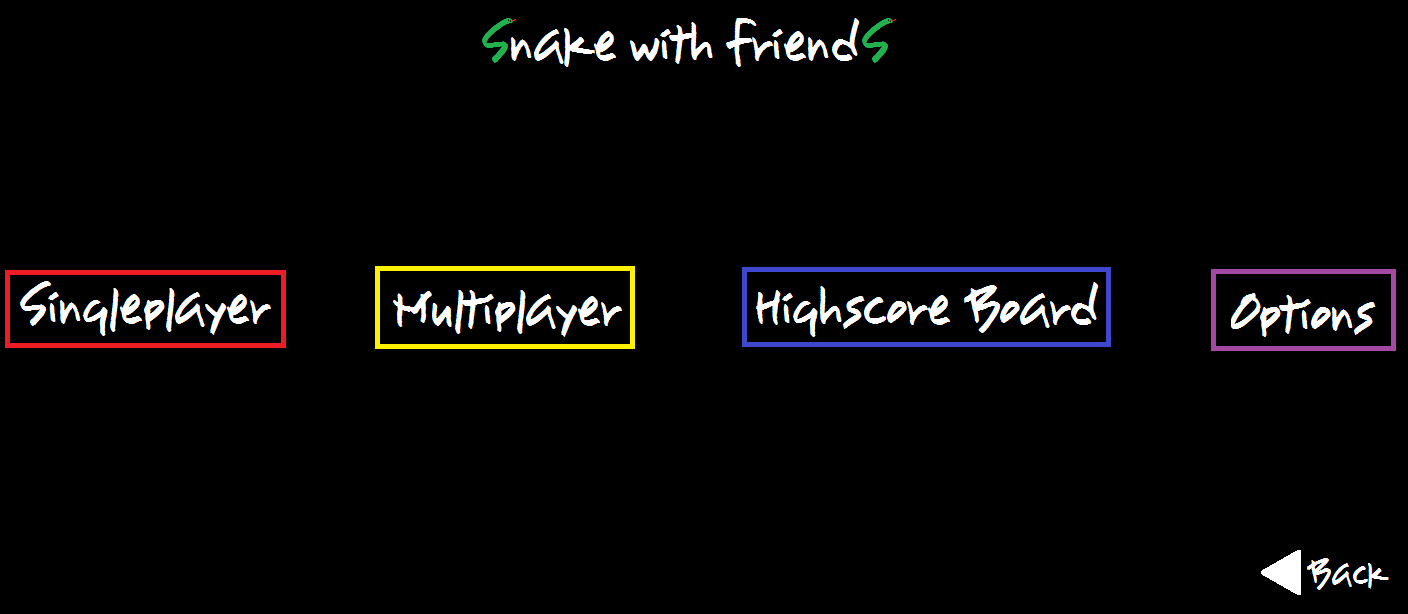
### Usability features

#### USER INTERFACE

Every page of the user interface will have a button to go back to the previous page and a button to quit. In game there will be a pause button. Having buttons for navigating the game is very user friendly and so that people can navigate screens very easily as everything is only a few clicks away.

Login – When the user opens the first thing, they will see is the name of the game with two buttons below (log in and register). The login button leads to a page that has two boxes for the user to type in their username and password. When the user presses enter or presses a tick box then a search will go be made that will check if the username exists and the password is correct. If it is correct, then the main menu will load. If it is not correct, then a message saying “your username or password is incorrect” is given. If the user presses the register button, then a page will load with three boxes for the user to type in a new username and type in their password twice to make sure that it is correct. If the username already exists, then the user will be told that the username already exists. If the passwords don’t match, then the user will be told that the passwords are not the same. If the username and password is valid the username will be teleported onto to the log on screen.

Main menu – after the user has logged in, they have the choice to look at the high score board, change parts of the game in options, play single player mode and play multiplayer mode. There will be a button for each choice and clicking any of the buttons will lead to the desired target.



High score – if the player chooses the high score then a table will be shown with the top twenty high scores.

Options – in the options the player can either change the key binds for the controls, the colour of the background or the colours of the snake. When changing the colours, the player will get a drop-down list of the possible colours. A drop-down list would be best because there are lots of options and it would be better than asking for RGB values. Choosing RGB values wouldn’t be best because not all people understand the RGB values and how they can control them. Having set colours would be better because it’s much easier to understand and there are no invalid inputs.

In game – when the game starts the controls are shown on the screen and “press space to start the game”. This allows the user to be ask fast as they want when starting the game or be as slow as they want. When the game has started at the bottom there will be a pause button and the current score of the player. If it is multiplayer mode, then the number of rounds won will be at the bottom. When the pause button is pressed the player has a choice to resume, quit or go to the main menu.

pause button 🡪

#### **USABILITY FEATURE**S

The input for my game is the keyboard and mouse, keyboard is main input when the game starts and the most intuitive controls for the snake movements would be the arrow keys. The most common controls are wasd for up, left, right and down but arrow keys are simple and can be understood by almost anyone. I have also given the user the ability to change the controls for any player if they want to. This allows users that are more comfortable in different positions to change what controls they have whenever they want.

In my game the player can change the colours of the snakes and the background, this helps colour blindness because they can change the colours to ones that they can differentiate. Green/red colours blindness is the most common and the two of the snakes are green and red, so they can be changed to colours that are easier to differentiate such as blue and orange.

My user interface is very user friendly as it is comprised of simple tasks such as clicking buttons or selecting items from a drop-down list. This would prevent invalid inputs, be clear, be consistent and be efficient.

#### VALIDATION

In my game the inputs are restricted so is not much I need to control, the most amount of validation needed is in the accounts system. In the accounts system I will need to make sure that all new usernames are valid, and all entered username are correct. I will also need to make that people type in the exact same password when logging. If the user is creating an account, I should let the password be the same as another account because if I don’t then it is possible some people’s passwords will be given away. When creating an account, the user should type in the password twice because if they were to type in their password wrong accidently then the user wouldn’t be able to log in again.

Another form of validation is the that the snake cannot go through a set area and can only stay in the “playable area”. I will make sure that the playable area doesn’t overlap with the area that the HUD is. This is to make sure that the user can always see the score and it is impossible for the food to spawn in the same place the score is. In my game if the user tries go outside the playable area then they will lose.

### Test plan during iterative development

|  |  |  |  |
| --- | --- | --- | --- |
| **Test number** | **Test** | **Test type** | **Expected outcome** |
| 1 | See if a snake of a single unit can be made | Normal | A single white block of 20x20 pixels should be made at the left most side of the game |
| 2 | See if I can control the mini snake and move it using the arrow keys | Normal | Every 160 milliseconds the mini snake should move by one block in the direction of the arrow key I pressed last |
| 3 | See if I can create a snake longer than one unit and have the same movement as a proper snake would | Normal | A snake of three units should be made and it should have the movement that the original snake games have. Each unit of the snake should be made and follow the unit in front |
| 4 | Boundaries | Validation | When the snakes first unit collides with the wall the snake should die, and the game will quit |
| 5 | Stop the snake going through itself | Validation | The snake should not be able to go in the opposite direction to which it is facing |
| 6 | Create the food | Normal | The food should spawn in a random place every time the game is run |
| 7 | Create food in a space that isn’t taken up by the snake | Validation | The food should spawn anywhere on the screen that isn’t occupied by the snake |
| 8 | Snake eats food | Normal | When the snake takes over the position of the food the food should show up somewhere else and the snake should become one unit longer |
| 9 | Stop the snake colliding with itself | Normal | If the snake where to collide with its own body, then it will die |
| 10 | In-game HUD | Normal | When you start playing at the bottom of the screen the score of the player and the pause button should appear |
| 11 | Pause button | Normal | A pause button should be created at the bottom of the screen and when it is pressed a screen will show that allows the exit the game, take a break or go back to the main menu. When the user resumes the game the position and the size of the snake will be the same. |
| 12 | Main menu | Normal | When the game first loads the main menu will show and there will be buttons that allow the user to navigate the game |
| 13 | Create game over screen | Normal | When the snake dies in single player mode a game over screen will load asking if the user wants to play again. |
| 14 | Creating a screen that tells the user the controls | Normal | When the user clicks single player mode the program should tell the user the controls and wait for them to start |
| 15 | Pick the number of players for multiplayer | Normal | When the player chooses multiplayer mode, they will choose the number of players they want to play with |
| 16 | Creating the snakes and the movement | Normal | When the chooses the number of players the snake will be made and begin to move |
| 17 | Stopping the snakes from colliding | Validation | When a snake’s head collides with another snake’s body it will die, when a snake’s head collide with another snake head both snakes will die and when a snake’s head collide with the wall it will die |
| 18 | Growing | Normal | Food should spawn and when a snake eats it then the snake should grow |
| 19 | Multiplayer HUD and winning multiplayer mode | Normal | At the bottom the screen the number of points each player has and the number of rounds each player has won and the game should display who won the game |
| 20 | Give user their controls for multiplayer mode | Normal | When the user picks how many players there are the game will show the user the controls for each player |
| 21 | Bug testing | Invalid | At this point in time I decided to test the game so far with a friend |
| 22 | Taking points from other players | Normal | When a snake runs into another snake the surviving snake gets longer by the length of the dead snake |
| 23 | Creating an intro screen | Normal | When the player loads the screen, the game should ask if the user would like to create an account or log into an account |
| 24 | Creating a database | Normal | When the user starts the game if the database doesn’t exist then it is made |
| 25 | Making textboxes | Normal | I need to create a textbox that the user can type in |
| 26 | Hashing | Normal | A hashing function that will hash the user’s password |
| 27 | Create an account | Normal | The user can create a new account with a username and password |
| 28 | Log into an account | Validation | If the user has an account, then they will need to type in their username and password to go to the main menu |
| 29 | Back button | Normal | In the game when the player wants the return to the previous screen, they can press this button |
| 30 | Creating the high score board | Normal | When the player wants the see the high score board, they will see the top twenty scores |
| 31 | Creating an options menu | Normal | In the options menu the user can pick if they want to change colours or controls. The user can pick the player they want change |
| 32 | Changing player colours | Normal | The player selects the player and can change the colour of that snake |
| 33 | Extra code | Validation | Removing extra code |
| 34 | Changing the controls | Normal | The player selects the player and can change the controls for that snake |
| 35 | Speeding up changing the colours | Normal | Making the colour change quicker and less laggy |
| 36 | Sound | Normal | When the player eats food, wins a game or loses a game a sound effect will play |

### Test plan for stake holder testing

#### ACCOUNT SYSTEM

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Input** | **Test type** | **Expected outcome** |
| Allow user to create an account on the log in screen | Click the register button | Normal | A screen that allows you to create an account |
| Be able to create an account | “Hello”- username  “splitter”- password  “splitter”- password | Normal | An account should be made with the name hello and the password splitter |
| Be able to log into an account | “Hello”- username  “splitter”-password | Normal | The main menu should load and all the settings for that account are loaded |
| Check that two accounts with the same name cannot be used | Create another account  “Hello”- username  “object”-password  “object”-password | Erroneous | The username is already taken so the user should be allowed to make that account and they will need to pick a new username |
| When creating a new account, the user should be able to type in the same password twice | “bob1234”- username  “brown”- password  “blue”- password | Erroneous | The passwords are not the same, so the user will need to try again to create an account. |
| When logging into an account all the characters need to be the same to allow entry into the account | “Hello”- username  “Splitter”- password | Erroneous | The user will not gain access to that account the password is incorrect. They used a capital S instead of a lower-case S |
| When logging into an account the username must be correct | “hello”- username  “splitter”- password | Erroneous | The user will not gain access to the account as there is no account with the name hello with a lower-case H |
| Creating an account with a similar name | “hello”- username  “splitter”- password  “splitter”- password | Extreme | This account should be allowed to be made as the username is not exact same as any other usernames |
| Check that all the controls are the same for that account | Change the up control to “5” then quit the game and log into the same account. Then log in and play the game and press 5 | Normal | The control binds are the same for that account. |

#### MENU

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Input** | **Test type** | **Expected outcome** |
| Check that the buttons for the main menu work | Click the button leading to each mode | Normal | Whichever mode I choose should open when clicked |
| Check that there are no hidden buttons | Click anywhere on the screen that isn’t a button | Erroneous | Nothing |
| Check that single player mode button works | Click single player mode | Normal | Single player mode should load |
| Check that the multiplayer mode button works | Click multiplayer | Normal | The number of players is asked by the computer |
| High score board shows | Click high score board | Normal | The high score board should load with the top twenty scores and the names of each of them |
| Options show | Click options button | Normal | The options menu should show |
| Check that the controls are given to the player when they start playing | Click single player mode | Normal | When single player mode loads it should tell me the controls to play the game |
| Multiplayer choice of players | Click single player mode and click on any number of players e.g 4 | Normal | Whichever multiplayer option you pick the game should load that desired number of snakes and show each snake’s controls. Four snakes should load when 4 players is clicked |
| Back button | Clicking the back button | Normal | This should send you to the main menu when pressed |
| Quit button | Click the quit button | Normal | The game should close when the quit button is pressed |

#### OPTIONS

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Input** | **Test type** | **Expected outcome** |
| Binding the keys | Bind d to move up for player two | Erroneous | There should be an error saying that d is already binded for player one. |
| Binding keys | Bind q to move up for player one | Normal | Player one should be able to use q to move up now |
| Binding keys | Bind 9 to move up for player one | Extreme | Player one should be able to move up using 9 now |
| Sound effects | Click the SFX button | Normal | The sound effects of the game should turn off |

#### GAME STATES

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Input** | **Test type** | **Expected outcome** |
| HUD | Start single player mode | Normal | The highest score is loaded, the pause button is loaded, and the current score is loaded. |
| Starting to move in single player mode | Pressing a random keys e.g l and up arrow | Erroneous | Nothing should happen when l and up arrow is pressed |
| Moving in single player mode | s | Normal | The snake start moving down |
| Checking control for movement | Pressing a when facing right | Erroneous | Nothing should happen |
| Checking quick response | Pressing s then a after one movement | Extreme | The snake should move down then left as a sharp 180-degree turn |
| Checking for square bug when snake is four units long | Repeatedly pressing w then a then s then d | Extreme | The snake should become a tight square |
| Changing direction in game | W, a, s, d | Extreme | When facing right only w and s should move the snake at all a and d shouldn’t do anything |
| Starting in multiplayer mode | w | Normal | This should move player one up and every other snake in the direction they were facing |
| Starting in multiplayer mode | r | Erroneous | If no controls are bound to r then nothing happens |
| Eating food | Snake’s head collide with food | Normal | The snake should grow by one unit and the score should go up by one |
| Snake runs into wall | Have the snake keep going forward | Normal | The snake should die when its head hits the wall |
| Snake dying | Have the snake go across the wall and turn when it meets the corner | Extreme | The snake should still survive when hugging the wall and turning in a corner |
| One snake left in multiplayer mode | Let all the snakes apart from one run into a wall | Normal | The final snake wins the game and one point is given to that player. Multiplayer mode then restarts |
| Last snakes alive die simultaneously | Have two snakes run into each other head on | Extreme | The game is a tie and no one gains any points |
| Eliminate another player | Have one snake run into the body of another snake | Normal | The snake that runs into the body dies and the surviving snake gets one point of eliminating an opponent |
| Easy mode | Get less than 5 points more than three times in a row | Normal | There should be a prompt asking you if you would like to play on easy mode where the snake moves slower |
| Hard mode | Get more than 20 points more than three times in a row | Normal | There should be a prompt asking you if you would like to play on hard mode where the snake moves faster |
| Testing the pause function | ESC or the pause button | Normal | A menu should show that asks you to either resume or go to the main menu |
| Quitting snake while in a round | Pause the game then press quit | Normal | The game should close |

#### SOUND TESTING

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Input** | **Test type** | **Expected outcome** |
| Check that the eat sound works | Eat the food in the game | Normal | A collision noise should be heard |
| Check that the death sound | Have the snake run into the wall | Normal | The death noise should be made |
| Check that the high score noise works | Beat the high score | Normal | A celebration noise should be made |

## Development of coded solution

### Iterative development

#### SINGLE PLAYER MODE

##### Creation of snake

In this section the goal is to make a snake class that I will use to create the snake for single player mode. I need to create a method that will draw the snake and a method for the snake to move. The attributes of the class will dictate where the snake is, how big it is and what colour the snake is. I will also need to stop the snake from going outside the screen and I will need to stop the snake from going in the opposite direction that it is facing.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test number** | **Test** | **Expected outcome** | **Actual outcome** |
| 1 | See if a snake of a single unit can be made | A single white block of 20x20 pixels should be made at the left most side of the game | The test worked, and I was able to create a mini snake of one unit. To create a mini snake, I used object orientated programming to create a snake class that contained important attributes of each snake and the instantiation method to create a snake. Inside the instantiation method I drew the snake as a 20x20 square. The attributes were x, y, width, height and colour. This allows me to change colour or size of the snake for other snakes if I need to. |
|  | | | |
| 2 | See if I can control the mini snake and move it using the arrow keys | Every 160 milliseconds the mini snake should move by one block in the direction of the arrow key I pressed last | The test worked, and I was able to control the movement of the snake and change the direction of movement. First, I made an if statement that would save what was inputted last. Then using the input, I created a function called movement which changed the pixels that mini snake occupied while simulated movement. |
|  | | | |
| 3 | See if I can create a snake longer than one unit and have the same movement as a proper snake would | A snake of three units should be made and it should have the movement that the original snake games have. Each unit of the snake should be made and follow the unit in front | The first test did not work. Before I tested it, I added a 2d list called positions which contained the position of each unit of the snake. Then throughout the code I removed the x and y variable and replaced them with the list. Then to draw the snake inside the instantiation method using a for loop I drew the rest of the snake. In the movement function I added a for loop that would update the positions of the rest of the body. The first test did not work because position[i] wouldn’t exist because the length of position doesn’t equal the index of the last element of the list.  The second attempt did not work as intended. To fix the first error I start the counting from the length of the list minus one. When I ran the code, the snake was not attached to fully. I realised that the snake was drawn according to the position itself which of would have been [1,15] and [2,15] so those units would appear in the top left of the screen.  The third attempt worked, to fix the second attempt I simply multiplied the position of the snake by 20 when I drew it. This is because the position in the list is divided by 20 so that I can create a grid list later which is 48 by 26. |
|  | | | |
|  | | | |
|  | | | |
| 4 | Boundaries | When the snakes first unit collides with the wall the snake should die, and the game will quit | Too add the boundary to the game I wrote that the snake would die if the snake goes beyond the screen. It worked however due to the order of the logic between the snake dying and the screen updating the snake would appear to cross the boundary then die.  To solve this all I needed to do was to move the death code after the update code. This worked, and the game would quit when the snake collides with the wall. |
|  | | | |
|  | | | |
| 5 | Stop the snake going through itself | The snake should not be able to go in the opposite direction to which it is facing | When I first coded this, I used the direction variable to check if it wants to move in the opposite direction. This did not work because if the player changed the direction up after facing right and then pressed left before the snake had moved at all then the snake would go in on itself.  To fix this I asked if the snake would collide with the second if it went back. If the snake would collide with itself then the snake cannot turn in that direction. This will not affect the death mechanism that I will add because there is no way for the snakes first unit to collide with the second unit. |
|  | | | |
|  | | | |

###### Review

I was able to create a snake class that was able to move around with the correct movement and I stopped it from exiting the screen. The snake moves one unit at a time, and it can change directions to up, left, right or down. The snake cannot change direction to the opposite direction as it would be going through itself. At the moment the movement for the snake is in a separate function and not part of the snake class. The current game only exists as single player mode and I will need to allow the user to play again when the snake dies instead of the game closing. I used pygame’s key values to check what button the user is pressing so that I can see which direction the snake needs to look in.

##### Creation of food

In this section I will be adding in the food which means I will need to make the snake grow when the food is eaten, and I will need to spawn the food again. The spawning of the food must be random, but It cannot spawn inside the snake. Due to the fact that the snake is now growing the player must not be able go through its own body so I will need to make the snake die if its head collides with its body.

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Create the food | The food should spawn in a random place every time the game is run | First, I created a function called create food which would pick a random x coordinate and a random y coordinate. Then I created a variable called eaten which would I will use to check if the food had been eaten. I then used an if statement to check if the food had been eaten and if I had then I would run the function and change eaten to false. When I ran this, it didn’t work, and I didn’t see the food at all. I thought I didn’t see the food because I ran the function after the display had been updated “pygame.display.flip()”. To solve this on my second attempt I moved the if statement to before the display is updated. This also did not work, and I didn’t see any sign of the food at all. After this attempt I thought that because of the if statement on the first update the food would appear but would then disappear because the eaten variable would become false so there was nothing making the food. On my third attempt I just wanted to test to see what would happen if I took the function out of the if statement. This time I did see the food but because the x and y coordinates change every time the game loop is function is run the position of the food changes place extremely fast and it would seem like the food was teleporting across the screen. For my fourth attempt I realised that I would have to create two functions: one used for generating random coordinates and the other for drawing the food itself. I also realised that I had to run the spawn function outside the game loop to give the initial coordinates of the food. Then inside the game loop I would run the create food function and create an if statement that would change the position of the food when eaten. When I ran my code, it worked, and the food was made and stayed in the same place. To make sure that the random function was working I ran the code again and the food spawned in a different place. |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
| 7 | Create food in a space that isn’t taken up by the snake | The food should spawn anywhere on the screen that isn’t occupied by the snake | On the first attempt wrote that if any of the snake’s body was in the same place as the food then the food would spawn somewhere else. I didn’t want to have to run the game every time to check if the food spawned in the same place as the snake. To make the checking process quicker I took the spawn function outside the if statement and I made it run as fast as possible. I didn’t see the snake overlap the food at all. To test that the new piece of code worked I ran the code with the spawn function inside the if statement. On this new test I controlled the snake and tried to eat the food. When they connected the food disappeared entirely. At first, I thought that this was a problem with the spawn function, so I ran the code again. This time when the snake ate the food the food did spawn again. I realised that this problem was with the spawn function and the random number generator. If the random number generator were to give a 26 for spawny or 48 for spawnx the food wouldn’t be visible. To fix this I changed range to 0-25 and 0-47. |
|  | | | |
|  | | | |
| 8 | Snake eats food | When the snake takes over the position of the food the food should show up somewhere else and the snake should become one unit longer | To make the snake grow when the food is eaten first, I made a function called grow that would increase the length of the position array. Then in the game loop I wrote that when the first unit of the snake is in the same place as the food then the grow function will we run. This did not work when run because the snake did not grow when it ate the food.  I wasn’t sure if the position list was growing or not and I wasn’t sure if the order of the code was correct as the food could be changing position before the grow function is run. I decided to print out the positions list every loop. I found that the position list didn’t grow at all when the snake ate the food.  I realised that the snake was not growing because the positions was a local variable in the grow function, so the list would need to be passed in as a parameter and then returned. Also, I decided to move the code for running the grow function before the code for running the spawn function so that the grow function would run. This worked as seen in the screenshot the snake grows whenever it eats the food. |
|  | | | |
|  | | | |
|  | | | |
| 9 | Stop the snake colliding with itself | If the snake where to collide with its own body, then it will die | On the first time I added the code it worked. I created a for loop that would check each unit of the snake’s body and check if the first unit of the snake has the same x and y coordinate as any of the other body units. I started my “for loop” from two because I didn’t want the first or second unit being check because the game would end because the first unit is in the same place as itself. I didn’t want to check the second unit because if the snake were to try to do a 180 degree turn then it might die when nothing should happen. In this stage I changed the colour of the snake to green and the food to red to more suit the design of my game. To make my games code more structured I started using more object orientated programming and I created a food class and I changed some functions into methods that are part of my snake class. |
|  | | | |

###### Review

I have almost finished single player mode and all I need to do is add the HUD and give the controls to the user to complete it. I added the ability to eat food as a method in the snake class so that the snake will grow. I added movement to the snake class as a method instead of a stand-alone function. I created a food class that creates the food and changes its position every time it is eaten. I also added colour to the snake to the snake and the food to make it stand out more. Since the snake can now grow, I needed to stop it from colliding with itself, when they do collide the snake dies and the game ends. The I colour that I added is a tuple of rgb colours that I can use for anything else that needs to be drawn by pygame.

#### GUI

##### HUD

In this section I am creating a heads up display at the bottom of the screen that allows the user to see their score and the pause button. Also, the snake cannot enter the area the HUD is occupying so when it does enter it then the snake will die. To create the HUD I will need to add text to the game so that the user can read their score.

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | In-game HUD | When you start playing at the bottom of the screen the score of the player and the pause button should appear | To create a HUD (Heads up display) first I needed a way to add text to the screen. To do this I created a text object that I will use to create text on the screen. The attributes that I am using are the attributes that give the text is properties. I am using three methods the first two are used to create text in the centre of the screen. when I need to create large text that is going to be in the middle of the screen, I will call the “showJustText” method. This will call the “mainText” method and together they will find the centre of the screen and put text inside it. To create the HUD, I needed a separate area underneath the snake area that the snake cannot go in. to do this I made the height of the display bigger, but I kept the death borders the same. I made a score object of the text class that would display the current score of the player. The text did show correctly but when the snake ate the food the score did not change. I also found that the score was extremely small and almost unreadable. The HUD did work as when the snake when into it died, and the game ended.  I realised that the reason that the score was not increasing was because even if the player1 length change the score wouldn’t change because it was outside the game loop and it wasn’t being constantly updated. To solve this, I change the text inside the score object so that it would be updated constantly. Also, I made the font size of the score bigger and moved it to the left corner, so I can have the pause button on the other side. To stop the user from running into an invisible wall at the button I made a white rectangle that was show the border that the snake could be in. This worked, and a white line appeared and when the players heads touches it the snake will die. I changed the colour of the line to blue, but it didn’t stand out so much, so I made it white again. |
|  | | | |
|  | | | |

###### Review

For the HUD I added a line that separated the playable area and the area for the HUD. I used the same pixel size of the playable area, but I increased the height of the game window and I added the HUD in the extra space created with the increased height. I added the score at the bottom of the screen, and I used the snake length minus three as the score number. To create text on the screen I made a class called newText that will take text and positions as attributes and the method will draw the text. I can use the newText class as a basis for whenever I need to display text to the screen such as on buttons or a title etc. At the moment the game is still only single player mode and when the snake dies the game ends so I will need to add the ability to play again after the snake dies.

##### Buttons and menu’s

A large part of the user interface is buttons and so creating a button class that can show a clickable button and runs a function is needed. The button class should have attributes that control the position of the button, the colour of the button, the size of the button and the function that the button will run if it is clicked. The methods of the buttons should allow me to draw the button and run a function if the button is clicked. Having buttons is useful for many reasons but I will use it primarily to allow the user to interact with the game and navigate it.

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | Pause button | A pause button should be created at the bottom of the screen and when it is pressed a screen will show that allows the exit the game, take a break or go back to the main menu. When the user resumes the game the position and the size of the snake will be the same. | At first, I created pause text to show what the button would be like and to find the dimensions of the button.  Then I created a button class that inherited its attributed and methods from the newText class. This allowed creating the button and the text in the same place without having to create a separate button object and text object every time I wanted to create a button with text on it. Inside the button class I added attributes that control the size of the button and the colour of it normally and when hovered over. The first method the button has is the instantiation method that allows the button class to inherit from the newText class. The next method is the drawing of the button with the required text inside it. The makeButton method also makes the button change colour when the user hovers the mouse over the button. The checkClicked method is used to check if the button has been clicked and it will print “test” when the button is clicked. I also called the makeButton method inside the checkClicked method so that I only need to call the checkClicked method to make the button. To test that the program knew that the button was being clicked I wrote that it will print “test” if it is clicked. I created a button called to test and I removed the newText object called pause and replaced it with a button. This worked and when I hovered over the button it turned grey and when clicked the button “test” was outputted. At first the position of the pause button was half off the screen but after experimenting with the position I found one that was in a good place.  I wanted to be able to add a function to the button that would run every time the button is clicked. To do this I made another attribute that had an action. I changed the checkClicked method so that when the button is clicked the function will run. to pause the game, I needed to create a screen that would pop up when paused that would allow the user to do various actions. To do this I created a pause function that say that the game is paused and has buttons that will resume the game, send them to the main menu (not created yet) or quit the game. The resume button sends the user to the unpaused function which makes paused equal to false and the quit button sends the user to the quitGame function. |
|  | | | |
|  | | | |
| 12 | Main menu | When the game first loads the main menu will show and there will be buttons that allow the user to navigate the game | Instead of calling the singlePlayer function first I called the mainMenu function. Then I created the mainMenu function I created the title and is used spaces to make the font colour different for the S’ at the beginning and the end. Then I created button that lead to single player mode, multi-player mode, high score board and options. Only the singlePlayer function has been made so the rest of the buttons don’t do anything. At first the positions of the buttons were not equal, but I was able to make it equal with some experimenting. In the end the main menu function did work and when single player was clicked the game started. |
|  | | | |
| 13 | Create game over screen | When the snake dies in single player mode a game over screen will load asking if the user wants to play again. | In the singlePlayer function whenever the snake died, and leave became true then I called the gameOver function first. I made the gameOver function is similar to the pause function and I had the same buttons but instead of the resume button I added play again.  in this stage I also decided that it would be a good idea to add a quit button to the main menu so that the user can quit from there. |
|  | | | |
|  | | | |
| 14 | Creating a screen that tells the user the controls | When the user clicks single player mode the program should tell the user the controls and wait for them to start | To create a screen that tells the user the controls I created text object that the told the user what the control is for up, down, left and right. To start single player mode the user will either must press space or press the next button. To do this I told the user to press space to continue and I created a button that said “next”. When tested everything worked as expected but when the “next” button is pressed the paused screen was loaded. I was extremely confused by this because I did not reference the pause function at all inside the controls function. To test that the singlePlayer function was even running I wrote “print (1)” inside of the function. When I pressed the next button, I found that 1 was output but only once. I thought that the because the next button was in the same place as the pause button in the singlePlayer function the pause button would think that it had been clicked so the pause function would of ran. To test this, I tried pressing the next function again but extremely fast. This time when I pressed it single player did load.  To stop this, I changed the position of the “next” button so that the pause button won’t be automatically clicked. |
|  | | | |
|  | | | |

###### Review

For the buttons I created a button class the inherits from the newText class, this is because I wanted to be able to draw the button box and the text in one method. I created a method that will draw the button and call the addText method and change the colour of the button if the mouse is hovering over it, and I created another method that will call the first method and run a function if the button is clicked. This means that I only need to call one method in order to draw the button and text and run the function if it is clicked.

In used the button class and the newText class to create menus, each different page is its own function and to get between pages buttons need to be pressed to call other functions. This allows me to create a user interface made from functions and buttons to call other functions.

In this sections I was able to create a game over screen which allowed me to ask the user if they would like to play again. Creating a game over screen instead of closing the game makes the game run a real game as all the options are in the players control and it will keep them on the game for longer.

#### MULTIPLAYER MODE

##### Creating snake with multiple players

To create multiplayer mode, I need to create multiple snake inside the same game. to this I need to create a new function that runs the multiplayer mode and I need to create a number of snake objects depending on how many players the user has selected. When the snake collides with another snake then it will need to be removed and I will need to add food that the snakes can use to grow.

|  |  |  |  |
| --- | --- | --- | --- |
| 15 | Pick the number of players for multiplayer | When the player chooses multiplayer mode, they will choose the number of players they want to play with | To let the user, pick the number of players I created a new function that had buttons, so the user could either pick two player, three player or four player. When the multiplayer button is clicked, I would run this function and that function would run the actual game. When the number of players is chosen a function is run that gives a value to the global variable numPlayers, so the game knows how many players there are. |
|  | | | |
| 16 | Creating the snakes and the movement | When the chooses the number of players the snake will be made and begin to move | First, I needed to create the snake object and to do that I used numPlayers to see how many snakes I need to make. Then I ran the createPlayers function in the player functions. Then in the multiplayer function drew the snake using the makeSnake method I added movement using the same format I used in the singlePlayer function. I also added a “dead” attribute to the snake which I can use so that the snake will stop being drawn when the snake dies, and the snake will stop moving.  I realised that if the snake isn’t being drawn the it could still collide with another snake and then the other snake would die without anything visible being in the way. To stop this when the snake is dead, I am going to teleport the snake to coordinates outside the area where it cannot collide with anything else. I did this by changing the coordinates of the 2d array stored inside the snake object to 100,100 which will be outside the playable area so it cannot be played with. |
|  | | | |
|  | | | |
| 17 | Stopping the snakes from colliding | When a snake’s head collides with another snake’s body it will die, when a snake’s head collide with another snake head both snakes will die and when a snake’s head collide with the wall it will die | First, I created a 3d list in test number 16 called allPos which stored the coordinates of each of the snakes in one single list. To check if any of the snakes have died first, I checked if any of the snake’s head are colliding. If any are colliding, then those snakes will die. Then I when through each snake and checked if any of the snake’s head is colliding with any of the other snake’s body. This took lots of lines of code because I couldn’t iterate through each snake array together because each snake would be different lengths, so I had to iterate through them one at a time. When a new snake is added I needed to stop the new snake from going through another snake. After I stopped the snakes from going through each other I needed to stop the snakes if hit the border then it will die. I used two for loops to go through each position of the snake and check that the snake hasn’t gone through wall and if it has then it would die. At first this did not work as the snake would go through the wall by one block and then die. I found that this was the same problem that occurred in the single player function. To stop this, I moved the death code after the movement method is called and then it worked as intended. |
|  | | | |
| 18 | Growing | Food should spawn and when a snake eats it then the snake should grow | I created a food object and then using the allPos list I check if any of the snake heads were touching the food and if they were then that snake would grow. Initially this did not work and when a snake ate the food it became several units longer instead of just one. I realised this was an error with the for loop because the for loop would run several times. To stop this, I added a break to the end of a for loop if the food is eaten so that it won’t repeated. After this was fixed, I ran it again, but the game crashed. The error was that in the death code the index had gone out of range for the allPos variable. This error happened because it would use the length of one of the snakes while iterating. When the iteration reached a snake longer than the others then the index would be out of range as the code would be trying to find coordinates of a shorter snake’s body that isn’t there. To fix this I checked which snake I was iterating through before I ran any code, so I only went through the desired snake. |
|  | | | |

###### Review

In this section I created a pick player function that would store the number of players that the user chooses in a global variable. Using that global variable, I made snake objects depending on the value inside the variable. With all the snakes I added food for them to grow and collision code so that the game knows when the food has been eaten. I also added collision code between other snakes and if they collide, they die by changing an attribute to false which means they won’t be drawn anymore. I also changed the coordinates of the snake to outside the playable area so that the snake cannot kill any other snakes while not being drawn. Stopping the snakes from colliding was a long process and required a lot of code because I needed to make sure that if the snake was touch any of the other snakes in any part of their body then the snake will die. Each unit of the snake has its own coordinates so I created a loop that would check if any of the snakes’ head was colliding with any other snake unit.

##### Adding multiplayer display

In this section I will need to add a screen that appears when the game ends in multiplayer mode. When the game end either a player wins, or a tie is given and in the next round the winner gets a point. I also need to add the controls of each player in this section before the start of the round. The HUD for multiplayer mode needs to be added and it should show the current score of each snake and the number of rounds that each player has won.

|  |  |  |  |
| --- | --- | --- | --- |
| 19 | Multiplayer HUD and winning multiplayer mode | At the bottom the screen the number of points each player has and the number of rounds each player has won and the game should display who won the game | First, I created newText objects that displayed the score of the snakes in the same way I displayed the score in single player mode. Then I created the HUD by drawing a white line that separated the HUD from the rest of the game. I also added a pause button that would allow the user to pause the game if needed. Then to find the winner I used a for loop to check how many players are alive and who the winner is. Using this I can output the winner of the game and I can add a point to their score. In the multiPlayerEnd function I used the winner variable to tell the user who won the game and I could add one more point to the players wins. If there is a tie, then no points are awarded, and the user will be told that it is a tie. Also, I used this function as a game over function so from there they can choose to play again or to go to the main menu or quit the game. If the player chooses to play again then the resetPlayers function will run where the players attributes will be reset, and the multiplayer function will run again. |
|  | | | |
| 20 | Give user their controls for multiplayer mode | When the user picks how many players there are the game will show the user the controls for each player | In the player functions I created text that displayed the up, down, left and right controls for each player. This was simple to do as it was very similar to the controls function but because there was more than one column of text it took some effort to put them in the right place without them overlapping and taking up too much space. In the player4 function I had to display all the controls for all the players, and I moved the controls for player 2 and 3 close because they were quite small and so player 1 and 4 controls took up more space. |
|  | | | |
| 21 | Bug testing | At this point in time I decided to test the game so far with a friend | When playing multiplayer mode, I found that there were many bugs in multiplayer mode. First the snakes were not dying when its head touched its body. Second the game could become choppy due to the large amount of collision checking. First, I needed to add more to the collision checking and see if the snakes head is touching its body, if it is then the snake should die. To stop the code from being choppy I added more if statements in the death code so that code that doesn’t need to be run won’t be run. |
|  | | | |
| 22 | Taking points from other players | When a snake runs into another snake the surviving snake gets longer by the length of the dead snake | When I was testing the game, I decided that the game would be more entertaining if a snake that kills another snake gains the length of that snake. To do when a snake dies the snake that kills it will gain the grow the length of the dead snake. This is done using a for loop which makes the snake grow for the length of the dead snake. At first this did not work and when more than one snake is dead the dead snake’s score increases extremely fast which freezes the game. To stop this, I added if statements that would stop the snake from growing if the snake is dead. |
|  | | | |

###### Review

I added a HUD to multiplayer snake and inside it is the score of each player and how many rounds each player has won. I also added a screen before multiplayer mode starts that tells the user the controls for each player. This required many newText objects as each piece of text is displayed separately. In this stage I needed to add end games results which I did by counting how many snakes did not have coordinates of the dead snakes. When there is one snake alive a win is given to that snake and a round win is given and when zero snakes are alive then a tie is given, and no snake gets a round win. In this stage as I was playing the game, I noticed that the snakes were not dying when they collided with themselves. To stop this, I added more code to check to see if the snake’s head was touching any part of its body. I also had the idea to give an incentive to make the snakes to try and kill the other snakes, to do this I made the snake grow in relation to how long the snake that it killed was. At the moment the database has not been created so I cannot call the users controls from the database. Unfortunately, in pygame anything that has to be drawn needs to have a line of code to do so. This means that I had a lot of code when displaying the users controls because I had to display each piece of text separately.

#### ACCOUNT SYSTEM

##### Adding basics of account system and typing GUI

I need to create a database that will store the controls of each player and the colour for each snake. I will also use this database to store the scores that each player gets and use it for the high score board. I will also create an intro screen in this screen for when the user starts up the game. I need to create textboxes in this sections that let users type inside the game which I am going to use for the log in system. I am also going to create a hashing algorithm that I am going to use for the password being stores. The aim of this section is to aid the completion of the account system by allowing the user to type inside the game and creating necessary functions needed for the account system.

|  |  |  |  |
| --- | --- | --- | --- |
| 23 | Creating an intro screen | When the player loads the screen, the game should ask if the user would like to create an account or log into an account | To create an intro screen, I added two newText objects that show the title. I used two objects with different colours to make the “S” standout like a green snake. There are two buttons that allow the user to go to the log in screen or the register screen. |
|  | | | |
| 24 | Creating a database | When the user starts the game if the database doesn’t exist then it is made | I used SQL to create a database with two tables: accounts and highScores. I wrote “IF NOT EXISTS” so that the tables would keep on being created. The code worked and when it was running the database was made. |
|  | | | |
| 25 | Making textboxes | I need to create a textbox that the user can type in | When logging in and creating an account an account I would need to find a way for the user to type in their username and password. To do this I created a textbox class that inherits newText. I added methods that will draw the box and if the box was clicked then it will change colour. |
|  | | | |
| 26 | Hashing | A hashing function that will hash the user’s password | I need a hashing function that will hash the password to make it unencryptable and completely secure. I would also need a function what would take the entered password and hash it to check if that is the correct password. To do this I created a function that took the password and created a salt (a random hex string) and encoded the password further and stored the salt together with the password. In the second function I split the stored password up into the hashed string and the salt. Then hashed the entered password and used the salt to decode it further and if the passwords were the same then the correct password must have been entered. |
|  | | | |

###### Review

To set up the account system first I need to create an intro screen to my game that asks if the user wants to create a new account or to log in to an account, I did this by using button objects and text objects. To create the database for the game I connected my programme to the database, which created it if it was not made, and if the accounts table or the high score table was not in the database then that was created as well. To make text boxes I made a textBox class that inherits the newText class. I added the makeBox method that would make the place attribute true when it is clicked. I also made the click method that calls the makeBox method and then changes the colour of the box is the place attribute is true. At the end of this section I added a hashing algorithm that takes an input that hashes text and then adds a salt to it as well. I also created a function that will check if an entered a password is the same as a stored password.

##### Registration and logging in

In this section I am creating the login and registration page that I am going to use to store account usernames and passwords and store the current user. I am going to validate the log in system as I am developing it by creating my own account with my own username and password. Using the textBox class, I created in the last section I will need to find a way to take inputs and allow the user to type to what ever textBox is being pressed. When the user makes an unexpectable input I will need to show an error message that will tell the user what went wrong with their input. When the user created an account I will take their password and use the hashing function that I made to hash the password then I need to save the password in the database.

|  |  |  |  |
| --- | --- | --- | --- |
| 27 | Create an account | The user can create a new account with a username and password | To create an account, I needed to add text boxes that allowed the user to type. At first, I used chr(event.key) as the character being added to the string but this did not work. When I typed a character that did not have a symbol an unknown box character would appear. To stop this, I used event.unicode to add a character to the text box. This worked but I found with the tab key a box would also appear. I made an exception for the tab key so that the box selected would change when the tab button was pressed. At first this did work but I noticed a bug where sometimes there would be multiple characters would be added when the key is pressed once. To stop this, I pushed the logic for adding characters inside the for loop for “for event in pygame.event.get()” which meant that it will be pressed only once when a button is pressed. I was able to successfully add the username and password to the database. I was also able to cover up the password by adding asterixis whenever the user pressed a key. |
|  | | | |
| 28 | Log into an account | If the user has an account, then they will need to type in their username and password to go to the main menu | I used what I learnt from the registering function how to add characters to the text and used that in the logIn function. I used the checking hash function to see if the entered password is correct. when testing the hashing function did work and I was only able to log in to the account when I entered the exact same password and username. |
|  | | | |
| 29 | Back button | In the game when the player wants the return to the previous screen, they can press this button | I made the “afk” global and the back function I made the variable false. This made the loop end so the current function would stop running and the previous function would run. |
|  | | | |

I made the registration system function and I created textBox objects and newText objects that tells the user to enter in a username and the password twice. I used the text boxes to let the user type, to do this I added a character to the text attribute of the object and whenever the player pressed backspace I removed one character from the text attribute. When the user has made a username and password and they are the correct length then the username and password are stored inside the database. If they password are not the same, then an error message is shown. For the login system I used a SELECT statement that will take the password of the entered username. If the password is the same as the hashed entered password, then the player will give access to that account. I added a back button that ends the game loop of the current function, so the previous function is loaded.

With the database and account system I will need to store the default controls in the database and then select the controls from the database in game. When the user plays single player mode or multiplayer mode then the controls will need to be shown from the database.

#### HIGH SCORE BOARD AND OPTIONS

##### High score board

In this section I am creating the high score board which will take the scores in the database and show the top twenty. This means I will also have to save any scores made in single player mode with normal mode to the database in the high score table.

|  |  |  |  |
| --- | --- | --- | --- |
| 30 | Creating the high score board | When the player wants the see the high score board, they will see the top twenty scores | To create a high score board, I used sql to select the highScores and the usernames currently stored in the database. Then I wanted to display the top twenty score so I used a for loop that would display the text in order. At first this did not work because the twenty scores would not fit on one column. To fix this I changed the placement of the text so that it would form another column that would show 10 of the scores. In order to get the high scores, I needed to use sql to update the high scores table. |
|  | | | |

In the high score board, I selected the highest twenty scores from the database, and I displayed all of them with their username. I displayed all of them in different places by using a for loop and if the index is greater than a certain amount the position of the text will change. This time I didn’t use the newText class because I would need to create a text object for each high score and display each one which would require lots of code that would take up space. I displayed the text by using a loop that would display each score individually and then I used the index of the loop to change the position of the text so that it would become ordered. In the game over procedure I checked that speed of the game was 160 to make sure the game was in normal mode, then I stored the length of the snake minus three and the account name into the database to be used for the high score board. When there are more than twenty scores in the database only the top twenty are shown.

##### Adding options to allow customizability

In this section I plan to add customisable options such as the ability to change colours and the ability to change the controls the player can use to move their snake. I also plan to do an overview test to see what I can improve in the game which I will I will add. In this section there will be many menus involved so that the user can pick what they want to change, which player they want to change and what they want to change it to. When allowing the user to change the controls I need to make sure that the controls are passed to the database and then called from the database to be used in single player mode and multiplayer mode.

|  |  |  |  |
| --- | --- | --- | --- |
| 31 | Creating an options menu | In the options menu the user can pick if they want to change colours or controls. The user can pick the player they want change | To create an option menu, I needed to a screen that would allow the user to pick if they wanted to change the snake’s colours or controls. I created two buttons that would allow the user to either change the colour or the controls. When the user picks either the colour or controls then the user can either pick player 1-4. I will save which player is selected so when they change the controls or colour then the correct snake will make the desired change. |
|  | | | |
| 32 | Changing player colours | The player selects the player and can change the colour of that snake | When the user selects the change colour and selects a player a function will run that holds the player number that is being changes. I created 8 buttons that had no text on them but had a colour. When a colour is selected the then that colour will update in sql and the current colour will change. This allows the user to customize the snakes colour and the colour change will be recorded in the database. This means that when the user loads up the game again then the colour will be the same as what it was changed to. Originally, I was going to allow the user to also change the background of the game as well, but I realised that it could be problematic as then I would need to change the colours of all the buttons. This means that by changing the colour of the background many of the buttons could blend into the background or be out of place. |
|  | | | |
| 33 | Extra code | Removing extra code | At this point I realised that there were many unnecessary global statements because I did not realise that once a variable had been made global once it does not need to be made global again unless it is one that will be constantly updated. To solve this, I needed to remove all the unnecessary global statements. |
| 34 | Changing the controls | The player selects the player and can change the controls for that snake | I used a similar approach as changing colours to allow me to change controls. When the user picked a player number, I created four text boxes that stored that players up, down, left and right key binds. Then when the user clicked on the control, they wanted to change they can press any button and then the control would change to that button. When the user tried to change the controls to certain keys such as arrow keys or keypad numbers then the name of the key wouldn’t be very useful. To get a better name I created a dictionary called keys and if the key pressed is inside the dictionary then the name that I gave to it will be used instead. |
|  | | | |
| 35 | Speeding up changing the colours and control | Making the colour change quicker and less laggy | I noticed that when I was changing the colours and controls it would take a while for the current colour and controls to change. This happened because it took a while to update the database so whenever the database is updated the game freezes for a bit. To give faster feedback I needed to change the colour to whatever the player clicks and then the database is updated. |
|  | | | |
| 36 | Sound | Adding sound effects | Adding sound effects was very simple, the game plays the WAV file whenever it is called so I added game over and munching noises. When single player mode ends a game over sound is played. In multiplayer mode if a player wins then a fanfare noise is made. I realised at this point I forgot to add the highest score at the bottom of single player mode. I used sql to select the highest score from the database and then I made a newText object that showed the highest score. I also added a sound that would play when a new high score is reached. |
|  | | | |
| 37 | Improving high score board | Removing duplicates | To make the high score board only show the highest score from each player I created a 2d list that stores the highest result from each player. Then contents of the arrays are showed on the high score board. |
|  | | | |

To create an options menu first I created a function that has text asking the user if they want to change the snake colours or the buttons. When the user selects controls or colours then four buttons come up with each one showing the number player they want to change. To let the user change colours, I made a function that that has eight different buttons. I also made a text box that shows the users current colour and when the user clicks a colour button then I will use an update statement to update that players colour. For changing the controls, I used the same method for picked the player number that the user wants to change. I created four textboxes for the controls (up, down, left and right) and when the user selects the text box, they can press a key and the control will change to the key that is pressed. I needed to create a dictionary that saved some of the key names so they will be easily understood. To speed up the changing of the current colour I changed the colour on screen before I changed the colour in the database as the update command can delay the program for about a second. After this I added sound the game when the snake eats and when game ends in single player mode and multiplayer mode. I also improved the scoreboard by only showing the highest score that each player got so that one player won’t dominate the scoreboard.

### Annotated code

import pygame #imports pygame module#

import time #imports time module#

import random #imports random module#

import sqlite3 #imports sql module#

import hashlib #imports hashing library module#

import uuid #imports randomised string module#

pygame.init()#initialises the pygame module#

width=960 #sets the width of the game window to 960 pixels#

height=570 #sets the height of the game window to 570 pixels#

black=(0,0,0) #creates a tuple with rgb values that represents black#

white=(255,255,255) #creates a tuple with rgb values that represents white#

green=(0,176,80) #creates a tuple with rgb values that represents green#

red=(237,28,36) #creates a tuple with rgb values that represents red#

blue=(63,72,204) #creates a tuple with rgb values that represents blue#

Dgrey=(68,68,68) #creates a tuple with rgb values that represents dark grey#

Lgrey=(180,180,180) #creates a tuple with rgb values that represents light grey#

purple=(148,0,255) #creates a tuple with rgb values that represents purple#

yellow=(255,242,0) #creates a tuple with rgb values that represents yellow#

bblue=(153,217,234) #creates a tuple with rgb values that represents baby blue#

orange=(255,127,39) #creates a tuple with rgb values that represents orange#

pink=(255,0,128) #creates a tuple with rgb values that represents pink#

colours={"black":black,"white":white,"green":green,"red":red,"blue":blue,

"dark grey":Dgrey,"light grey":Lgrey,"purple":purple,"yellow":yellow,

"baby blue":bblue,"orange":orange,"pink":pink} #creates a dictionary that links colour name to its rgb tuple#

keys={273:"up arrow",274:"down arrow",276:"left arrow",275:"right arrow",

264:"keypad 8",261:"keypad 5",260:"keypad 4",262:"keypad 6",259:"keypad 3",

258:"keypad 2",257:"keypad 1",263:"keypad 7",265:"keypad 9"} #creates a dictionary that stores Unicode values that links to the name of that key#

munch=pygame.mixer.Sound("munch.wav") #stores a munch sound effect in munch#

gameWin=pygame.mixer.Sound("win.wav") #stores a win sound effect in gameWin#

loss=pygame.mixer.Sound("game over.wav") #stores a game over sound effect is loss#

newHigh=pygame.mixer.Sound("highscore.wav") #stores a high score sound effect for new high scores#

clickNoise=pygame.mixer.Sound("click.wav") #stores a click noise for button clicks#

myFont="Youngsook BTN.ttf" #this saves the youngsook BTN font in myFont#

paused=False #this creates a global variable which will be used for pausing the game#

view=pygame.display.set\_mode((width,height)) #this creates the window for the game with the dimensions of what is stored in the width and height varibles#

pygame.display.set\_caption("Snake With Friends") #this sets the name of the window#

rate=pygame.time.Clock()#this creates a variable that helps to keep track of time#

con=sqlite3.connect("SnakeWithFriends.db") #this connects the program with the SnakeWithFriends database, and it will create the database if it does not exist#

c=con.cursor()#this creates a cursor for the database stored in c#

c.execute("CREATE TABLE IF NOT EXISTS accounts(username TEXT,password TEXT, p1up INTEGAR, p1down INTEGAR, p1left INTEGAR, p1right INTEGAR, p2up INTEGAR, p2down INTEGAR, p2left INTEGAR, p2right INTEGAR, p3up INTEGAR, p3down INTEGAR, p3left INTEGAR, p3right INTEGAR, p4up INTEGAR, p4down INTEGAR, p4left INTEGAR, p4right INTEGAR, p1colour TEXT, p2colour TEXT, p3colour TEXT, p4colour TEXT)") #this creates the table accounts with the written fields if the table does not exist#

c.execute("CREATE TABLE IF NOT EXISTS highScores(username TEXT, score INTEGAR)") #this creates a table called highScores with fields username and score if it does not exist#

def hashy(text): #this creates a function called hashy with a text parameter#

salt = uuid.uuid4().hex #this creates a random string of hex that will be used to encode the password#

return hashlib.sha512(salt.encode()+text.encode()).hexdigest()+':'+salt #this returns the text hashed and encoded with the salt and then adds the salt at the end of the string which is separated by a colon#

def checkPassword(hashed,newPass): #this creates a function with parameters of hashed and newPass#

password, salt = hashed.split(':') #this splits the hashed password and the salt from the colon#

if password == hashlib.sha512(salt.encode() + newPass.encode()).hexdigest():#this takes the entered password and hashes it and encodes it with the same salt used in the first string and checks if it the same as the correct password#

return True #if the passwords are the same then true is returned#

else: #if the passwords are not the same then#

return False #if the passwords are not the same return false#

def intro(): #this creates a function for the intro page on the game#

over=True #this creates a variable that holds a Boolean value that tells the program whether or not to continue the function#

midTitle=newText(" nake With Friend ",80,myFont,white,480,200) #this creates a newText objects that adds the middle of the title to the top of the screen#

endTitle=newText("S S",80,myFont,green,480,200) #this creates a newText objects that adds the ends of the title to the top of the screen#

log=button("-Log in",50,myFont,white,405,400,150,70,black,green,logIn) #this creates a button objects that adds a log in button in the middle of the screen that runs the logIn function#

reg=button("-Register",50,myFont,white,375,300,210,70,black,blue,createAccount) #this creates a button objects that adds a register button above the log in button that runs the createAccount function#

finish=button("Quit",38,myFont,white,886,523,90,50,black,red,quitGame) #this creates a button objects that adds a quit button that exits the game if pressed#

global event #this sets event to a global variable#

while over==True: #this starts a loop that continues while over is true#

for event in pygame.event.get():#this gets all the events that happen in the game#

if event.type == pygame.QUIT: #if the player pressed the exit button then#

quitGame() #run the quitGame function#

global mouse #this sets mouse to a global variable#

global click #this sets click to a global variable#

mouse=pygame.mouse.get\_pos()#this gets the position of the mouse and saves it in the mouse variable#

click=pygame.mouse.get\_pressed()#this checks if any of the buttons on the mouse are being clicked#

view.fill(black) #this sets the whole display to black#

midTitle.showJustText()#this shows the midTitle text#

endTitle.showJustText()#this shows the endTitle text#

log.checkClicked()#this runs the checkClicked method for log#

reg.checkClicked()#this runs the checkClicked method for reg #

finish.checkClicked()#this runs the checkClicked method for finish#

pygame.display.flip()#this updates the screen and displays anything that has been sent to display#

def logIn():#this creates a function called logIn#

global afk #this sets afk to a global variable#

global account #this sets account to a global variable#

account="" #this sets account to an empty string#

afk=True #this sets afk to true#

tempPass="" #this sets tempPass to an empty string #

wrong=False #this sets wrong to false#

title=newText("Sign In",70,myFont,green,480,70) #this creates an object that says Sign In#

userN=newText("Enter Your Username",50,myFont,blue,480,160) #this creates a text object that asks the user for their username#

givenUser=textBox("",50,None,black,230,200,500,35,Lgrey,white) #this creates a textbox that is empty#

passW=newText("Enter Your Password",50,myFont,blue,480,360) #this creates a text object that asks for the users password#

givenPass=textBox("",50,None,black,230,400,500,35,Lgrey,white) #this creates a textbox that will be used to let the user enter their password#

finish=button("Back",38,myFont,white,0,0,100,50,black,green,back) #this creates a back button#

enter=newText("Press Enter To Continue",20,myFont,white,480,550) #this creates a text object that tells the user to enter to continue#

end=button("Quit",38,myFont,white,886,523,90,50,black,red,quitGame) #this creates a button that lets the user quits#

wrongUP=newText("Your Username Or Password Is Incorrect",40,myFont,red,480,510) #this creates a text object that tells the user that their username or password is incorrect#

global event

while afk==True:

for event in pygame.event.get()

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN: #if a key press event occurs then#

if event.key == pygame.K\_RETURN or event.key == pygame.K\_TAB: #if the key is return or tab then#

givenUser.place=False #the given user textbox is no longer selected#

givenPass.place=True #the given pass textbox has now been selected#

if givenUser.place == True: #if given user is selected then#

if event.key == pygame.K\_BACKSPACE: #if the key pressed is backspace then#

givenUser.text=givenUser.text[:-1] #remove one character from the text inside the textbox#

elif event.key == pygame.K\_RETURN or event.key == pygame.K\_TAB: #if the key pressed isn’t backspace but it is return or tab then#

givenUser.place=False #the given user textbox is no longer selected#

givenPass.place=True #the given pass textbox is selected#

elif len(givenUser.text) < 15: #else if the length of the text is less than 15 characters then#

givenUser.text=givenUser.text+event.unicode #add the character pressed to the text inside the textbox#

if givenPass.place == True: #if the given pass textbox is selected then#

if event.key == pygame.K\_RETURN: #if the button pressed it enter then#

givenPass.place=False #the given pass textbox is no longer selected#

c.execute("SELECT password FROM accounts WHERE username=?",

(givenUser.text,)) #select the password from the accounts table when the username is the text that the user entered#

password=c.fetchall() #all the data that was requested is saved in password#

if password != []: #if the result was not empty then#

if checkPassword(password[0][0],tempPass) == True: #if the result of the password checking function is true then the password is correct then#

account=givenUser.text #account contains the username that was entered#

mainMenu() #run the mainMenu function#

wrong=True #wrong is now true#

elif event.key == pygame.K\_BACKSPACE: #else if the key is backspace then#

givenPass.text=givenPass.text[:-1] #the text inside the given pass textbox becomes one character shorter#

tempPass=tempPass[:-1] #remove the last character#

elif event.key == pygame.K\_TAB: #else if the key is tab then#

pass #do nothing#

elif len(givenPass.text) < 30: #if the length of the text entered is less than 30#

givenPass.text="" #the text inside the textbox is empty#

tempPass=tempPass+event.unicode #add the button pressed to the text#

for i in range(0,len(tempPass)): #repeat the loop from 0 to the length of tempPass#

givenPass.text=givenPass.text+"\*" #add an Asterix to text#

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

title.showJustText()

givenUser.click()

userN.showJustText()

passW.showJustText()

givenPass.click()

finish.checkClicked()

enter.showJustText()

end.checkClicked()

if wrong == True:

wrongUP.showJustText()

pygame.display.flip()

def createAccount():#this creates a function called createAccount#

global afk

afk=True

wrong=False

taken=False

short=False

tempPass1=""

tempPass2=""

title=newText("Register Account",70,myFont,green,480,70)

userN=newText("Enter Your Username 3-15 Characters",40,myFont,blue,480,140)

givenUser=textBox("",50,None,black,230,170,500,35,Lgrey,white)

passW1=newText("Enter Your Password 5-30 Characters)",40,myFont,blue,480,270)

givenPass1=textBox("",50,None,black,230,300,500,35,Lgrey,white)

passW2=newText("Enter Your Password Again 5-30 Characters",40,myFont,blue,480,400)

givenPass2=textBox("",50,None,black,230,430,500,35,Lgrey,white)

wrongPass=newText("The Passwords Do Not Match",40,myFont,red,480,510)

finish=button("Back",38,myFont,white,0,0,100,50,black,green,back)

userTaken=newText("That Username Is Taken",40,myFont,red,480,510)

shortUP=newText("Your Username Or Password Is Too Short",40,myFont,red,480,510)

enter=newText("Press Enter To Continue",20,myFont,white,480,550)

end=button("Quit",38,myFont,white,886,523,90,50,black,red,quitGame)

global event

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_RETURN:

givenUser.place=False

givenPass1.place=False

givenPass2.place=True

if event.key == pygame.K\_TAB:

afk=True

if givenUser.place == True:

if event.key == pygame.K\_BACKSPACE:

givenUser.text=givenUser.text[:-1]

elif len(givenUser.text) < 15:

givenUser.text=givenUser.text+event.unicode

if givenPass1.place == True:

if event.key == pygame.K\_BACKSPACE:

givenPass1.text=givenPass1.text[:-1]

tempPass1=tempPass1[:-1]

elif len(givenPass1.text) < 30:

givenPass1.text=givenPass1.text+"\*"

tempPass1=tempPass1+event.unicode

if givenPass2.place == True:

if event.key == pygame.K\_RETURN:

givenPass2.place=False

taken=False

wrong=False

short=False

if len(givenUser.text) > 2 and len(givenPass1.text) > 4: #if the length of the username text is greater than two and the length of the password text is greater than 4 then#

short=False #short is false#

if tempPass1 == tempPass2: #if both passwords entered are the same then#

wrong=False #wrong is false#

c.execute("SELECT username FROM accounts WHERE username=?",

(givenUser.text,)) #select usernames from the accounts table where the username is the same as the entered username#

if c.fetchall() == []:#if nothing is returned then#

c.execute("INSERT INTO accounts(username,password,p1up,p1down,p1left,p1right,p2up,p2down,p2left,p2right,p3up,p3down,p3left,p3right,p4up,p4down,p4left,p4right,p1colour,p2colour,p3colour,p4colour)VALUES(?,?,273,274,276,275,119,115,97,100,105,107,106,108,264,261,260,262,'green','blue','red','yellow')", #create a new record with thee username and password#

(givenUser.text,hashy(tempPass1))) #this stores the created username in the database and the hashed password#

con.commit()#this commits all the data to the database#

logIn() #this calls the log in function#

else:

taken=True

else:

wrong=True

else:

short=True

elif event.key == pygame.K\_BACKSPACE:

givenPass2.text=givenPass2.text[:-1]

tempPass2=tempPass2[:-1]

elif event.key == pygame.K\_TAB:

pass

elif len(givenPass1.text) < 30:

givenPass2.text=givenPass2.text+"\*"

tempPass2=tempPass2+event.unicode

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

title.showJustText()

givenUser.click()

userN.showJustText()

passW1.showJustText()

givenPass1.click()

passW2.showJustText()

givenPass2.click()

enter.showJustText()

if wrong == True:

wrongPass.showJustText()

if taken == True:

userTaken.showJustText()

if short == True:

shortUP.showJustText()

end.checkClicked()

finish.checkClicked()

pygame.display.flip()

class snake: #this creates a class called snake#

def \_\_init\_\_(self,coordinates,snakew,snakeh,colour,length,direction,up,down,left,right,score): #this is the instantiation method for all the attributes of the snake class#

self.coordinates=coordinates

self.snakew=snakew

self.snakeh=snakeh

self.colour=colour

self.length=length

self.direction=direction

self.up=up

self.down=down

self.left=left

self.right=right

self.dead=False

self.score=score

def makeSnake(self): #this is the draw snake method#

for i in range(0,len(self.coordinates)): #this repeats the loop as many times as the length of the snake#

pygame.draw.rect(view,self.colour, [self.coordinates[i][0]\*20,self.coordinates[i][1]\*20,self.snakew,self.snakeh]) #this commands draws one square of the snake#

def grow(self): #this makes a grow method#

pygame.mixer.Sound.play(munch) #this plays a munch noise#

self.coordinates=self.coordinates+[[self.coordinates[len(self.coordinates)-1][0],self.coordinates[len(self.coordinates)-1][1]]] #this makes the snake add one unit to itself#

self.length=self.length+1 #this sets the length of the snake to one longer#

def changeDir(self,event): #this creates a change direction method#

if event.key == self.right and self.coordinates[0][0] != self.coordinates[1][0]-1: #if the player presses the right control and the snake is not facing left then#

self.direction="right" #the snake now faces right#

elif event.key == self.left and self.coordinates[0][0] != self.coordinates[1][0]+1: #if the player presses the left control and the snake is not facing right then#

self.direction="left" #the snake now faces left#

elif event.key == self.up and self.coordinates[0][1] != self.coordinates[1][1]+1: #if the player presses the up control and the snake is not facing down then#

self.direction="up" #the snake now faces up#

elif event.key == self.down and self.coordinates[0][1] != self.coordinates[1][1]-1: #if the player presses the down control and the snake is not facing up then#

self.direction="down" #the snake now faces down#

def movement(self): #this creates a movement method#

for i in range(len(self.coordinates)-1,0,-1): #this starts a loop from the length of the snake to 0 with a step of -1#

self.coordinates[i][0]=self.coordinates[i-1][0]

#this moves each unit of the snake to the position of the previous unit and using the direction that the snake is facing it moves the snakes head by one unit#

self.coordinates[i][1]=self.coordinates[i-1][1]

if self.direction == "right":

self.coordinates[0][0]=self.coordinates[0][0]+1

elif self.direction == "left":

self.coordinates[0][0]=self.coordinates[0][0]-1

elif self.direction == "up":

self.coordinates[0][1]=self.coordinates[0][1]-1

elif self.direction =="down":

#this is the food class that has coordinate attributes and colour attributes. The createFood method draws the food and the spawn method changes the coordinates of the snake when eated.#

self.coordinates[0][1]=self.coordinates[0][1]+1

class food:

def \_\_init\_\_(self,x\_co,y\_co,colour):

self.x\_co=x\_co

self.y\_co=y\_co

self.colour=colour

def createFood(self):

pygame.draw.rect(view,self.colour, [self.x\_co\*20,self.y\_co\*20,20,20])

def spawn(self):

self.x\_co = random.randint(0,47)

self.y\_co = random.randint(0,25)

class newText:

def \_\_init\_\_(self,text,size,font,colour,xplace,yplace):

self.text=text

self.size=size

#this creates an object called newText that is going to be used to create text on the screen. By editting the attributes I can change the position of the text and the size and colour. The showJustText takes a centre point and draws the text around that point. The addText method draws the text from the top right corner to the end.#

self.font=font

self.colour=colour

self.xplace=xplace

self.yplace=yplace

def mainText(self,textProp):

textSquare=textProp.render(self.text,True,self.colour)

return textSquare,textSquare.get\_rect()

def showJustText(self):

textType=pygame.font.Font(self.font,self.size)

textS,textR=self.mainText(textType)

textR.center=(self.xplace,self.yplace)

view.blit(textS,textR)

def addText(self):

font=pygame.font.Font(self.font,self.size)

text=font.render(self.text,False,self.colour)

view.blit(text,(self.xplace,self.yplace))

class textBox(newText):

def \_\_init\_\_(self,text,size,font,colour,xplace,yplace,width,height,bColour,newColour):

newText.\_\_init\_\_(self,text,size,font,colour,xplace,yplace)

self.width=width

self.height=height

self.bColour=bColour

self.newColour=newColour

self.place=False

def makeBox(self):

if self.xplace < mouse[0] < (self.xplace+self.width) and self.yplace < mouse[1] < (self.yplace+self.height) and click[0]==1:

self.place=True

if ((self.xplace > mouse[0] or mouse[0] > (self.xplace+self.width)) or (self.yplace > mouse[1] or mouse[1] > (self.yplace+self.height))):

if click[0]==1:

self.place=False

#the textbox class inherits from the newText class and has more attributes that control the instructiveness of the box. The click draws the box and changes the colour depending on the place attribute’s value. The click method runs the makeBox method which checks if the text box has been selected.#

def click(self):

self.makeBox()

if self.place == True:

pygame.draw.rect(view,self.newColour, [self.xplace,self.yplace,self.width,self.height])

else:

pygame.draw.rect(view,self.bColour, [self.xplace,self.yplace,self.width,self.height])

self.addText()

class button(textBox):

def \_\_init\_\_(self,text,size,font,colour,xplace,yplace,width,height,bColour,newColour,action=None):

textBox.\_\_init\_\_(self,text,size,font,colour,xplace,yplace,width,height,bColour,newColour)

self.action=action

def makeButton(self):

if self.xplace < mouse[0] < (self.xplace+self.width) and self.yplace < mouse[1] < (self.yplace+self.height):

pygame.draw.rect(view,self.newColour, [self.xplace,self.yplace,self.width,self.height])

#the button class inherits from the textbox class and adds an action attribute which is a name of a function. Using the makeButton method I can draw the button and if the mouse if hovering over the button then it will be change colour. The checkClicked method runs the makeButton method and runs a function if the button is clicked.#

self.place=True

else:

pygame.draw.rect(view,self.bColour, [self.xplace,self.yplace,self.width,self.height])

self.place=False

self.addText()

def checkClicked(self):

self.makeButton()

if self.place==True and click[0]==1 and self.action != None:

pygame.mixer.Sound.play(clickNoise)

self.action()

def clicked(self):

self.makeButton()

if self.place==True and click[0]==1:

return True

else:

return False

def mainMenu():

menu=True

#in the main menu function I created the single player, multiplayer, high score board and options buttons for the game with the title of the game at the top#

global count

global waitTime

global difficulty

count=0

waitTime=160

difficulty="normal"

midHeading=newText(" nake With Friend ",60,myFont,white,480,100)

endHeading=newText("S S",60,myFont,green,480,100)

singlemode=button("Single Player",30,myFont,black,61,300,175,45,white,blue,controls)

multimode=button("MultiPlayer",30,myFont,black,297,300,155,45,white,blue,pickPlayer)

scores=button("High Score Board",30,myFont,black,513,300,230,45,white,blue,highScoreBoard)

options=button("Options",30,myFont,black,804,300,95,45,white,blue,option)

pickAccount=button("Sign Out",38,myFont,white,820,0,140,50,black,blue,intro)

end=button("Quit",38,myFont,white,886,523,90,50,black,red,quitGame)

global event

while menu==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

midHeading.showJustText()

endHeading.showJustText()

singlemode.checkClicked()

multimode.checkClicked()

scores.checkClicked()

options.checkClicked()

end.checkClicked()

pickAccount.checkClicked()

pygame.display.flip()

def keyName(asc): #creates a function called keyName with parameter asc#

if asc in keys: #if asc is inside the key dictionary then#

return keys[asc] #return the value stores in asc of the dictionary#

else: #if not then#

return pygame.key.name(asc) #return pygames saved key name for that value#

def quitGame():#creates a function that’s quits the game#

con.commit() #this commits all the data saved in the database into memory#

c.close() #this closes the cursor#

con.close() #this closes the connection between the database and the program#

pygame.quit() #this closes pygame#

quit() #this closes the python window#

def resume():

global paused

paused = False

def pause():

global paused

paused=True

timer=0

pauseText=newText("Paused",60,myFont,green,480,260)

cont=button("-Resume",38,myFont,white,400,350,140,50,black,green,resume)

menu=button("-Main menu",38,myFont,white,400,400,190,50,black,blue,mainMenu)

end=button("-Quit",38,myFont,white,400,450,90,50,black,red,quitGame)

while paused==True:

#the pause function creates a menu that has buttons that will send the user back the game, to the main menu or to quit the game.#

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

pauseText.showJustText()

cont.checkClicked()

menu.checkClicked()

end.checkClicked()

pygame.display.flip()

def back():

global afk

afk = False

def makep1():

global account

global p1

global up1

global down1

global left1

global right1

global colour1

c.execute("SELECT p1up, p1down, p1left, p1right, p1colour FROM accounts WHERE username=?",

(account,))

p1=c.fetchall()

up1=keyName(p1[0][0])

down1=keyName(p1[0][1])

left1=keyName(p1[0][2])

right1=keyName(p1[0][3])

colour1=colours[p1[0][4]]

def makep2():

global account

global p2

global up2

global down2

global left2

global right2

global colour2

c.execute("SELECT p2up, p2down, p2left, p2right, p2colour FROM accounts WHERE username=?",

(account,))

p2=c.fetchall()

if p2[0][0] in keys:

up2=keys[p2[0][0]]

else:

up2=pygame.key.name(p2[0][0])

#I made four make player functions that made a snake and selected the controls from the database. I used these functions to get the controls for the player to display it on screen and to get the colour for that player.#

if p2[0][1] in keys:

down2=keys[p1[0][1]]

else:

down2=pygame.key.name(p2[0][1])

if p2[0][2] in keys:

left2=keys[p2[0][2]]

else:

left2=pygame.key.name(p2[0][2])

if p2[0][3] in keys:

right2=keys[p2[0][3]]

else:

right2=pygame.key.name(p2[0][3])

colour2=colours[p2[0][4]]

def makep3():

global account

global p3

global up3

global down3

global left3

global right3

global colour3

c.execute("SELECT p3up, p3down, p3left, p3right, p3colour FROM accounts WHERE username=?",

(account,))

p3=c.fetchall()

if p3[0][0] in keys:

up3=keys[p3[0][0]]

else:

up3=pygame.key.name(p3[0][0])

if p3[0][1] in keys:

down3=keys[p3[0][1]]

else:

down3=pygame.key.name(p3[0][1])

if p3[0][2] in keys:

left3=keys[p3[0][2]]

else:

left3=pygame.key.name(p3[0][2])

if p3[0][3] in keys:

right3=keys[p3[0][3]]

else:

right3=pygame.key.name(p3[0][3])

colour3=colours[p3[0][4]]

def makep4():

global account

global p4

global up4

global down4

global left4

global right4

global colour4

c.execute("SELECT p4up, p4down, p4left, p4right, p4colour FROM accounts WHERE username=?",

(account,))

p4=c.fetchall()

if p4[0][0] in keys:

up4=keys[p4[0][0]]

else:

up4=pygame.key.name(p4[0][0])

if p4[0][1] in keys:

down4=keys[p4[0][1]]

else:

down4=pygame.key.name(p4[0][1])

if p4[0][2] in keys:

left4=keys[p4[0][2]]

else:

left4=pygame.key.name(p4[0][2])

if p4[0][3] in keys:

right4=keys[p4[0][3]]

else:

right4=pygame.key.name(p4[0][3])

colour4=colours[p4[0][4]]

def controls():

global up1

global down1

global left1

global right1

global colour1

makep1()

upText=newText("Up - "+up1,40,myFont,colour1,480,150)

downText=newText("Down - "+down1,40,myFont,colour1,480,210)

leftText=newText("Left - "+left1,40,myFont,colour1,480,270)

rightText=newText("Right - "+right1,40,myFont,colour1,480,330)

bigText=newText("Controls",60,myFont,purple,480,40)

endText=newText("Press space to continue",30,myFont,yellow,480,450)

Next=button("Next",38,myFont,white, 435,500,90,50,black,blue,singlePlayer)

finish=button("Back",38,myFont,white,0,0,100,50,black,blue,back)

global afk

#in the control function I made I used newText objects to display the player controls. When the user presses space or the next button then the singlePlayer function will run.#

afk=True

global event

while afk == True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

afk=False

singlePlayer()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

upText.showJustText()

downText.showJustText()

leftText.showJustText()

rightText.showJustText()

bigText.showJustText()

endText.showJustText()

Next.checkClicked()

finish.checkClicked()

pygame.display.flip()

def singlePlayer(): #this creates the single player function#

c.execute("SELECT score FROM highScores ORDER BY score DESC LIMIT 1") #this selects the highest score from the highScores table, and it selects the largest one#

highScore=c.fetchall() #this saves the data from the database in highScore#

global player1 #this sets player1 to be a global variable#

Food=food(random.randint(0,47),random.randint(0,25),red) #this created a food object with random coordinates called Food#

timer=0

leave=False

global p1

global colour1

player1=snake([[2,15],[1,15],[0,15]],20,20,colour1,3,"right",p1[0][0],p1[0][1],p1[0][2],p1[0][3],0) #this creates a snake object calls player1 with the controls and colour from the database#

score=newText("Score: "+str((player1.length-3)),40,myFont,white,0,520)

stop=button("Pause",38,myFont,white,850,523,130,50,black,purple,pause)

if len(highScore) != 0: #if there is not nothing in highScore then#

highScoreText=newText("Highscore: "+str(highScore[0][0]),40,myFont,white,300,520) #create a high score text to show the highest score#

global event

while leave==False:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN: #if the user presses a key then#

player1.changeDir(event) #this runs the change direction method for player1#

if event.key == pygame.K\_p:

paused=True

pause()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

player1.makeSnake()

if player1.coordinates[0][0] == Food.x\_co and player1.coordinates[0][1] == Food.y\_co: #if the player coordinates are the same as the foods then#

player1.grow() #player1 grows#

score.text="Score: "+str(player1.length-3) #increase the score by 1#

if len(highScore) != 0: #if highScore isn’t empty then#

if player1.length-3 > highScore[0][0]: #if player1’s score is greater than the highScore#

highScoreText.text="Highscore: "+str(player1.length-3) #this changed the high score to the players score#

elif player1.length-3 == highScore[0][0]: #else if the player score is equal to the high score then#

pygame.mixer.Sound.play(newHigh) #play a new high score sound#

for i in range(0,len(player1.coordinates)): #repeat loop for the length of the player coordinate times#

if player1.coordinates[i][0] == Food.x\_co and player1.coordinates[i][1] == Food.y\_co: #if the player coordinates intersect the food coordinates then#

Food.spawn()#change the position of the food#

Food.createFood()#draw the food#

pygame.draw.rect(view,white, [0,520,960,3]) #draw the HUD line#

score.addText()

if len(highScore) != 0: #if highScore is not empty then#

highScoreText.addText()#draw the highscore text#

stop.checkClicked()

pygame.display.flip()

speed=rate.tick()#speed is equal to the number of frames past#

timer=timer+speed #adds the number of frames that have passed to timer#

if timer > waitTime: #if the timer is greater the game refresh time then#

player1.movement()#player1 moves#

timer=0

if player1.coordinates[0][0] >= 48 or player1.coordinates[0][0] < 0: #if the player coordinates are outside the range then#

gameOver() #run the gameOver function#

if player1.coordinates[0][1] >= 26 or player1.coordinates[0][1] < 0: #if the player is outside the range then#

gameOver()#runt he game over function#

for i in range(2,len(player1.coordinates)): #for 2 the length of player 1 repeat#

if player1.coordinates[0][0] == player1.coordinates[i][0] and player1.coordinates[0][1] == player1.coordinates[i][1]: #if the coordinates are the same as the snake heads then#

gameOver()#run the gameOver function#

def gameOver():

pygame.mixer.Sound.play(loss) #play the loss sound effect#

if waitTime == 160:

c.execute("INSERT INTO highScores(username,score)VALUES(?,?)",

(account,player1.length-3)) #update the highscore table with the score the user got#

con.commit() #commit the data to the database#

global count

global difficulty

over=True

if player1.length-3 < 20: #if the score is less than 20 then#

count=count-1 #remove one from count#

elif player1.length-3 >= 60: #if the score is greater than or equal to 60#

count=count+1 #increase count by 1#

else:

count=0

bigText=newText("Game Over",60,myFont,green,480,260)

cont=button("-Play again",38,myFont,white,400,350,195,50,black,green,singlePlayer)

menu=button("-Main menu",38,myFont,white,400,400,190,50,black,blue,mainMenu)

end=button("-Quit",38,myFont,white,400,450,90,50,black,red,quitGame)

if count == -3 and difficulty == "normal":

cont.action=easy

elif count == 3 and difficulty == "normal":

cont.action=hard

if count > 0 and difficulty == "easy":

cont.action=normal

if count < 0 and difficulty == "hard":

cont.action=normal

global event

while over==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

bigText.showJustText()

cont.checkClicked()

menu.checkClicked()

end.checkClicked()

pygame.display.flip()

def easy():

over=True

bigText=newText("Would you like to play on easy mode?",60,myFont,blue,480,260)

yes=button("Yes",38,myFont,black,240,400,65,50,white,green,makeEasy)

no=button("No",38,myFont,black,650,400,50,50,white,red,resetCount)

while over==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

bigText.showJustText()

yes.checkClicked()

no.checkClicked()

pygame.display.flip()

def makeEasy():

global waitTime

global difficulty

global count

waitTime=200

difficulty="easy"

count=0

singlePlayer()

def normal():

over=True

bigText=newText("Would you like to play on normal mode?",60,myFont,blue,480,260)

yes=button("Yes",38,myFont,black,240,400,65,50,white,green,makeNormal)

no=button("No",38,myFont,black,650,400,50,50,white,red,resetCount)

while over==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

#in these functions I asked the user if they wanted to change the difficulty of the game. If they did then I would change the wait time to make the snake move faster or slower.#

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

bigText.showJustText()

yes.checkClicked()

no.checkClicked()

pygame.display.flip()

def makeNormal():

global waitTime

global difficulty

global count

waitTime=160

difficulty="normal"

count=0

singlePlayer()

def hard():

over=True

bigText=newText("Would you like to play on Hard mode?",60,myFont,blue,480,260)

yes=button("Yes",38,myFont,black,240,400,65,50,white,green,makeHard)

no=button("No",38,myFont,black,650,400,50,50,white,red,resetCount)

while over==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

bigText.showJustText()

yes.checkClicked()

no.checkClicked()

pygame.display.flip()

def makeHard():

global waitTime

global difficulty

global count

waitTime=120

difficulty="hard"

count=0

singlePlayer()

def resetCount():

global count

count=0

singlePlayer()

def pickPlayer():

choice=newText("Choose the number of players",60,myFont,blue,480,100)

play2=button("2 Players",38,myFont,black,120,345,165,50,white,yellow,players2)

play3=button("3 Players",38,myFont,black,405,345,155,50,white,bblue,players3)

play4=button("4 Players",38,myFont,black,680,345,160,50,white,purple,players4)

finish=button("Back",38,myFont,white,0,0,100,50,black,green,back)

global numPlayers

numPlayers=1

#in this function I asked the user to pick how many player they wanted to play with in multiplayer mode#

global afk

afk=True

global event

while afk == True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

choice.showJustText()

play2.checkClicked()

play3.checkClicked()

play4.checkClicked()

finish.checkClicked()

pygame.display.flip()

def players2():

global numPlayers

global up1

global down1

global left1

global right1

global colour1

makep1()

global up2

global down2

global left2

global right2

global colour2

makep2()

numPlayers=2

createPlayers()

p1Text=newText("Player 1",40,myFont,colour1,240,100)

p1upText=newText("Up - "+up1,40,myFont,colour1,240,160)

p1downText=newText("Down - "+down1,40,myFont,colour1,240,220)

p1leftText=newText("Left - "+left1,40,myFont,colour1,240,280)

p1rightText=newText("Right - "+right1,40,myFont,colour1,240,340)

p2Text=newText("Player 2",40,myFont,colour2,720,100)

p2upText=newText("Up - "+up2,40,myFont,colour2,720,160)

p2downText=newText("Down - "+down2,40,myFont,colour2,720,220)

p2leftText=newText("Left - "+left2,40,myFont,colour2,720,280)

p2rightText=newText("Right - "+right2,40,myFont,colour2,720,340)

bigText=newText("Controls",50,myFont,purple,480,30)

endText=newText("Press space to continue",30,myFont,bblue,480,480)

Next=button("Next",38,myFont,white,435,520,90,50,black,blue,multiPlayer)

finish=button("Back",38,myFont,white,0,520,100,50,black,green,pickPlayer)

global afk

afk=True

global event

while afk == True:

for event in pygame.event.get():

#in this function I gave the user the controls for player 1 and 2#

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

afk=False

multiPlayer()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

p1Text.showJustText()

p1upText.showJustText()

p1downText.showJustText()

p1leftText.showJustText()

p1rightText.showJustText()

p2Text.showJustText()

p2upText.showJustText()

p2downText.showJustText()

p2leftText.showJustText()

p2rightText.showJustText()

bigText.showJustText()

endText.showJustText()

Next.checkClicked()

finish.checkClicked()

pygame.display.flip()

def players3():

global up1

global down1

global left1

global right1

global colour1

makep1()

#in this function I gave the user the controls for player 1,2 and 3#

global up2

global down2

global left2

global right2

global colour2

makep2()

global up3

global down3

global left3

global right3

global colour3

makep3()

global numPlayers

numPlayers=3

createPlayers()

p1Text=newText("Player 1",40,myFont,colour1,200,100)

p1upText=newText("Up - "+up1,40,myFont,colour1,200,160)

p1downText=newText("Down - "+down1,40,myFont,colour1,200,220)

p1leftText=newText("Left - "+left1,40,myFont,colour1,200,280)

p1rightText=newText("Right - "+right1,40,myFont,colour1,200,340)

p2Text=newText("Player 2",40,myFont,blue,480,100)

p2upText=newText("Up - "+up2,40,myFont,colour2,480,160)

p2downText=newText("Down - "+down2,40,myFont,colour2,480,220)

p2leftText=newText("Left - "+left2,40,myFont,colour2,480,280)

p2rightText=newText("Right - "+right2,40,myFont,colour2,480,340)

p3Text=newText("Player 3",40,myFont,colour3,760,100)

p3upText=newText("Up - "+up3,40,myFont,colour3,760,160)

p3downText=newText("Down - "+down3,40,myFont,colour3,760,220)

p3leftText=newText("Left - "+left3,40,myFont,colour3,760,280)

p3rightText=newText("Right - "+right3,40,myFont,colour3,760,340)

bigText=newText("Controls",50,myFont,purple,480,30)

endText=newText("Press space to continue",30,myFont,bblue,480,480)

Next=button("Next",38,myFont,white,435,520,90,50,black,blue,multiPlayer)

finish=button("Back",38,myFont,white,0,520,100,50,black,green,pickPlayer)

global afk

afk=True

global event

while afk == True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

afk=False

multiPlayer()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

p1Text.showJustText()

p1upText.showJustText()

p1downText.showJustText()

p1leftText.showJustText()

p1rightText.showJustText()

p2Text.showJustText()

p2upText.showJustText()

p2downText.showJustText()

p2leftText.showJustText()

p2rightText.showJustText()

p3Text.showJustText()

p3upText.showJustText()

p3downText.showJustText()

p3leftText.showJustText()

p3rightText.showJustText()

bigText.showJustText()

endText.showJustText()

Next.checkClicked()

finish.checkClicked()

pygame.display.flip()

def players4():

global up1

global down1

global left1

global right1

global colour1

makep1()

global up2

global down2

global left2

global right2

global colour2

makep2()

global up3

global down3

global left3

global right3

global colour3

makep3()

global up4

global down4

#in this function I gave the user the controls for player 1,2,3 and 4#

global left4

global right4

global colour4

makep4()

global numPlayers

numPlayers=4

createPlayers()

p1Text=newText("Player 1",30,myFont,colour1,160,100)

p1upText=newText("Up - "+up1,30,myFont,colour1,160,160)

p1downText=newText("Down - "+down1,30,myFont,colour1,160,220)

p1leftText=newText("Left - "+left1,30,myFont,colour1,160,280)

p1rightText=newText("Right - "+right1,30,myFont,colour1,160,340)

p2Text=newText("Player 2",30,myFont,colour2,400,100)

p2upText=newText("Up - "+up2,30,myFont,colour2,400,160)

p2downText=newText("Down - "+down2,30,myFont,colour2,400,220)

p2leftText=newText("Left - "+left2,30,myFont,colour2,400,280)

p2rightText=newText("Right - "+right2,30,myFont,colour2,400,340)

p3Text=newText("Player 3",30,myFont,colour3,600,100)

p3upText=newText("Up - "+up3,30,myFont,colour3,600,160)

p3downText=newText("Down - "+down3,30,myFont,colour3,600,220)

p3leftText=newText("Left - "+left3,30,myFont,colour3,600,280)

p3rightText=newText("Right - "+right3,30,myFont,colour3,600,340)

p4Text=newText("Player 4",30,myFont,colour4,820,100)

p4upText=newText("Up - "+up4,30,myFont,colour4,820,160)

p4downText=newText("Down - "+down4,30,myFont,colour4,820,220)

p4leftText=newText("Left - "+left4,30,myFont,colour4,820,280)

p4rightText=newText("Right - "+right4,30,myFont,colour4,820,340)

bigText=newText("Controls",50,myFont,purple,480,30)

endText=newText("Press space to continue",30,myFont,bblue,480,480)

Next=button("Next",38,myFont,white,435,520,90,50,black,blue,multiPlayer)

finish=button("Back",38,myFont,white,0,520,100,50,black,green,pickPlayer)

global afk

afk=True

global event

while afk == True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE:

afk=False

multiPlayer()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

p1Text.showJustText()

p1upText.showJustText()

p1downText.showJustText()

p1leftText.showJustText()

p1rightText.showJustText()

p2Text.showJustText()

p2upText.showJustText()

p2downText.showJustText()

p2leftText.showJustText()

p2rightText.showJustText()

p3Text.showJustText()

p3upText.showJustText()

p3downText.showJustText()

p3leftText.showJustText()

p3rightText.showJustText()

p4Text.showJustText()

p4upText.showJustText()

p4downText.showJustText()

p4leftText.showJustText()

p4rightText.showJustText()

bigText.showJustText()

#in the createPlayers function I created the snake object for all the player#

endText.showJustText()

Next.checkClicked()

finish.checkClicked()

pygame.display.flip()

def createPlayers():

global p1

global colour1

global player1

player1=snake([[0,23],[0,24],[0,25]],20,20,colour1,3,"up",p1[0][0],p1[0][1],p1[0][2],p1[0][3],0) #creates the player1 snake object#

global p2

global colour2

global player2

player2=snake([[47,2],[47,1],[47,0]],20,20,colour2,3,"down",p2[0][0],p2[0][1],p2[0][2],p2[0][3],0) #creates the player2 snake object#

if numPlayers >= 3:

global p3

global colour3

global player3

player3=snake([[2,0],[1,0],[0,0]],20,20,colour3,3,"right",p3[0][0],p3[0][1],p3[0][2],p3[0][3],0) #creates the player3 snake object#

if numPlayers == 4:

global p4

global colour4

global player4

player4=snake([[45,25],[46,25],[47,25]],20,20,colour4,3,"left",p4[0][0],p4[0][1],p4[0][2],p4[0][3],0) #creates the player4 snake object#

def resetPlayers():

global p1

global colour1

global player1

player1=snake([[0,23],[0,24],[0,25]],20,20,colour1,3,"up",p1[0][0],p1[0][1],p1[0][2],p1[0][3],player1.score) #creates the player1 snake object#

global p2

global colour2

global player2

player2=snake([[47,2],[47,1],[47,0]],20,20,colour2,3,"down",p2[0][0],p2[0][1],p2[0][2],p2[0][3],player2.score) #creates the player2 snake object#

if numPlayers >= 3:

global p3

global colour3

global player3

player3=snake([[2,0],[1,0],[0,0]],20,20,colour3,3,"right",p3[0][0],p3[0][1],p3[0][2],p3[0][3],player3.score) #creates the player3 snake object#

if numPlayers == 4:

global p4

global colour4

global player4

player4=snake([[45,25],[46,25],[47,25]],20,20,colour4,3,"left",p4[0][0],p4[0][1],p4[0][2],p4[0][3],player4.score) #creates the player4 snake object#

multiPlayer()

def multiPlayer():

eaten=False

leave=False

global colour1

global colour2

global colour3

global colour4

timer=0

global winner

winner=0

Food=food(random.randint(0,47),random.randint(0,25),purple)

allPos=["-","-","-","-"] #creates an array with 4 dashes inside#

stop=button("Pause",38,myFont,white,850,523,130,50,black,purple,pause)

global alive

alive=0

p1score=newText("Player 1 Score: 0",19,myFont,colour1,0,523)

p2score=newText("Player 2 Score: 0",19,myFont,colour2,200,523)

if numPlayers >= 3: #if there are three or more players then#

p3score=newText("Player 3 Score: 0",19,myFont,colour3,400,523)

if numPlayers == 4: #if there are four players then#

p4score=newText("Player 4 Score: 0",19,myFont,colour4,600,523)

global p1wins

global p2wins

global p3wins

global p4wins

p1wins=newText("Player 1 Wins: "+str(player1.score),19,myFont,colour1,0,546)

p2wins=newText("Player 2 Wins: "+str(player2.score),19,myFont,colour2,200,546)

if numPlayers >= 3:

p3wins=newText("Player 3 Wins: "+str(player3.score),19,myFont,colour3,400,546)

if numPlayers == 4:

p4wins=newText("Player 4 Wins: "+str(player4.score),19,myFont,colour4,600,546)

global event

while leave==False:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

player1.changeDir(event)

player2.changeDir(event)

if numPlayers >= 3:

player3.changeDir(event)

if numPlayers == 4:

player4.changeDir(event)

if event.key ==pygame.K\_p:

paused=True

pause()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

for i in range(0,numPlayers):

for j in range(0,len(allPos[i])):

if allPos[i][0][0] == Food.x\_co and allPos[i][0][1] == Food.y\_co: #if any of the player heads are touching the food then#

if i == 0:

player1.grow()

if i == 1:

player2.grow()

if i == 2:

player3.grow()

if i == 3:

player4.grow()

eaten=True

break #stop the loop#

Food.createFood()

if eaten == True:

Food.spawn()

eaten = False

for i in range(0,numPlayers):

for j in range(0,len(allPos[i])):

if allPos[i][j][0] == Food.x\_co and allPos[i][j][1] == Food.y\_co: #if any of the snake’s bodies are touching the food then#

Food.spawn()

stop.checkClicked()

p1score.addText()

p1wins.addText()

p1score.text="Player 1 Score: "+str(player1.length-3) #update the player score#

p2score.addText()

p2wins.addText()

p2score.text="Player 2 Score: "+str(player2.length-3)

if numPlayers >= 3:

p3score.addText()

p3wins.addText()

p3score.text="Player 3 Score: "+str(player3.length-3)

if numPlayers == 4:

p4score.addText()

p4wins.addText()

p4score.text="Player 4 Score: "+str(player4.length-3)

if player1.dead == False: #if player 1 is alive then#

player1.makeSnake() #draw the snake#

allPos[0]=player1.coordinates #put the player coordinates in allPos#

else: #if not then#

player1.coordinates=[[100,100],[101,101]] #change the player coordinates#

allPos[0]=player1.coordinates

if player2.dead == False:

player2.makeSnake()

allPos[1]=player2.coordinates

else:

player2.coordinates=[[100,100],[101,101]]

allPos[1]=player2.coordinates

if numPlayers >= 3:

if player3.dead == False:

player3.makeSnake()

allPos[2]=player3.coordinates

else:

player3.coordinates=[[100,100],[101,101]]

allPos[2]=player3.coordinates

if numPlayers == 4:

if player4.dead == False:

player4.makeSnake()

allPos[3]=player4.coordinates

else:

player4.coordinates=[[100,100],[101,101]]

allPos[3]=player4.coordinates

pygame.draw.rect(view,white, [0,520,960,3]) #draw the line separating the HUD and the playable area#

alive=0

for i in range(0,numPlayers):

if allPos[i] != [[100,100],[101,101]]: #if the coordinates of the snake are not in the dead zone then#

alive=alive+1 #add one to alive#

winner=i #the winner is i#

if alive <= 1: #if there are no players left then#

multiPlayerEnd() #end multiplayer mode#

pygame.display.update() #update the pygame displayer#

speed=rate.tick()

timer=timer+speed

if timer > 160:

if player1.dead == False:

player1.movement()

if player2.dead == False:

player2.movement()

if numPlayers >= 3:

if player3.dead == False:

player3.movement()

if numPlayers == 4:

if player4.dead == False:

player4.movement()

timer=0

for i in range(0,numPlayers):

for j in range(1,len(allPos[i])):

if i == 0:

if allPos[0][0][0] == allPos[0][j][0] and allPos[0][0][1] == allPos[0][j][1]: #if player1’s head is touching is body then#

player1.dead = True #player 1 is dead#

if i == 1:

if allPos[1][0][0] == allPos[1][j][0] and allPos[1][0][1] == allPos[1][j][1]: #if player2’s head is touching is body then#

player2.dead = True #player 2 is dead#

if i == 2:

if allPos[2][0][0] == allPos[2][j][0] and allPos[2][0][1] == allPos[2][j][1]: #if player3’s head is touching is body then#

player3.dead = True #player 3 is dead#

if i == 3:

if allPos[3][0][0] == allPos[3][j][0] and allPos[3][0][1] == allPos[3][j][1]: #if player4’s head is touching is body then#

player4.dead = True #player 4 is dead#

for j in range(0,len(allPos[i])):

if allPos[0][0][0] == allPos[1][0][0] and allPos[0][0][1] == allPos[1][0][1]: #if player1 and player2 heads collide then#

player1.dead = True

player2.dead = True

if i == 0:

if allPos[0][j][0] == allPos[1][0][0] and allPos[0][j][1] == allPos[1][0][1] and player1.dead == False: #if player 2 hits player1’s body and player 1 is alive then#

player2.dead = True

for k in range(0,player2.length): #repeat the loop from 0 to the length of player2#

player1.grow() #add one unit to player1#

if i == 1:

if allPos[1][j][0] == allPos[0][0][0] and allPos[1][j][1] == allPos[0][0][1] and player2.dead == False: #if player1 hits player2 and player 2 is alive then#

player1.dead = True

for k in range(0,player1.length): #repeat the loop from 0 to the length of player2#

player2.grow() #add one unit to player2#

if numPlayers >= 3:

if allPos[0][0][0] == allPos[2][0][0] and allPos[0][0][1] == allPos[2][0][1]: #if player1 and player3 heads collide then#

player1.dead = True

player3.dead = True

if allPos[1][0][0] == allPos[2][0][0] and allPos[1][0][1] == allPos[2][0][1]: #if player2 and player3 heads collide then#

player2.dead = True

player3.dead = True

if i == 2:

if allPos[2][j][0] == allPos[0][0][0] and allPos[2][j][1] == allPos[0][0][1] and player3.dead == False: #if player1 hits player3 and player3 is alive then#

player1.dead = True

for k in range(0,player1.length):

player3.grow()

if allPos[2][j][0] == allPos[1][0][0] and allPos[2][j][1] == allPos[1][0][1] and player3.dead == False: #if player2 hits player3 and player3 is alive then#

player2.dead = True

for k in range(0,player2.length):

player3.grow()

if i == 0:

if allPos[0][j][0] == allPos[2][0][0] and allPos[0][j][1] == allPos[2][0][1] and player1.dead == False: #if player3 hits player1 and player1 is alive then#

player3.dead = True

for k in range(0,player3.length):

player1.grow()

if i == 1:

if allPos[1][j][0] == allPos[2][0][0] and allPos[1][j][1] == allPos[2][0][1] and player2.dead == False: #if player3 hits player2 and player2 is alive then#

player3.dead = True

for k in range(0,player3.length):

player2.grow()

if numPlayers == 4:

if allPos[0][0][0] == allPos[3][0][0] and allPos[0][0][1] == allPos[3][0][1]: #if player1 and player4 heads collide then#

player1.dead = True

player4.dead = True

if allPos[1][0][0] == allPos[3][0][0] and allPos[1][0][1] == allPos[3][0][1]: #if player2 and player4 heads collide then#

player2.dead = True

player4.dead = True

if allPos[2][0][0] == allPos[3][0][0] and allPos[2][0][1] == allPos[3][0][1]: #if player3 and player4 heads collide then#

player3.dead = True

player4.dead = True

if i == 3:

if allPos[3][j][0] == allPos[0][0][0] and allPos[3][j][1] == allPos[0][0][1] and player4.dead == False: #if player1 hits player4 and player4 is alive then#

player1.dead = True

for k in range(0,player1.length):

player4.grow()

if allPos[3][j][0] == allPos[1][0][0] and allPos[3][j][1] == allPos[1][0][1] and player4.dead == False: #if player2 hits player4 and player4 is alive then#

player2.dead = True

for k in range(0,player2.length):

player4.grow()

if allPos[3][j][0] == allPos[2][0][0] and allPos[3][j][1] == allPos[2][0][1] and player4.dead == False: #if player3 hits player4 and player4 is alive then#

player3.dead = True

for k in range(0,player3.length):

player4.grow()

if i == 0:

if allPos[0][j][0] == allPos[3][0][0] and allPos[0][j][1] == allPos[3][0][1] and player1.dead == False: #if player4 hits player1 and player1 is alive then#

player4.dead = True

for k in range(0,player4.length):

player1.grow()

if i == 1:

if allPos[1][j][0] == allPos[3][0][0] and allPos[1][j][1] == allPos[3][0][1] and player2.dead == False: #if player4 hits player2 and player2 is alive then#

player4.dead = True

for k in range(0,player4.length):

player2.grow()

if i == 2:

if allPos[2][j][0] == allPos[3][0][0] and allPos[2][j][1] == allPos[3][0][1] and player3.dead == False: #if player4 hits player3 and player3 is alive then#

player4.dead = True

for k in range(0,player4.length):

player3.grow()

for i in range(0,numPlayers):

if allPos[i][0][0] >= 48 or allPos[i][0][0] < 0: #if the snake is left or right to the playable area then#

if i == 0:

player1.dead = True

elif i == 1:

player2.dead = True

elif i == 2:

player3.dead = True

elif i == 3:

player4.dead = True

if allPos[i][0][1] >= 26 or allPos[i][0][1] < 0: #if the snake is above or below the playable area then#

if i == 0:

player1.dead = True

elif i == 1:

player2.dead = True

elif i == 2:

player3.dead = True

elif i == 3:

player4.dead = True

rate.tick(5000) #this sets the maximum frames per second to 5000#

def multiPlayerEnd():

global winner

global alive

over=True

tie=newText("It's a Tie",60,myFont,green,480,260)

win=newText("",60,myFont,green,480,260)

cont=button("-Play again",38 ,myFont,white,400,350,195,50,black,green,resetPlayers)

menu=button("-Main menu",38,myFont,white,400,400,190,50,black,blue,mainMenu)

end=button("-Quit",38,myFont,white,400,450,90,50,black,red,quitGame)

if alive == 1:

pygame.mixer.Sound.play(gameWin) #this plays a win sound#

if winner == 0:

win.text="Player 1 Wins"

player1.score=player1.score+1

p1wins.text="Player 1 Wins: "+str(player1.score)

if winner == 1:

win.text="Player 2 Wins"

player2.score=player2.score+1

p2wins.text="Player 2 Wins: "+str(player2.score)

if winner == 2:

win.text="Player 3 Wins"

player3.score=player3.score+1

p3wins.text="Player 3 Wins: "+str(player3.score)

if winner == 3:

win.text="Player 4 Wins"

player4.score=player4.score+1

p4wins.text="Player 4 Wins: "+str(player4.score)

else:

pygame.mixer.Sound.play(loss) #this plays a loss sound effect#

global event

while over==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

if alive == 1:

win.showJustText()

if alive == 0:

tie.showJustText()

cont.checkClicked()

menu.checkClicked()

end.checkClicked()

pygame.display.flip()

def highScoreBoard():

global afk

afk=True

global event

x=80

numScores=0

allScores=[[],[]] #this creates a 2d array called allScores#

c.execute("SELECT \* FROM highScores ORDER BY score DESC") #this selects the score and the username from the highScores table in descending order#

score=c.fetchall()

for i in range(0,len(score)): #this repeats a loop from 0 to the length of scores#

if score[i][0] not in allScores[0]: #if the score is not in allScores then#

allScores[0].append(score[i][0]) #append the score to allScores#

allScores[1].append(score[i][1]) #append the username to allScores#

finish=button("Back",35,myFont,white,0,0,90,45,black,blue,back)

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

if len(allScores[0]) > 19: #if the length of the scores is more than 19 then#

numScores=20

else:

numScores=len(allScores[0]) #numScores is the length of the scores#

for i in range(0,numScores): #repeat the loop from 0 to numScores#

if i > 9: #if i is greater than 9 then#

x=560 #x coordinate is 560#

y=i\*54+20-540 #the y coordinate is i\*54+20-540#

else:

x=100 #x coordinate is 100#

y=i\*54+20 #y coordinate is i\*54+20#

font=pygame.font.Font(myFont,35)

text=font.render((str(i+1)+". Score: "+str(allScores[1][i])+" by "+allScores[0][i]),False,green) #text is created with the number of the score and the username and the score itself#

view.blit(text,(x,y)) #this displays the text#

finish.checkClicked()

pygame.display.flip()

def option():

global afk

afk=True

global account

global colourz

c.execute("SELECT p1colour, p2colour, p3colour, p4colour FROM accounts WHERE username=?", #this selects the colours for player1,2,3 and 4 from the current players record in the accounts table#

(account,))

colourz=c.fetchall() #this adds all the play colours to colourz#

finish=button("Back",35,myFont,white,0,0,90,45,black,blue,back)

col=button("Colour",38,myFont,black,220,345,105,50,white,yellow,changeColour)

con=button("Controls",38,myFont,black,600,345,135,50,white,purple,changeControls)

heading=newText("What would you like to change?",60,myFont,green,480,150)

global event

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

finish.checkClicked()

col.checkClicked()

con.checkClicked()

heading.showJustText()

if afk==False:

break

pygame.display.flip()

def changeColour():

global account

global event

global colourz

finish=button("Back",38,myFont,white,0,520,100,50,black,green,option)

heading=newText("Which snake's colour would you like to change?",45,myFont,green,480,150)

p1col=button("Player 1",30,myFont,white,103,300,105,45,black,colours[colourz[0][0]],p1colu)

p2col=button("Player 2",30,myFont,white,311,300,115,45,black,colours[colourz[0][1]],p2colu)

p3col=button("Player 3",30,myFont,white,529,300,110,45,black,colours[colourz[0][2]],p3colu)

p4col=button("Player 4",30,myFont,white,742,300,115,45,black,colours[colourz[0][3]],p4colu)

over=True

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

#in this function the user is asked which player’s colour they would like to change#

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

p1col.checkClicked()

p2col.checkClicked()

p3col.checkClicked()

p4col.checkClicked()

finish.checkClicked()

heading.showJustText()

if afk==False:

break

pygame.display.flip()

def changeControls():

global account

global event

global colourz

finish=button("Back",38,myFont,white,0,520,100,50,black,green,option)

heading=newText("Which snake's controls would you like to change?",45,myFont,green,480,150)

p1con=button("Player 1",30,myFont,white,103,300,105,45,black,colours[colourz[0][0]],p1cont)

p2con=button("Player 2",30,myFont,white,311,300,115,45,black,colours[colourz[0][1]],p2cont)

p3con=button("Player 3",30,myFont,white,529,300,110,45,black,colours[colourz[0][2]],p3cont)

p4con=button("Player 4",30,myFont,white,742,300,115,45,black,colours[colourz[0][3]],p4cont)

while afk==True:

for event in pygame.event.get():

#in this function the user is asked which player’s control they would like to change#

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

finish.checkClicked()

p1con.checkClicked()

p2con.checkClicked()

p3con.checkClicked()

p4con.checkClicked()

heading.showJustText()

if afk==False:

break

pygame.display.flip()

def p1cont():

global player

player=1

pcontrols()

def p2cont():

global player

player=2

pcontrols()

def p3cont():

global player

player=3

pcontrols()

def p4cont():

global player

player=4

pcontrols()

def pcontrols():

global account

global colourz

global player

finish=button("Back",35,myFont,white,0,0,90,45,black,blue,changeControls)

menu=button("Main Menu",38,myFont,white,0,523,180,50,black,green,back)

chooseControl=newText("Click control you want to bind",45,myFont,green,480,70)

changeControl=newText("Press the button you want to use",45,myFont,green,480,70)

upBox=textBox("",50,None,black,355,200,250,35,Lgrey,white)

downBox=textBox("",50,None,black,355,270,250,35,Lgrey,white)

leftBox=textBox("",50,None,black,355,340,250,35,Lgrey,white)

rightBox=textBox("",50,None,black,355,410,250,35,Lgrey,white)

upCon=newText("Up:",30,myFont,colours[colourz[0][player-1]],305,200)

downCon=newText("Down:",30,myFont,colours[colourz[0][player-1]],275,270)

leftCon=newText("Left:",30,myFont,colours[colourz[0][player-1]],280,340)

rightCon=newText("Right:",30,myFont,colours[colourz[0][player-1]],270,410)

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

if event.type == pygame.KEYDOWN:

if upBox.place == True:

upBox.text=keyName(event.key)

upBox.click()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"up"+") = ? WHERE username=?", #updates the up control with the button that player selects#

(event.key,account,))

con.commit()

elif downBox.place == True:

downBox.text=keyName(event.key)

downBox.click()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"down"+") = ? WHERE username=?", #updates the down control with the button that player selects#

(event.key,account,))

con.commit()

elif leftBox.place == True:

leftBox.text=keyName(event.key)

leftBox.click()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"left"+") = ? WHERE username=?", #updates the left control with the button that player selects#

(event.key,account,))

con.commit()

elif rightBox.place == True:

rightBox.text=keyName(event.key)

rightBox.click()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"right"+") = ? WHERE username=?", #updates the right control with the button that player selects#

(event.key,account,))

con.commit()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

c.execute("SELECT (%s),(%s),(%s),(%s) FROM accounts WHERE username=?" #this selects the controls for the currently selected players.#

%("p"+str(player)+"up","p"+str(player)+"down","p"+str(player)+"left","p"+str(player)+"right"), #this adds in variables into the select statement so I can select specific record from variables#

(account,))

playerCont=c.fetchall()

upBox.text=keyName(playerCont[0][0])

downBox.text=keyName(playerCont[0][1])

leftBox.text=keyName(playerCont[0][2])

rightBox.text=keyName(playerCont[0][3])

view.fill(black)

finish.checkClicked()

menu.checkClicked()

if upBox.place == True or downBox.place == True or leftBox.place == True or rightBox.place == True: #if the player has clicked a textbox then#

changeControl.showJustText()

else:

chooseControl.showJustText()

upCon.addText()

downCon.addText()

leftCon.addText()

rightCon.addText()

upBox.click()

downBox.click()

leftBox.click()

rightBox.click()

if afk==False:

break

pygame.display.flip()

def p1colu():

global player

player=1

pcolour()

def p2colu():

#in these functions I was setting the player variable to the player number that they user picked#

global player

player=2

pcolour()

def p3colu():

global player

player=3

pcolour()

def p4colu():

global player

player=4

pcolour()

def pcolour():

global afk

global account

global colourz

global player

global currentCol

global blueb

global greenb

global yellowb

global pinkb

global purpleb

global orangeb

global whiteb

global menu

finish=button("Back",35,myFont,white,0,0,90,45,black,blue,changeColour)

chooseColour=newText("Pick a colour",45,myFont,white,480,50)

currentCol=button("Current Colour",35,myFont,black,0,80,225,45,colours[colourz[0][player-1]],colours[colourz[0][player-1]],None)

redb=button("",30,myFont,black,260,200,200,50,red,red,pred)

blueb=button("",30,myFont,black,500,320,200,50,blue,blue,pblue)

greenb=button("",30,myFont,black,260,380,200,50,green,green,pgreen)

yellowb=button("",30,myFont,black,260,320,200,50,yellow,yellow,pyellow)

pinkb=button("",30,myFont,black,500,200,200,50,pink,pink,ppink)

purpleb=button("",30,myFont,black,500,260,200,50,purple,purple,ppurple)

orangeb=button("",30,myFont,black,260,260,200,50,orange,orange,porange)

whiteb=button("",30,myFont,black,500,380,200,50,white,white,pwhite)

menu=button("Main Menu",38,myFont,white,0,523,180,50,black,green,back)

afk=True

while afk==True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

quitGame()

global mouse

global click

mouse=pygame.mouse.get\_pos()

click=pygame.mouse.get\_pressed()

view.fill(black)

c.execute("SELECT p1colour, p2colour, p3colour, p4colour FROM accounts WHERE username=?",#this selects all the player colours from the database from the current user’s record#

(account,))

colourz=c.fetchall()

currentCol.newColour=colours[colourz[0][player-1]]

currentCol.bColour=colours[colourz[0][player-1]]

chooseColour.showJustText()

finish.checkClicked()

currentCol.checkClicked()

redb.checkClicked()

blueb.checkClicked()

greenb.checkClicked()

yellowb.checkClicked()

pinkb.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

if afk==False:

break

pygame.display.flip()

def pred():

currentCol.newColour=red

currentCol.bColour=red

currentCol.checkClicked()

blueb.checkClicked()

greenb.checkClicked()

yellowb.checkClicked()

pinkb.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'red' WHERE username=?", #this updates the accounts table and changes the player colour to red#

(account,))

con.commit()

def pblue():

currentCol.newColour=blue

currentCol.bColour=blue

currentCol.checkClicked()

greenb.checkClicked()

yellowb.checkClicked()

pinkb.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'blue' WHERE username=?", #this updates the accounts table and changes the player colour to blue#

(account,))

con.commit()

def pgreen():

currentCol.newColour=green

currentCol.bColour=green

currentCol.checkClicked()

yellowb.checkClicked()

pinkb.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'green' WHERE username=?", #this updates the accounts table and changes the player colour to green#

(account,))

con.commit()

def pyellow():

currentCol.newColour=yellow

currentCol.bColour=yellow

currentCol.checkClicked()

pinkb.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'yellow' WHERE username=?", #this updates the accounts table and changes the player colour to yellow#

(account,))

con.commit()

def ppink():

currentCol.newColour=pink

currentCol.bColour=pink

currentCol.checkClicked()

purpleb.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'pink' WHERE username=?", #this updates the accounts table and changes the player colour to pink#

(account,))

con.commit()

def ppurple():

currentCol.newColour=purple

currentCol.bColour=purple

currentCol.checkClicked()

orangeb.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'purple' WHERE username=?", #this updates the accounts table and changes the player colour to purple#

(account,))

con.commit()

def porange():

currentCol.newColour=orange

currentCol.bColour=orange

currentCol.checkClicked()

whiteb.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'orange' WHERE username=?", #this updates the accounts table and changes the player colour to orange#

(account,))

con.commit()

def pwhite():

currentCol.newColour=white

currentCol.bColour=white

currentCol.checkClicked()

menu.checkClicked()

pygame.display.flip()

c.execute("UPDATE accounts SET ("+"p"+str(player)+"colour"+") = 'white' WHERE username=?", #this updates the accounts table and changes the player colour to white#

(account,))

con.commit()

intro()

## Evaluation

### Stake holder alpha testing

I had my stake holder go through my game and test my game trying to break it and testing the validation of the game. My stakeholder Venkata tested the multiplayer mode with me and we did not find any problems.

#### ACCOUNT SYSTEM

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Outcome** |
| Allow user to create an account on the log in screen | Click the register button | This was a very simple test that is used to test my buttons. When the register button is pressed it loads the “register account” page. |
|  | | |
| Be able to create an account | “Hello”- username  “splitter”- password  “splitter”- password | I created an account called Hello with the password splitter. When I pressed enter the account was successfully made and I was sent into the login screen to log into my account. |
|  | | |
| Be able to log into an account | “Hello”- username  “splitter”-password | I tried to log into my account with the username and password I had just created, and I used to log into my account. |
|  | | |
| Check that two accounts with the same name cannot be used | Create another account  “Hello”- username  “object”-password  “object”-password | To test that I cannot create an account with the same name I tried to create an account with the username Hello. When I tried |
|  | | |
| When creating a new account, the user should be able to type in the same password twice | “bob1234”- username  “brown”- password  “blue”- password | The test worked and when the passwords were not the same an error message showed that said that the passwords do not match. |
|  | | |
| When logging into an account all the characters need to be the same to allow entry into the account | “Hello”- username  “Splitter”- password | I tried to use a capital letter for the “s” in splitter to see if access would be allowed. It did not work and so all the characters need to be in the correct case for the user to be given access to that account |
|  | | |
| When logging into an account the username must be correct | “hello”- username  “splitter”- password | I tried to use a capital letter for the username to see if I could log in. I made the “H” in hello lower case and I was no able to log in as the username was not a perfect match. |
|  | | |
| Creating an account with a similar name | “hello”- username  “splitter”- password  “splitter”- password | I tried to make an account with a similar username and password to see if I would be denied. It worked I was able to create the account. |
|  | | |
| Check that all the controls are the same for that account | Change the up control to “5” then quit the game and log into the same account. Then log in and play the game and press 5 | To test that the user data is saved I changed the up control for player 1 to 5. Then I closed the game and opened it again I was able to move up using 5. |
|  | | |

#### MENU

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Outcome** |
| Check that the buttons for the main menu work | Click the button leading to each mode | To check that all the buttons worked I pressed each one to check that the buttons worked |
|  | | |
| Check that there are no hidden buttons | Click anywhere on the screen that isn’t a button | To make sure the buttons were confined to the space they should be I clicked randomly around the screen in places that didn’t have buttons. The test worked and I didn’t find any hidden buttons. |
| Check that single player mode button works | Click single player mode | When I pressed single player mode the controls were loaded |
|  | | |
| Check that the multiplayer mode button works | Click multiplayer | When multiplayer was clicked the user was given a choice of number of players |
|  | | |
| High score board shows | Click high score board | To check that high score board works I pressed the button and the highest scores was shown |
|  | | |
| Options show | Click options button | When I clicked the options button, I was given the option to either change the colour or the controls |
|  | | |
| Check that the controls are given to the player when they start playing | Click single player mode | When I pressed single player mode the controls were loaded |
|  | | |
| Multiplayer choice of players | Click single player mode and click on any number of players e.g 4 | When multiplayer was clicked the user was given a choice of number of players |
|  | | |
| Back button | Clicking the back button | The back button makes the user go back a screen when pressed |
|  | | |
| Quit button | Click the quit button | When the quit button is pressed he game is closed |
|  | | |

#### OPTIONS

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Outcome** |
| Binding the keys | Bind d to move up for player two | In options I change the control for up to “d”. To check that it worked I went to multiplayer mode and the control for player two became “d”. when I pressed in game the snake moved right and up |
|  | | |
| Binding keys | Bind q to move up for player one | I changes the up control for player one to q and in game when I press q player one will move up |
|  | | |
| Binding keys | Bind 9 to move up for player one | To make sure that all keys can be changed the up control to 9 and when I pressed 9 in game the snake went up |
|  | | |
| Sound effects | Click the SFX button | I decided that there was no need to add a sound effects button because the player can turn down the volume in the volume mixer |
| Change colours | Change player 4’s colour to white | The test worked and when I clicked the white button the snakes colour changed to white. |

#### GAME STATES

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Outcome** |
| HUD | Start single player mode | When single player mode starts at the bottom of the screen the score, highest score and pause button shows |
|  | | |
| Starting to move in single player mode | Pressing a random keys e.g l and up arrow | When I pressed “I” nothing happened but when I pressed up the snake moved up. Originally, I decided that the player would start by pressing a directional button, but I changed the way to start by pressing next or space. Then after I had started the single player the snake would automatically move right. |
|  | | |
| Moving in single player mode | Down arrow | When the down arrow is pressed the snake starts to move down |
|  | | |
| Checking control for movement | Pressing left arrow when facing right | It should be impossible to do a 180 turn as the snake would go through itself and when left arrow is pressed while facing right nothing happened. |
|  | | |
| Checking quick response | Pressing right arrow then left arrow after one movement | When pressing two buttons very quickly the first input needs to be used first before the second input is used. This means the snake moves right and not left. |
|  | | |
| Checking for square bug when snake is four units long | Repeatedly pressing up arrow then left arrow then down arrow then right arrow | When the snake is four units long it should be impossible for the snake to eat itself. To make sure this is right I made the snake go around in a circle to see if it would die. It did not die so the test was a success. |
|  | | |
| Changing direction in game | Up arrow, left arrow, down arrow, right arrow | To make sure all the controls work I need to press all four of them to make sure they work which they did. |
|  | | |
| Starting in multiplayer mode | Up arrow | To test that multiplayer worked I pressed up arrow which moved the snake up. |
|  | | |
| Starting in multiplayer mode | r | “r” is not binded to anything so when pressed the snakes should not change direction. |
| Eating food | Snake’s head collide with food | When the snake ran into the food, the food was eaten and the snake got longer. |
|  | | |
| Snake runs into wall | Have the snake keep going forward | When the snake ran into the wall it died and the game over screen was shown. |
|  | | |
| Snake dying | Have the snake go across the wall and turn when it meets the corner |  |
|  | | |
| One snake left in multiplayer mode | Let all the snakes apart from one run into a wall | When one snake is left alive then the game ends and the last player alive wins the game. |
|  | | |
| Last snakes alive die simultaneously | Have two snakes run into each other head on | When the last two snake run head first into each other they both die at the same time and the game ends in a tie. |
|  | | |
| Eliminate another player | Have one snake run into the body of another snake | When a snake runs into another snake the first snake will die and the second snake will gain the length of the first snake. Originally, I didn’t plan for this but with feedback from my stakeholders I decided that it would be good for snakes to gain the length of other snakes to give the snake more of an incentive compete. |
|  | | |
| Easy mode | Get less than 5 points more than three times in a row | When I was making the game, I decided that players who are getting less than a 20 score should be allowed to try easy mode. When the player gets less than 20 points three times in a row they will be asked if they want to play on easy mode. |
|  | | |
| Hard mode | Get more than 20 points more than three times in a row | I decided that it would be too easy to get more than twenty points, so I changed the number of points to 60. If the player gets more than 60 points three times in a row they will be asked to play on hard mode. |
|  | | |
| Testing the pause function | ESC or the pause button | When the player presses the pause button or the ESC then the pause menu comes up. |
|  | | |
| Quitting snake while in a round | Pause the game then press quit | The quit button worked, and the game was closed when pressed. |
|  | | |

#### SOUND TESTING

|  |  |  |
| --- | --- | --- |
| **Test** | **Input** | **Outcome** |
| Check that the eat sound works | Eat the food in the game | When the food is eaten a crunching noise is made. |
| Check that the death sound | Have the snake run into the wall | I decided that there would be no need for a death noise as usually the game would end when a snake would die. Instead I added game over noises that happen when the game is finished. |
| Check that the high score noise works | Beat the high score | When the high score is beaten a congratulations-noise is made. |

### Comparing with success criteria

#### ESSENTIAL

Three different options – I decided that it would be essential that I would have at least three different buttons that the user could choose in the main menu. At the end of my game there were four different buttons on the main menu because I added an options button as well that allows the user to customise the game. I met this requirement and I exceeded my expectations of it. (Screenshots on page 148-151)

Classic single player mode – when the player selects single player first the controls are given to the player. When the player is ready, they can either press space or press the next button to start the actual game. Underneath the playable area there is the current score and the highest score anyone has achieved. I did not add the highest score for that session at the bottom because I didn’t think that it would be needed, and it would make the HUD more cramped. The default controls are the arrow keys for movement and if any changes are made then it will appear on the controls screen. I did not let the user choose how hard they wanted the game to be because it would make the game more interactive if they player could earn hard mode or be given easy mode if they are struggling. Also, I decided that instead of the size of one unit getting bigger or smaller in easy mode the snake moves slower and in hard mode the snake moves faster. It does not make sense for easy mode to have a smaller size because although it makes the snake closer and food closer together it can also make the game harder as well, this is because when the snake gets larger it will get in its way more, so it becomes harder to pass yourself to get to the food. If the size of one unit was smaller then it becomes easier to pass yourself to get to the food to the likelihood of collisions decrease. The difficulty of the game depends on how well the player performs which I will explain later in this section. Single player was a success and all the requirements were met and I improved on the difficulty system as well. (Screenshot on page 154 - 156)

Multiplayer mode – when the player picks multiplayer mode, they are asked to select how many players there are (either 2, 3 or 4). The controls are given for each player once the user has selected the number of players. When the user presses space to continue then the snakes will load and depending on how many there are the score of each snake appears at the bottom with the number of rounds won. The round ends when one player is left or when there are no players left. If there is one player left, then that player is wins a round and is awarded a round point. If there are no players left from both of the players simultaneously dying, then a tie is given, and no points are given. When the snakes are first created, they are all three units long and have a score of 0. I put the snakes in each corner with each snake facing the snake in a loop. Multiplayer mode was completed in the way I envisioned it in my success criteria making it a success. (Screenshot on page 157 - 160)

High score board – the high score board shows the top twenty scores from the players and each player can only be in the top twenty once as their highest score is shown. As soon as a player gets a high score then it will be updated to the player can see it straight away. I did not show the time it took the user to do it because I decided there would be no point to showing the time as it did not change the score system and it would take up space on the HUD of single player mode and the high score board. I would say that this was successful as I met almost all the point and I reduced clutter in the high score board as each player can only enter the high score board once. (Screenshot on page 150)

Pausing – I added a pause feature to singlePlayer mode and multiplayer mode so that when the player presses p or clicks the pause button then the pause menu comes up. Inside the pause menu you can resume or quit the game or return to the main menu. This point was met successfully. (Screenshot on page 162)

The player can see a top down simple view of the snake. (Screenshot on page 154-160)

Sound effects – when the snake eats the food and when the game ends. In single player mode a game over noise is heard and in multiplayer when there is a winner then a celebratory noise is made. A different type of celebratory noise is made when the player gets a new high score. I added clicking noises for my game whenever a button is clicked so the user gets more feedback from the game. (Testing on page 162-163)

Account system – when the game starts the user is given a choice to log in to an account or to create an account. This allows users to customise their controls and colours and have them saved to that account. I used a hashing and salting algorithm to save the password in the database so that it cannot be read and used by anyone. The account system has been used successfully and I used it to help implement the options function. (Screenshot on page 143-147)

Database – the database holds the controls and the colours that have been changed for that account and it holds the high scores. I write to it and read from it when logging in and registering so that I can read hashed passwords and create new account. I select from the database to see if any usernames are taken. The database was successful, and I can view all the data inside it using third party software. (Use of database on page 143-147)

Difficulty – to make the game easier or harder if the player gets below 20 three times in a row then they will be asked if they want to play on easy mode which will have a slower speed. A slower speed of the snake would help player as it would give them time to make decisions for where they want to move before they are forced too. I made this an option instead of a compulsion as it could make the game less fun for some players as they would have to play the game very slowly. If the player gets over 60 three times in a row, then they will be asked if they would like to player on hard mode where the snake moves faster. Making the snake moves faster makes it harder for the player to think about what to do next so it makes the game harder. I improved upon the success criteria in the analysis as I made a better system for changing the difficulty in the game. (Screenshot on page 161)

#### OPTIONAL

Practise mode – I did not add a practise mode to my game because it would take lots of memory to store the details of where the snake moves in the game. I would also need to make the screen of the game bigger so that it would be able to fit two screens of the same game. Changing the size of the screen would be difficult as I would need to change pixel positions for many different items such as pause button HUD etc. I decided that I wouldn’t make a practise mode because not many people would play it and it would be a waste of memory space for them.

Options – I did add customizability to controls so that the player can have their own key binds. This point was successful. (Screenshot on page 152-153)

I also made it possible for the players to change the colour of the snake, but I did not make it possible for the user to change the background because it would make some buttons stand out and could merge different coloured items together. (Screenshot on page 154)

I did not make the snake bigger or smaller to change the difficulty as I explained in the essential section.

#### UNNECESSARY

Graphics – I kept very basic graphics to my game because it would be extremely difficult to change the graphics of my snake from 2d to 2.5d. It would also be harder to see the exact position the snake is in which would make it harder for user as they won’t be able to see how close the snake is to the food. This point was not completed.

Multiplayer custom mode – I did not add a new custom mode because of the time restraint that I had I wouldn’t be able to finish it on time.

More players in multiplayer mode – I didn’t add more players because it would be extremely difficult to add more than four people on the same keyboard around a computer.

Online multiplayer mode - I was not able to create an online multiplayer mode as it would take very long to complete, and I did not have the time space to do it.

Computer AI – I would have struggled to create a computer AI for multiplayer snake due to the amount of time that I had to complete it.

### Evaluation on usability features

User interface – the user interface I have created is very simple and intuitive, when the mouse hovers over a button the button will change colour so that the user knows that it is interactive. I used buttons for the navigation of my game because it is intuitive and quick to use.

Log in and registration – when logging in or registering text boxes are used for the user to type. I made the text boxes easy to understand as well by making them grey and then making them white when pressed. The user will know which user they are typing in because it will be the one that is lit up. If the user makes an invalid input, then red text will show up telling the user what went wrong so that they can try again.

High score – the high scores are shown as a table with two columns showing the top twenty score from top to bottom, highest to lowest.

Options – when changing player colours the user picks the player they want to change and then a screen of colours is shown. When the user clicks a colour the player that they picked will have their colour changed to their desired colour. The method of changing snake colours are buttons that will change the colour of the snake. I decided that this was the best method of changing snake colours as it is intuitive, and no validation is needed. When changing controls all the user needs to do is select the player and the direction they want to change and then click any button on the keyboard. When the player clicks the button, the control will change to the button that was pressed. This is much simpler than typing in the button you want to change it to as press the button they want to use.

In game – when the player picks simple player mode the controls are given to the player, the player controls when the game will start by pressing space or the next button, this gives time to the player to check their controls and feel comfortable in the game before playing it. I created a pause feature that helps the user a lot by taking a break in the game or return to the main menu.

#### HOW MY FEATURES HELP THE USER

The options menu helps the user a lot because the player has the ability to change the controls to what they feel comfortable with. To help with colour-blindness I added the ability for the users to change the snake colour so that they can see colours that they can recognise. The game has a black background which makes items stand out more and make it easier to see for the visually impaired. The simplistic graphics also helps the visually impaired as they can distinguish between things easily.

The user interface I have created is very easy to understand and I have given instructions for most parts of my game and I made the rest of the game intuitive. The user interface is filled with buttons and the user does not need any complex inputs to access the game completely, this restricts the need for validation.

In game I stopped the snake from going outside the size of the screen, to do this if the snake tries to go past the border then it will die. This is a form of validation that I used to stop the snake from going places it shouldn’t be able to.

### What changed from the design?

In the design I said that I would use a 2d list that sored the coordinates of all the items in the game, this list would create a grid with the snakes and the food, and I would use this grid to check the snakes are colliding or going outside the playable area. I decided that I would make a 2d list for each snake instead of one big list because the big list will have lots of unused spaces which is a waste of memory, it would also make it harder for me to detect collisions with the snake and the boundary as it is very common for list range errors to occur.

In my design I wrote that I would create a death function or method when the snake is going to die. I didn’t need to do this for single player mode as the game ends when one snake dies. For multiplayer mode I didn’t create a death function or method because the death code wouldn’t work if one of the snake objects stopped existing. Instead of using a function or method I used the alive attribute to check whether or not to displayer the snake. If the snake was not alive then I would stop drawing it and I would change the coordinates to somewhere outside the playable area.

In my design I said that I would use a load high score function and a new high score function, but I found that the sql statements for updating and selecting data from the database are extremely short, so I didn’t need to create new functions just to load the high scores.

I wrote that I was going to hash the passwords and save them and then check that they are correct by hashing the user entered password and checking it against the saved password. I realised that simple hashing isn’t secure and common passwords can be easily cracked as they will all be the same inside a database. I used hashing and salting which hashes every password the same way and then using a randomly generated string (called a salt) I encoded the already hashed password and then attached the salt to the hashed and salted string. To check that the password is correct I made a function that takes the saved encoded password and the entered password, the entered password is hashed and then the salt saved at the end of the encoded password is used to encode the entered password, if the passwords are the same then they will have gone through the same process making it the same string. Hashing and salting is more secure than just hashing as in the database none of the passwords are the same as they were all hashed with different strings.

I said that I would use a drop-down menu to change the colours in my game, but it was easier to use buttons instead because each of them can run a function that will change the data saved in the database as well as on the screen.

### What is there to improve?

I could improve my game by adding a multiplayer mode that will have custom power ups such as 5 units longer or faster snake or teleport etc. for some people normal multiplayer mode could get boring and having powers could add a twist to the game making people want to play it more.

I could try to make a light themed version of the game where the background is white, and the text becomes black. This would attract more players as they have a greater customizability of the game.

I could try and let the user be able control the snake just with left and right click on the mouse, this would make the entire game controllable just with the mouse which gives better accessibility for people who are playing with one hand.

To improve my game, I could add an AI to play in multiplayer mode if the user has no one to play with but they want to play multiplayer mode. Adding a competent AI to the game would be extremely difficult when there is an obstacle such as another player.

If it were to continue with this project with more time spent on it, I would try to make a snake “battle royale” game mode where 100 players play multiplayer snake online and the last snake to survive wins. This would take about double the time I spent on the project I finished, and I would need a dedicated server to run my game.

### How can I maintain my code?

At the moment there are still bugs in my game that I did not find a solution for in my time developing the game.

When changing the colours or controls for a player there was a bit of delay between changing it and it being displayed on screen. I tried to remedy this in the development by updating the current colour before I updated the database because updating the database was causing the delay. This still caused problems because after the colour changed there is a moment where the game freezes and you cannot interact with it if you are too fast. To maintain my code overtime, I would try to find a way to instantly change the colour or find a way to tell the user that they cannot do anything for the time the button do not work.

Sometimes in multiplayer mode if you don’t press a button for long enough then it won’t be registered, and the snake won’t move in the direction that was pressed. Ultimately python and pygame poses lots of limitations for creating complex games and it is not so good at handling many inputs at once. If I were to work on this long term, I would maintain the code by switching to another language such as C#, this would also help me if I were to push the limits of my snake game further. Another problem that python has is that sometimes the game can feel slightly laggy in multiplayer mode due to the amount of code inside the function, switching to C# would also help me to get more consistent frames per second.

Another bug is that if the user changed all the controls to button with long name e.g. arrow keys, then when the controls are shown for multiplayer mode the text will overlap and it would be difficult to read it.

I would like to fix the typing mechanism in my registration function because when backspace is held only one character is removed until the button is pressed again. There are many things different from the conventional typing method and the typing method I created, if I were to switch to a different language then there should be an inbuilt typing function in the game which would be easier and simpler to use as I wouldn’t need to make it from scratch.

To maintain my database, I should start to remove scores that are below the top twenty. This is because at the moment all scores are saved in the database but only the top twenty scores are shown. Even though the scores do not take up much space it would be best if scores that have no use should be deleted from the database to reduce its memory space.

### Overview of the project

Most of the project met my expectations I made in the analysis, I was able to create a multiplayer snake game with a log in system and the ability for the user to change the controls to whatever they want. I have met my own requirements to an extent that I am happy with although the biggest let down is the inconsistency in the controls, this was a result of me choosing python as my language to code in which has significantly impacted the limits of my game. if I were to do the project again, I would use another more widespread language such as C# or visual basic to code my game. Apart from the lack of smoothness in the controls I would like to create online capabilities so that users can play with each other online with having to be together. Unfortunately, my ambitions were not realistic and with the time I had I am happy with what I have achieved.