The Plan

My plan is to advance my programming skills by creating a variety of old school games, for example, snake, pong, etc. I will then link them together with a game launcher that will monitor and centralise the games and the scores. This project will include making graphical programs and using file manipulation in python.

Snake Game Part 1

Today I began creating a simple snake game using Python. I used the turtle library to do the graphics aspect of the game and individual subroutines that can be called up every time the player makes a move in the game. This is the first step to creating the basic game.

Snake Game Part 2

The first stage of this project is creating the screen. I set the screen size to 550px X 550px. This is just big enough for the game to play in and will be compatible with all PCs. The background is also set to white to allow you to easily see where the snake is and the food. This code is shown in figure 1.

Snake Game Part 3

The next step is to create the sprites and configure the movement for the snake.

Snake Game Part 4

This step involves coding and debugging the collision detection aspect of the game. This game uses a collision algorithm to detect three different collision types.

1. If the snake collides with the food

2. If the snake collides with the border

3. If the snake collides with itself

Figure 3 shows the algorithms used to detect what collides with what.

Snake Game Part 5

The final step is coding the game over screen and debugging the game before final deployment. The game over screen is shown when the snake collides with anything other than the food on the screen. The game over screen displays your score (which is determined by how many pieces of food you have collected), "Game Over" text and next instructions on how to exit the game. When you click anywhere on the game it exists and returned to the main menu program.

Figure 4 shows the game over screen code

Menu System Part 1

Figure 5 outlines the beginning of the Menu System program. To start with I made it so you can choose between "Games", "Scores" or quitting the application altogether. There is also validation in the IF function in the form of an else statement at the bottom which restarts the subroutine if you don't input a valid answer

Menu System Part 2

Figure 6 shows a deeper view of the "Games" tab, this allows you to select one of the five games provided from the launcher. This also comes with validation and restarts the subroutine if invalid input is detected. This directly launches the games from this IF function.

Menu System Part 3

Figure 7 shows the scoring system for the whole project, it begins asking whether you want to view the scores or play the games, this is so the program can determine whether the program should open the file in a readable format or in an appending format. It also asks for your name so it can keep track of who is creating the scores.

Space Invaders Part 1

Figure 8 shows the beginning of the program for the game. This code sets the screen up, imports the images from file explorer so you can have a picture of an enemy and the player and draws the border for the sprites to move in.

Space Invaders Part 2

Figure 9 is the scoring part of the program, this fetches the score from the main while loop in the code and then it formats it in the font Arial, then displays it every time the screen is refreshed.

Space Invaders Part 3

Figure 10 shows the creation of the player, it sets the position and the movement speed, it also overlays that previously imported image. This snippet of code also designs and sets the enemies up. The number of enemies is configurable by changing the variable "number\_of\_enemies" and again the other imported image overlays the enemy sprites as well.

Space Invaders Part 4

Figure 11 is about how the player destroys the enemies and how he moves. Like with the player and enemies the bullets need to be set up in the program, this is done by setting the speed and position of the bullet, etc. Each movement is in a separate subroutine so it can be called up at any point in the program.

Space Invaders Part 5

Figure 12 shows the main game loop which is situated in a while loop, this allows the game to constantly check for movements and update the frame multiple times per second. This section of the loop handles the enemy’s movements.

Space Invaders Part 6

Figure 13 shows the collision detection aspect of the program, this runs in the main loop so it is constantly checking if an object has collided. It works on a coordinates principle where it looks to see if an object has overlapped certain coordinates. If it has then it resets the position of the bullet and hides the enemy object it hit. It also adds to the score which at the end of the game adds to the global score text file.

Tetris Part 1

Figure 14 shows the code that sets up the scene for the game. I'm using the Tkinter library instead of the Turtle library to complete this game because Turtle does not have the ability to do this complex of a game. It begins by drawing the canvas and then setting up the various different types of Tetris block.

Tetris Part 2

This snippet of code (Figure 15) shows the mechanics behind how the Tetris blocks rotate. It is set in a subroutine so it can be called up by the main game loop at any time during the game, this is triggered by the "W" key.

Tetris Part 3

Figure 16 shows the prediction part of the game. This game predicts where you are going to drop the block and what orientation you are going to do it in. It does this by creating an almost ghost-like representation of the block as it travels down the screen. Again, each of these functions are in subroutines.

Tetris Part 4

This subroutine (Figure 17) shows the play again button that can be pressed when the game is over. This is another benefit to using the Tkinter library, as you wouldn't be able to use buttons in Turtle. This function restarts the program and reloads all of the assets.

Tetris Part 5

Figure 18 shows the function of the other button which is the quit, this allows the player to quit the game and return back to the game launcher when they have finished the game. It also uploads their score to the score database under the corresponding name.

Pong Part 1

This is the first part of the Tetris code development process. As shown in Figure 19, I began to set up the screen for the play area and I did this by using the Turtle module that is embedded within the python programming language. I then began setting up the left and right paddles that are controlled by the players. I then designed the ball, again using the turtle module.

Pong Part 2

The next thing to do was create the scoring display, again I used the turtle module which allows me to update the screen every frame, this can be seen in figure 20. The next step was to create the movement functions for the player's paddles. I did this by moving the paddles a set distance every time they pressed the corresponding key.

Pong Part 3

The last part was to finalise the functionality of the game and to add in the menu system and score integration, the following code can be found in figure 21. I began to create the movement for the ball which only required two lines of code. I then used collision detection to find out if the ball hit the wall and what would happen, if the ball hits the walls to the left or right then the points will be given to the corresponding player. If the ball hits the top or bottom of the play area then it will be bounced off in the opposite y-axis of the ball’s original trajectory. I then completed collision detection between the paddles and ball, this was different to the walls because when the ball hit the paddles it would bounce off in the opposite x-axis to the ball’s original trajectory. The last step was to include the function of scoring and exiting the game back to the menu system which was achieved by using a lambda function.

Hangman Part 1

Like with all the other games the first stage is setting up the area the player will see and/or interact with. This game uses the turtle function and the command line. The reference to the code can be found in figure 22. At this stage, I not only created the window that would show the hangman but also the basic first stage of the image.

Hangman Part 2

The next part is the main function of the program that checks to see if the letters picked by the player are in the word that was randomly picked from an inbuilt array. This can be found in figure 23. The entire function is nested inside a while loop which only stops if you run out of guesses, the maximum number of incorrect guesses is 6. You also have 4 categories to pick from each of which has its own array of words. This part of the game is all based around the command-line interface.

Hangman Part 3

The next stage is adding the body parts onto the hangman after a letter has been guessed incorrect, this can be seen in figure 24. Every time a player guesses the wrong letter it increments a variable by 1 each time and then is filtered through a group of "IF" statements that add on the corresponding body part in order, this is what makes this part of the game visual.

Hangman Part 4

The last part of the game (figure 25) is part of the menu system and how the game interacts with the scoring file that is shared between all of the games. When the player has run out of guesses the program asks them if they want to play again, if they say yes then it restarts the game, if they say no then it will save their current progress to the scoring file and reopen the menu system program.

Conclusion

Completing this project has advanced my knowledge of the Python programming language dramatically and has greatly increased my knowledge of computer graphics and programming in general. This has been the largest programming project that I have completed so far and it improved my skills working with large programming projects across multiple scripts.