**code breakdown**

**1 – 19**

basic template for initializing a MATLAB GUI application:  
The code defines a function named snake\_g that can take a variable number of input arguments and may return a variable number of output arguments.  
It sets up some variables related to GUI initialization.   
It creates a structure named gui\_State with fields that store various properties related to the GUI, such as the GUI's name, whether it should be a singleton (only one instance allowed), and callback functions.   
The code checks if any input arguments were passed to the snake\_g function.  
If so, it assumes that a callback function is specified and stores its function handle in the gui\_Callback field of gui\_State.   
It then handles the output arguments by calling the gui\_mainfcn function with the appropriate input arguments.

**22 – 48**

The function is a callback function that is executed just before a MATLAB GUI named snake\_g is made visible to the user.  
The function sets the handles.output field to the handle of the GUI figure (hObject). This field is used to manage the GUI's data and components.   
The handles structure is updated with the new handles, output value using guidata(hObject, handles).   
This ensures that the modified handles structure is accessible in other callback functions.   
The function sets up the first axes (axes1) in the GUI and turns off axis ticks and labels to prepare for displaying graphical elements.   
Two global variables highestScore and score\_keeper are declared, allowing their values to be shared across different functions in the MATLAB workspace.  
The text displayed in the GUI element with the tag highest\_score is updated to show the value of highestScore, the reason for using num2str is that GUI elements in MATLAB can typically only display text (strings).   
If you want to display a numeric value in a GUI text element (like a label or textbox), you need to convert the numeric value to a string representation.

**50 – 59**

The purpose of the function is Returning the default output value of the GUI application through the varargout cell array.  
Triggering the start\_game\_Callback function, which is likely responsible for starting the game functionality in the application.

Up press button:

direction is used to control the movement of the snake, in response to user input.

This code snippet sets the direction variable to 2 (representing "up") only if the current direction is not "down" (direction is not equal to 4).   
If the current direction is "down" (direction is 4), the code inside the if block is not executed, and the value of direction remains unchanged.

**108 - 131**

Initializes and sets up several global variables (mata, matb, matc, direction, points, highestScore, and score\_keeper) used throughout the application.   
Updates two GUI elements (highest\_score and score) with the initial values of highestScore and points.   
Sets initial coordinates for a game object (snake) in the game board using arrays locx and locy.   
Initializes three matrices (mata, matb, and matc) to represent the game board state. Calls the function update\_snake to update the graphical representation of the snake on the game board based on the initial coordinates.

**133 – 164**

Creates boundaries from right to left up and down in yellow.  
In this code we generate a random point within a 3D matrix (mata), ensuring that the point does not already exist in another set of coordinates (locx and locy).   
Once a unique random point is found, the code marks the point with white color   
(RGB value 255) in three separate matrices (mata, matb, and matc).  
Finally, it displays the updated matrices as a color image using the imshow function.  
Imshow function combines the three matrices (mata, matb, and matc) into a single color image and displays it on the screen using the imshow function.  
The RGB values in each channel determine the color of each pixel in the displayed image.

**165 – 207**

First, we clear the previous position of the snake from the image, making way for updating the image with the snake's new position i.e. removing the visual representation of the previous positions of the snake's body.  
In the first if we handle the logic when the snake's head encounters a randomly generated point (food).   
If the snake eats the food, it grows longer, and a new random point is generated for the next food.   
If the snake does not eat the food, its tail is moved forward, maintaining its length as it moves.   
In code wise we copies the elements from the first element (locx(1)) to the last element (locx(len)) of the locx array to the elements starting from the second position (locx(2)) to the last position (locx(len+1)) in the array.   
It effectively shifts all the elements one position to the right, making room for the new position of the snake's head.   
If the condition is happening it generates a new random point (apple) to be placed on the game board, making sure the point is not already part of the snake's body.   
It marks the new apple's position with white color in the matrices mata, matb, and matc.   
It increments the player's score (points) and updates the highest score if necessary.   
If the snake did not eat the apple:  
It updates the arrays locx and locy to move the snake's body one step forward by shifting all the elements one position to the left.   
The game continues indefinitely within the infinite loop, constantly updating the game board and responding to the player's interactions with the snake.

**208 – 265**

In this part the code snippet handles the scenario when the snake is moving to the right/left/down/up, and it checks if the snake has hit the boundary of the game board.  
If the boundary is hit, it displays a "Game Over" message and stops the game.   
otherwise, it moves the snake one step to the right/left/down/up.  
In the second part of the code we check if the snake's head collides with any part of its body.   
If a collision is detected, it ends the game and displays a "Game Over" message. Otherwise, it updates the snake's position and continues the game by calling the update\_snake function with the current locx and locy arrays.

**266 – 277**

It this part of the code in order to avoid endless display of the figure we use try and catch function this code ensures that the program keeps running while the specific window is open.   
If the window is closed by the user, the program gracefully stops the loop and closes the window (if it was still open) after the loop ends.

**308 – 339**

This code is an event handler that responds to key presses in a MATLAB figure.   
It updates the global variable direction based on the pressed arrow keys (up, down, left, right) to control the movement direction of an object, such as a snake or character, within the game.   
If other keys are pressed, the direction variable remains unchanged.

**Notes**

When you press the start game button, in order to control the snake with the keyboard, you must click with the mouse on the game display immediately after pressing the start game button.   
This is due to the fact that when we click on start the game, the focus of the game focuses on the button that is pressed and not on the keyboard, so in order to return the focus to the keyboard, you have to click with the mouse on the game display.