```
import random
from tkinter import *
from tkinter import messagebox
from PIL import Image
from playsound import playsound
```

The `random` module is imported for generating random numbers.

The `tkinter` module is imported, along with `messagebox`, for creating the game's GUI.

The `PIL` module is imported to handle image processing.

The `playsound` module is imported to play sound effects during the game.

```
score = 0

run=True
```

The `score` variable is initialized to keep track of the player's score.

The `run` variable is set to `True` to control the main game loop.

```
while run:
    window = Tk()
    window.geometry('905x700')
    window.title('HANGMAN')
    bg = PhotoImage(file="forest_BG.png")
    window.config(bg = '#96AE31')
```

The main game loop begins with a `while` loop that executes as long as `run` is `True`.

Inside the loop, a new `Tk` window is created for the game.

The window size is set to 905x700 pixels, and the title is set to "HANGMAN".

A background image, loaded from "forest_BG.png", is set as the background of the window.

The window's background color is set to '#96AE31'.

```
label = Label(window, image=bg)
label.place(x=0,y=0)
```

A `Label` widget is created to display the background image.

The `place` method is used to position the label at coordinates (0, 0) within the window.

```
lose_count = 0
win_count = 0
```

The `lose_count` and `win_count` variables are initialized to keep track of the number of incorrect guesses and correct guesses, respectively.

```
with open('animals.txt','r') as file:
    l = file.readlines()
index = random.randint(0,len(l)-1)
selected_word = l[index].strip('\n')
```

The code opens the "animals.txt" file, which contains a list of animal words, in read mode.

The contents of the file are read into a list called `l`.

A random index is generated using `random.randint()` to select a word from the list.

The selected word is stored in the `selected_word` variable after removing the newline character.

```
x = 200
for i in range(0,len(selected_word)):

exec('d{}=Label(window,text="_",bg="#96AE31",font=("arial",40))'.format(i))
    exec('d{}.place(x={},y={})'.format(i,x,430))
    x += 60
```

The code creates a series of `Label` widgets, one for each letter in the selected word, to display the word's letters as dashes initially.

The `exec()` function is used to dynamically generate variables `d0`, `d1`, `d2`, and so on, up to `dN`, where N is the length of the selected word.

Each label is positioned horizontally using the `place()` method, with the `x` coordinate incremented by 60 pixels for each label.

```
alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
for letter in alphabet:

exec('{}=PhotoImage(file="{}.png")'.format(letter,letter))
```

The `alphabet` list contains all the lowercase letters of the English alphabet.

The code uses the `exec()` function to create a series of image objects, one for each letter, by dynamically generating variable names `a`, `b`, `c`, and so on, up to `z`.

The image objects are created using the `PhotoImage()` constructor and loaded from image files named after each letter, such as "a.png", "b.png", etc.

```
hmi = ['h1','h2','h3','h4','h5','h6','h7']
for hangman in hmi:
exec('{}=PhotoImage(file="{}.png")'.format(hangman,hangman))
```

The `hmi` list contains the names of the hangman image files, such as "h1.png", "h2.png", etc.

Similar to the previous loop, the code creates image objects for each hangman image by dynamically generating variable names `h1`, `h2`, and so on, up to `h7`.

The images are loaded using the `PhotoImage()` constructor.

```
button = [['b1','a',0,500],['b2','b',70,500],
['b3','c',140,500],['b4','d',210,500],['b5','e',280,500],
['b6','f',350,500],['b7','g',420,500],['b8','h',490,500],
['b9','i',560,500],['b10','j',630,500],
['b11','k',700,500],['b12','l',770,500],
['b13','m',840,500],['b14','n',0,570],
['b15','o',70,570],['b16','p',140,570],
['b17','q',210,570],['b18','r',280,570],
['b19','s',350,570],['b20','t',420,570],
['b21','u',490,570],['b22','v',560,570],
['b23','w',630,570],['b24','x',700,570],
['b25','y',770,570],['b26','z',840,570]]
for q1 in button:

exec('{}=Button(window,bd=0,command=lambda:check("{}","{}"),bg="#96AE31",activebackground="#E7FFFF",font=10,image={})'.format(q1[0],q1[1],q1[0],q1[1]))

exec('{}.place(x={},y={})'.format(q1[0],q1[2],q1[3]))
```

The `button` list contains sublists, each representing a button to select a letter.

Each sublist consists of a unique button name, a letter, and the x and y coordinates for the button's placement.

Using a loop, the code dynamically creates button objects and assigns a command to each button using a lambda function.

The button objects are created using the `Button()` constructor and positioned within the window using the `place()` method.

```
han = [['c1','h1'],['c2','h2'],['c3','h3'],['c4','h4'],['c5','h5'],['c6','h6'],['c7','h7']]
for p1 in han:

exec('{}=Label(window,bg="#96AE31",image={})'.format(p1[0],p1[1]))
```

The `han` list contains sublists, each representing a hangman image label.

Each sublist consists of a unique label name and the corresponding hangman image.

Using a loop, the code dynamically creates label objects for each hangman image using the `Label()` constructor.

The labels are set to have a background color of '#96AE31'.

```
c1.place(x=300,y=0)
```

The first hangman image label, `c1`, is positioned at coordinates (300, 0) within the window.

This is the initial hangman image displayed at the start of the game.

```
def close():
    global run
    answer = messagebox.askyesno('ALERT','YOU WANT TO EXIT THE GAME?')
    if answer:
        run = False
        window.destroy()


e1 = PhotoImage(file='button_exit.png')
```

```
ex =
Button(window,bd=0,command=close,bg="#96AE31
",activebackground="#96AE31",font=10,image=e1)
ex.place(x=770,y=10)
```

The `close()` function is defined to handle the exit button's command.

It displays a messagebox to confirm if the player wants to exit the game.

If the player confirms the exit, `run` is set to `False`, and the window is destroyed.

An image object, `e1`, is created for the exit button using the `PhotoImage()` constructor.

A button object, `ex`, is created with the exit button's properties and positioned at coordinates (770, 10) within the window.

```
s2 = 'SCORE:' + str(score)
s1 = Label(window, text=s2, bg="#739F02",
font=("jokerman", 25))
s1.place(x=10, y=10)
```

The current score is converted to a string and combined with the "SCORE:" label text.

A label, `s1`, is created to display the score using a green background color and the "jokerman" font at a size of 25.

The label is positioned at coordinates (10, 10) within the window.

```
def correct_answer_sound():
    playsound("correct_answer_tring!.mp3")


def wrong_answer_sound():
    playsound("wrong_answer_dejected.mp3")
```

```
def game_win__sound():
    playsound("game_success_tring!.mp3")


def lost_game_sound():
    playsound("game_fail_toptolow.mp3")
```

Four functions are defined to play sound effects during the game.

Each function uses the `playsound()` function from the `playsound` module to play a specific sound file.

```
def check(letter, button):
    global lose_count, win_count, run, score
    exec('{}.destroy()'.format(button))
    if letter in selected_word:
        correct_answer_sound()
        for i in range(0, len(selected_word)):
            if selected_word[i] == letter:
                win_count += 1
                exec('d{}.config(text="{}")'.format(i,
letter.upper()))
        if win_count == len(selected_word):
            score += 1
            game_win__sound()
            answer = messagebox.askyesno('GAME
OVER', 'YOU WON!\nWANT TO PLAY AGAIN?')
            if answer == True:
                run = True
                window.destroy()
            else:
                run = False
                window.destroy()
    else:
        lose_count += 1
        wrong_answer_sound()
        exec('c{}.destroy()'.format(lose_count))
        if lose_count == 7:
            lost_game_sound()
            answer = messagebox.askyesno('GAME
OVER', 'YOU LOST!\nWANT TO PLAY AGAIN?')
            if answer == True:
                run = True
                window.destroy()
            else:
                run = False
                window.destroy()
```

The `check()` function is defined to handle the button clicks and check if the selected letter is correct or incorrect.

The function takes two parameters: `letter` (the selected letter) and `button` (the name of the button clicked).

The function first destroys the button that was clicked using the `destroy()` method.

If the selected letter is present in the selected word, the function plays the correct answer sound and updates the corresponding dashes with the correct letter.

The `win_count` is incremented for each correct guess.

If `win_count` is equal to the length of the selected word, it means the player has guessed all the letters correctly and won the game.

The player's score is increased by 1, and a messagebox is displayed to ask if they want to play again.

If the player chooses to play again, `run` is set to `True`, and the window is destroyed.

If the player chooses not to play again, `run` is set to `False`, and the window is destroyed.

If the selected letter is not in the selected word, the function plays the wrong answer sound and removes a part of the hangman image.

The `lose_count` is incremented for each incorrect guess.

If `lose_count` reaches 7, it means the player has made too many incorrect guesses and lost the game.

The lost game sound is played, and a messagebox is displayed to ask if the player wants to play again.

The same logic as for the win condition is applied to determine the player's choice and update the `run` variable accordingly.

window.mainloop()

The main game loop is started using the `mainloop()` method of the `Tk` window object.

This method runs the event loop, allowing the program to respond to user interactions and update the GUI.