

GUI Hangman Tutorial

In this tutorial you will learn to create a hangman game using with GUI using the MVC framework.

Setup Environment

Steps:

1. Create virtual environment:

- Mac users, always use `python3` instead of `python`
- `python -m venv .venv`

2. make `.gitignore` file:

- add virtual environment folder: `/.venv`

3. activate virtual environment

- o open a new terminal:
- o Windows: `.\.venv\Scripts\activate`
- o Mac: `source .venv/bin/activate`
- o Your prompt should start with `(.venv)`

4. Update pip

- o `python -m pip install --upgrade pip`

5. Install packages

- o `python -m pip install PyQt6`

6. Add files to the project directory

- o `hangman.ui`
- o `main_window.py`
- o Rename `main_window.py` to `hangman.py`

7. Generate the Python UI file

- o `pyuic6 -o ui_hangman.py -x hangman.ui`

8. Update hangman.py file

- o in line 3 change `<ui_filename>` to `ui_hangman`
- o Run the `hangman.py` file and your UI should show.

9. Download the `assets.zip` file into your project directory

- extract the file to create a `assets` directory with 12 image files
- confirm the files are there and then delete the `assets.zip` file

10. Create a new Python script file called `datastore.py`

11. Download the `dictionary.txt` file into your project directory

12. Your project directory is now set-up. It should contain:

- o `.venv` directory - virtual environment folder
- o `assets` directory - containing 12 gallows images
- o `.githubattributes` - file for GitHub operations
- o `.githubignore` - file listing the directories or files not to be synced
- o `datastore.py` - the Python script containing the model component of our program
- o `dictionary.txt` - file containing the word list we will use

- `hangman.py` - the Python script containing the control component of our program
- `hangman.ui` - Qt Designer file which is used to generate our UI
- `ui_hangman.py` - the Python script containing the view component of our program

13. make sure all your files are saved, then in GitHub Desktop:

- make a commit
- push to repository

Create Datastore

Looking at the Class Diagram, and start with the simplest class Datastore.

All it does is loads the `dictionary.txt` file and then returns a random word when requested.

Create and initialise class

```
class Datastore():

    def __init__(self):
        """
        initialise datastore by reading dictionary file and
        adding each word into a list
        """
        with open("dictionary.txt","r") as word_file:
            self.words = word_file.read().splitlines()
```

Create a test file `test.py` to test the `Datastore` as we go along.

```
from datastore import Datastore

db = Datastore()
```

Run it.

Add `get_word()` method

```
import random

class Datastore():

    def __init__(self):
        """
        initialise datastore by reading dictionary file and
        adding each word into a list
        """
        with open("dictionary.txt","r") as word_file:
            self.words = word_file.read().splitlines()

    def get_word(self):
        """
        returns a random word of 3 or more characters
        return: str
        """
        word = ""
```

```

while len(word) < 3:
    word = random.choice(self.words)

return word

```

Now use `test.py` to call the `get_word()` method a couple of time to ensure it is working and random

```

from datastore import Datastore

db = Datastore()

print(db.get_word())

```

Datastore done.

Initialise MainWindow Class

Open up the `hangman.py` file and put some plans into the constructor, as well as some doc strings

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from ui_hangman import Ui_Mainwindow

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#

        # ----- initialise UI with starting values ----- #

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

    def signals(self):
        """
        Connects the UI buttons to the corresponding functions (see slots)
        """
        pass

    # ----- slots ----- #

if __name__ == '__main__':

```

```
app = QApplication(sys.argv)
main_win = MainWindow()
main_win.show()
sys.exit(app.exec())
```

Game Variables

What variables does the game need.

- Database
- A word to guess
- The word as guessed so far
- The number of misses

```
import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

    def signals(self):
        """
        Connects the UI buttons to the corresponding functions (see slots)
        """
        pass

    # ----- slots ----- #

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
```

```
main_win.show()
sys.exit(app.exec())
```

Run and check for errors

UI starting values

Guessed word

When the game starts, the `Mainwindow` needs to get a word, and then work out how many `_` to use to display it's letters. Create a `choose_word()` method and call it from the constructor.

```
import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """
        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)

    def signals(self):
```

```

        """
        Connects the UI buttons to the corresponding functions (see slots)
        """

        pass

# ----- slots ----- #

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Add breakpoints beside line 31 `self.main_win.show()` so we can observe the values of `self.word` and `self.guessed_word`. Then run in debugging mode.

Display Guessed Word

Now that we have the guessed word we should display it on the UI. The UI needs a string to display in it's label, so we need to convert our list to a string.

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from ui_hangman import Ui_MainWindow
from datastore import Datastore

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()

    def show(self):
        """
        Displays main window
        """

        self.main_win.show()

```

```

def choose_word(self):
    """
    Gets word from datastore, and creates corresponding
    list for guessed letters
    """
    self.word = self.db.get_word()
    self.guessed_word = ["_"] * len(self.word)

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    pass

# ----- slots ----- #

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = Mainwindow()
    main_win.show()
    sys.exit(app.exec())

```

Now, run and test it.

Display Gallows

Finally, we need to show the correct progress of the gallows.

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()

```

```

self.ui.setupUi(self.main_win)

# ----- initialise game variables -----#
self.db = Datastore()
self.word = ""
self.guessed_word = []
self.misses = 0

# ----- initialise UI with starting values ----- #
self.choose_word()
self.display_guesses()
self.display_gallows()

def show(self):
    """
    Displays main window
    """
    self.main_win.show()

def choose_word(self):
    """
    Gets word from datastore, and creates corresponding
    list for guessed letters
    """
    self.word = self.db.get_word()
    self.guessed_word = ["_"] * len(self.word)

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    pass

```



```

# ----- slots ----- #

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Run and test. Since the 0 misses is blank, change the value of misses a couple of time to make sure that it is working. Don't forget to change it back to 0

User Interface is now initialised.

Slots and Signals

Unlike a console mode application, which is executed in a sequential manner, a GUI based application is event driven. Functions or methods are executed in response to user's actions like clicking on a button, selecting an item from a collection or a mouse click etc., called **events**.

Widgets used to build the GUI interface act as the source of such events. Each PyQt widget, which is derived from QObject class, is designed to emit '**signal**' in response to one or more events. The signal on its own does not perform any action. Instead, it is 'connected' to a '**slot**'. The slot can be any **callable Python function**.

Quit button

Lets start with the simplest. The quit button. We already have a `signals()` method, but it needs to be called at the end of the constructor.

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_MainWindow
from datastore import Datastore

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()

```

```

self.display_guesses()
self.display_gallows()
self.signals()

def show(self):
    """
    Displays main window
    """
    self.main_win.show()

def choose_word(self):
    """
    Gets word from datastore, and creates corresponding
    list for guessed letters
    """
    self.word = self.db.get_word()
    self.guessed_word = ["_"] * len(self.word)

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)

    # ----- slots ----- #

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Test it and make sure the

New word button

Now lets get a bit more complicated. For the new word button we need to create our own function (method).

```
import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """
        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)

    def display_guesses(self):
```

```

"""
Display the guessed letters to the UI
"""

display_word = ""
for character in self.guessed_word:
    display_word = display_word + character + " "

self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """

    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """


    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """

    # get new word
    self.choose_word()
    self.display_guesses()
    # reset GUI
    self.misses = 0
    self.display_gallows()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = Mainwindow()
    main_win.show()
    sys.exit(app.exec())

```

Now test the new word button. Although e can't see the word, we should be able to see the number of  change.

Letter Buttons

This is a bit trickier. When we click a letter button, we want to call a function that will:

- Disable the clicked button
- Checks if letter is in the word
- Updates the state of the game

We have 26 letter buttons on our UI, so that could mean 26 functions, but that would be repeating yourself. So instead we write one function and pass the button object as an argument. This way the function can call methods on that button.

Getting the button letter value

Let's try this with the **A** button, and, at first just try and get the letter value of the button.

```
import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """
        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)
```

```

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """

    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """

    # get new word
    self.choose_word()
    self.display_guesses()

    # reset GUI
    self.misses = 0
    self.display_gallows()

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,
    checks for state of the game.
    """

    # get letter
    guess = button.text().lower()
    print(guess)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()

```

```
main_win.show()
sys.exit(app.exec())
```

Running it should print `a` in the terminal

Disabling the button

Now lets disable the button to make sure they cannot click it again.

```
import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """

        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """

        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)
```

```

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """
    # get new word
    self.choose_word()
    self.display_guesses()
    # reset GUI
    self.misses = 0
    self.display_gallows()

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,
    checks for state of the game.
    """
    # get letter
    guess = button.text().lower()
    print(guess)

    # disable button
    button.setEnabled(False)

```



```

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Test it to make sure the A button only displays a `a` in the terminal once, no matter how many times you press it.

Check if letter is in the word

Now we need to check if the letter is in the word

If it is in the word, then we need to reveal it in the `guessed word`

If it is not in the word, we need to increase the gallows

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class MainWindow:
    def __init__(self):
        """
        Initialise game window
        """
        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """
        self.main_win.show()

```

```

def choose_word(self):
    """
    Gets word from datastore, and creates corresponding
    list for guessed letters
    """
    self.word = self.db.get_word()
    self.guessed_word = ["_"] * len(self.word)

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """
    # get new word
    self.choose_word()
    self.display_guesses()
    # reset GUI
    self.misses = 0
    self.display_gallows()

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,

```

```

        checks for state of the game.
        """

        # get letter
        guess = button.text().lower()
        print(guess)

        # disable button
        button.setEnabled(False)

        # Check if letter is in word
        if guess in self.word:
            # add guess to guessed_word
            for index, letter in enumerate(self.word):
                if guess == letter:
                    self.guessed_word[index] = guess.upper()
            # display guessed_word
            self.display_guesses()
        else:
            # add to the misses count, update GUI and check if game over
            self.misses += 1
            self.display_gallows()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = Mainwindow()
    main_win.show()
    sys.exit(app.exec())

```

Test by running enough times that you get both a reveal and a increase in the gallows.

Check for end of game conditions

Once the game state has been updated, we need to check and see if the game has ended.

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""

```

```

self.guessed_word = []
self.misses = 0

# ----- initialise UI with starting values ----- #
self.choose_word()
self.display_guesses()
self.display_gallows()
self.signals()

def show(self):
    """
    Displays main window
    """
    self.main_win.show()

def choose_word(self):
    """
    Gets word from datastore, and creates corresponding
    list for guessed letters
    """
    self.word = self.db.get_word()
    self.guessed_word = ["_"] * len(self.word)

def display_guesses(self):
    """
    Display the guessed letters to the UI
    """
    display_word = ""
    for character in self.guessed_word:
        display_word = display_word + character + " "

    self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))

```

```

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """

    # get new word
    self.choose_word()
    self.display_guesses()
    # reset GUI
    self.misses = 0
    self.display_gallows()

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,
    checks for state of the game.
    """

    # get letter
    guess = button.text().lower()
    print(guess)

    # disable button
    button.setEnabled(False)

    # Check if letter is in word
    if guess in self.word:
        # add guess to guessed_word
        for index, letter in enumerate(self.word):
            if guess == letter:
                self.guessed_word[index] = guess.upper()
        # display guessed_word
        self.display_guesses()
        # check for win
        if "_" not in self.guessed_word:
            self.ui.result_lb.setText("winner!")
    else:
        # add to the misses count, update GUI and check if game over
        self.misses += 1
        self.display_gallows()
        # check for loss
        if self.misses == 11:
            self.ui.result_lb.setText(f"The word was {self.word.upper()}")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = Mainwindow()
    main_win.show()
    sys.exit(app.exec())

```

This is going to be hard to test since there is no more than 3 letter word consisting of only `a`, so let's copy the `a` button signal and repeat it for all the other buttons. Also, to make testing easier, let print the word to the terminal.

```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """

        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """

        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)
        print(self.word)

    def display_guesses(self):
        """
        Display the guessed letters to the UI
        """

        display_word = ""
        for character in self.guessed_word:
            display_word = display_word + character + " "

```

```

self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))
    self.ui.b_btn.clicked.connect(lambda: self.letter_btn(self.ui.b_btn))
    self.ui.c_btn.clicked.connect(lambda: self.letter_btn(self.ui.c_btn))
    self.ui.d_btn.clicked.connect(lambda: self.letter_btn(self.ui.d_btn))
    self.ui.e_btn.clicked.connect(lambda: self.letter_btn(self.ui.e_btn))
    self.ui.f_btn.clicked.connect(lambda: self.letter_btn(self.ui.f_btn))
    self.ui.g_btn.clicked.connect(lambda: self.letter_btn(self.ui.g_btn))
    self.ui.h_btn.clicked.connect(lambda: self.letter_btn(self.ui.h_btn))
    self.ui.i_btn.clicked.connect(lambda: self.letter_btn(self.ui.i_btn))
    self.ui.j_btn.clicked.connect(lambda: self.letter_btn(self.ui.j_btn))
    self.ui.k_btn.clicked.connect(lambda: self.letter_btn(self.ui.k_btn))
    self.ui.l_btn.clicked.connect(lambda: self.letter_btn(self.ui.l_btn))
    self.ui.m_btn.clicked.connect(lambda: self.letter_btn(self.ui.m_btn))
    self.ui.n_btn.clicked.connect(lambda: self.letter_btn(self.ui.n_btn))
    self.ui.o_btn.clicked.connect(lambda: self.letter_btn(self.ui.o_btn))
    self.ui.p_btn.clicked.connect(lambda: self.letter_btn(self.ui.p_btn))
    self.ui.q_btn.clicked.connect(lambda: self.letter_btn(self.ui.q_btn))
    self.ui.r_btn.clicked.connect(lambda: self.letter_btn(self.ui.r_btn))
    self.ui.s_btn.clicked.connect(lambda: self.letter_btn(self.ui.s_btn))
    self.ui.t_btn.clicked.connect(lambda: self.letter_btn(self.ui.t_btn))
    self.ui.u_btn.clicked.connect(lambda: self.letter_btn(self.ui.u_btn))
    self.ui.v_btn.clicked.connect(lambda: self.letter_btn(self.ui.v_btn))
    self.ui.w_btn.clicked.connect(lambda: self.letter_btn(self.ui.w_btn))
    self.ui.x_btn.clicked.connect(lambda: self.letter_btn(self.ui.x_btn))
    self.ui.y_btn.clicked.connect(lambda: self.letter_btn(self.ui.y_btn))
    self.ui.z_btn.clicked.connect(lambda: self.letter_btn(self.ui.z_btn))

    # ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """
    # get new word

```

```

self.choose_word()
self.display_guesses()
# reset GUI
self.misses = 0
self.display_gallows()

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,
    checks for state of the game.
    """
    # get letter
    guess = button.text().lower()
    print(guess)

    # disable button
    button.setEnabled(False)

    # Check if letter is in word
    if guess in self.word:
        # add guess to guessed_word
        for index, letter in enumerate(self.word):
            if guess == letter:
                self.guessed_word[index] = guess.upper()
        # display guessed_word
        self.display_guesses()
        # check for win
        if "_" not in self.guessed_word:
            self.ui.result_lb.setText("Winner!")
    else:
        # add to the misses count, update GUI and check if game over
        self.misses += 1
        self.display_gallows()
        # check for loss
        if self.misses == 11:
            self.ui.result_lb.setText(f"The word was {self.word.upper()}")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Now test your program and get to both end game states (win and loss)

Updating the `New word` button

Almost there. In testing you may have noticed that when you finished a game:

- after pressing the `New word` button, the letter buttons were still disabled
- you can keep pressing letter buttons
- the result message is still on the screen

Let fix those.


```

import sys
from PyQt6.Qtwidgets import QApplication, QMainWindow
from PyQt6.QtCore import QApplication
from PyQt6.QtGui import QPixmap
from ui_hangman import Ui_Mainwindow
from datastore import Datastore

class Mainwindow:
    def __init__(self):
        """
        Initialise game window
        """

        # ----- setup UI elements ----- #
        self.main_win = QMainWindow()
        self.ui = Ui_Mainwindow()
        self.ui.setupUi(self.main_win)

        # ----- initialise game variables -----#
        self.db = Datastore()
        self.word = ""
        self.guessed_word = []
        self.misses = 0

        # ----- initialise UI with starting values ----- #
        self.choose_word()
        self.display_guesses()
        self.display_gallows()
        self.signals()

    def show(self):
        """
        Displays main window
        """

        self.main_win.show()

    def choose_word(self):
        """
        Gets word from datastore, and creates corresponding
        list for guessed letters
        """

        self.word = self.db.get_word()
        self.guessed_word = ["_"] * len(self.word)
        print(self.word)

    def display_guesses(self):
        """
        Display the guessed letters to the UI
        """

        display_word = ""
        for character in self.guessed_word:
            display_word = display_word + character + " "

```

```

self.ui.word_lb.setText(display_word)

def display_gallows(self):
    """
    Displays the gallow progression to the UI
    """
    file_name = (f"./assets/{self.misses}.png")
    gallow = QPixmap(file_name)
    self.ui.gallow_lb.setPixmap(gallow)

def signals(self):
    """
    Connects the UI buttons to the corresponding functions (see slots)
    """
    # control buttons
    self.ui.quit_btn.clicked.connect(QCoreApplication.instance().quit)
    self.ui.new_word_btn.clicked.connect(self.new_word_btn)

    # letter buttons
    self.ui.a_btn.clicked.connect(lambda: self.letter_btn(self.ui.a_btn))
    self.ui.b_btn.clicked.connect(lambda: self.letter_btn(self.ui.b_btn))
    self.ui.c_btn.clicked.connect(lambda: self.letter_btn(self.ui.c_btn))
    self.ui.d_btn.clicked.connect(lambda: self.letter_btn(self.ui.d_btn))
    self.ui.e_btn.clicked.connect(lambda: self.letter_btn(self.ui.e_btn))
    self.ui.f_btn.clicked.connect(lambda: self.letter_btn(self.ui.f_btn))
    self.ui.g_btn.clicked.connect(lambda: self.letter_btn(self.ui.g_btn))
    self.ui.h_btn.clicked.connect(lambda: self.letter_btn(self.ui.h_btn))
    self.ui.i_btn.clicked.connect(lambda: self.letter_btn(self.ui.i_btn))
    self.ui.j_btn.clicked.connect(lambda: self.letter_btn(self.ui.j_btn))
    self.ui.k_btn.clicked.connect(lambda: self.letter_btn(self.ui.k_btn))
    self.ui.l_btn.clicked.connect(lambda: self.letter_btn(self.ui.l_btn))
    self.ui.m_btn.clicked.connect(lambda: self.letter_btn(self.ui.m_btn))
    self.ui.n_btn.clicked.connect(lambda: self.letter_btn(self.ui.n_btn))
    self.ui.o_btn.clicked.connect(lambda: self.letter_btn(self.ui.o_btn))
    self.ui.p_btn.clicked.connect(lambda: self.letter_btn(self.ui.p_btn))
    self.ui.q_btn.clicked.connect(lambda: self.letter_btn(self.ui.q_btn))
    self.ui.r_btn.clicked.connect(lambda: self.letter_btn(self.ui.r_btn))
    self.ui.s_btn.clicked.connect(lambda: self.letter_btn(self.ui.s_btn))
    self.ui.t_btn.clicked.connect(lambda: self.letter_btn(self.ui.t_btn))
    self.ui.u_btn.clicked.connect(lambda: self.letter_btn(self.ui.u_btn))
    self.ui.v_btn.clicked.connect(lambda: self.letter_btn(self.ui.v_btn))
    self.ui.w_btn.clicked.connect(lambda: self.letter_btn(self.ui.w_btn))
    self.ui.x_btn.clicked.connect(lambda: self.letter_btn(self.ui.x_btn))
    self.ui.y_btn.clicked.connect(lambda: self.letter_btn(self.ui.y_btn))
    self.ui.z_btn.clicked.connect(lambda: self.letter_btn(self.ui.z_btn))

def set_button_enabled(self, val):
    """
    Changes the enabled status of the letter buttons to passed value
    val: bool
    """
    self.ui.a_btn.setEnabled(val)

```

```

self.ui.b_btn.setEnabled(val)
self.ui.c_btn.setEnabled(val)
self.ui.d_btn.setEnabled(val)
self.ui.e_btn.setEnabled(val)
self.ui.f_btn.setEnabled(val)
self.ui.g_btn.setEnabled(val)
self.ui.h_btn.setEnabled(val)
self.ui.i_btn.setEnabled(val)
self.ui.j_btn.setEnabled(val)
self.ui.k_btn.setEnabled(val)
self.ui.l_btn.setEnabled(val)
self.ui.m_btn.setEnabled(val)
self.ui.n_btn.setEnabled(val)
self.ui.o_btn.setEnabled(val)
self.ui.p_btn.setEnabled(val)
self.ui.q_btn.setEnabled(val)
self.ui.r_btn.setEnabled(val)
self.ui.s_btn.setEnabled(val)
self.ui.u_btn.setEnabled(val)
self.ui.v_btn.setEnabled(val)
self.ui.w_btn.setEnabled(val)
self.ui.x_btn.setEnabled(val)
self.ui.y_btn.setEnabled(val)
self.ui.z_btn.setEnabled(val)

# ----- slots ----- #
def new_word_btn(self):
    """
    Chooses a new word and resets the UI
    """
    # get new word
    self.choose_word()
    self.display_guesses()
    # reset GUI
    self.misses = 0
    self.display_gallows()
    self.set_button_enabled(True)
    self.ui.result_lb.setText("")

def letter_btn(self, button):
    """
    Disables the clicked button, checks if letter is in the word,
    checks for state of the game.
    """
    # get letter
    guess = button.text().lower()
    print(guess)

    # disable button
    button.setEnabled(False)

    # Check if letter is in word
    if guess in self.word:
        # add guess to guessed_word

```

```

        for index, letter in enumerate(self.word):
            if guess == letter:
                self.guessed_word[index] = guess.upper()
            # display guessed_word
            self.display_guesses()
            # check for win
            if "_" not in self.guessed_word:
                self.ui.result_lb.setText("Winner!")
        else:
            # add to the misses count, update GUI and check if game over
            self.misses += 1
            self.display_gallows()
            # check for loss
            if self.misses == 11:
                self.ui.result_lb.setText(f"The word was {self.word.upper()}")
                self.set_button_enabled(False)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec())

```

Test you game through to both end conditions. Then delete the terminal print statements in line 46 and line 160 and we're done.