Github Link:
Requirement of the game:

1. Randomly generate a four-digit number:

   import the inner function "random" to generate a random number

```
import random

def generate_random_number(): # generate a random number for the game
    return random.randint(1000, 9999)
```

2. The program will keep asking the user to guess the number until the player guesses it correctly or has quitted:

   the variable 'guess' is a value from the input of players, the function require players input a four-digital number or 'q' for quitting game

```
def get_user_guess():
    while True:
        guess = input("Enter a 4-digit number or 'q' to quit: ") # require player in put a 4-digit number or 'q'
        if guess.lower() == 'q':
            return None
        if len(guess) != 4 or not guess.isdigit(): # if the input is not a number or a 4 digital number, it's invalid input
            print("Invalid input.")
        else:
            return guess
```

3. When the number is entered, the program will respond with hints using ‘circle’ and ‘x’ to show how accurate the guess was:
   a. A ‘circle’ indicates that one digit is correct and is in the right spot
   b. A ‘x’ indicates that one digit is correct but in the wrong spot

   - translate the input to a string and compare each position of the random and each position of the input whether it matches.
   - if the number and position are correct, add 1 in the variable 'O_count',
   - if the number is correct only, add 1 in the variable "X_count"
   - print the number of O and the number of X to show players

4. Once the game is finished,
   a. The number of attempts taken will be displayed
   b. the player will be asked to quit or to play again

   - when the game repeats, add 1 in the variable “play_times” to compute how many times players play.
   - the program will ask plays whether continue the game at the end before the game finishes.

5. Player can quit the game anytime:

players can quit game at any time when the input is “q”

```
while play_again:
    secret_number = str(generate_random_number())
    play_times = 0

    while True:
        user_guess = get_user_guess()
        if user_guess is None: # if the input is 'q', break the program
            break

        play_times += 1
        O_count, x_count = compare_numbers(secret_number, user_guess)

        print(f"Hints: {'O' * O_count} {'X' * x_count}")

        if O_count == 4:
            print(f"You've guessed the number {secret_number} in {play_times} play times.")
            break

    play_again_input = input("play again? (yes/no): ")
    play_again = play_again_input.lower() == 'yes'

print("game finish")
```

Automated unit testing tool: unittest

1. Test the function of generating a random number:

   Object:
   Make sure the generate_random_number function generates random numbers in the range between 1000 and 9999

   Requirement:

   - Use “self.assertTrue()” to assert that a condition is true
   - “generate_random_number()” function returns a random number, which we store in “random_number variable”
   - use “1000 <= random_number <= 9999” to make sure “random_number” is between 1000 and 9999

```python
import unittest
from guess_game import generate_random_number


def test_generate_random_number(self):
    random_number = generate_random_number()
    self.assertTrue(1000 <= random_number <= 9999)


if __name__ == "__main__":
    unittest.main()
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Software-Unit-Testing-Report> python -m unittest test_random_number.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

2. Test the function of getting the input that players guess

   Object:
   The goal of the test is to verify that the “get_user_guess” function behaves correctly in different situations, including player exits, invalid input, and valid input.

Requirement:

- test that if players input ‘q’ and the function should return “None”, the game will finish.

- test that if players input “abcd”, it’s a invalid value, the function should print “Invalid input.” to show players.

- test that if players input “123”, it’s a invalid value, the function should print “Invalid input.” to show players.

```python
import unittest
from guess_game import get_user_guess

def test_get_user_guess(self):
    with unittest.mock.patch("builtins.input", side_effect=["12345", "abcd", "123", "q"]):
        self.assertEqual(get_user_guess(), None)  # Test user quitting
        self.assertEqual(get_user_guess(), None)  # Test invalid input (not a digit)
        self.assertEqual(get_user_guess(), None)  # Test invalid input (not 4 digits)
        self.assertEqual(get_user_guess(), "1234")  # Test valid input

```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Software-Unit-Testing-Report> python -m unittest test_player_guess.py

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```

3. Test the function of comparing the input and the random number

Object:
The goal of the test is to ensure that the “compare_numbers” function correctly calculates and returns the corresponding hints (O and X) for different guesses and correct answers

Requirement:

- when no numbers match, the function will return (0, 0)
- when numbers and position all match , the function will return (4, 0)
- when numbers match but position not match, the function will return (0, 4)
- when numbers all match but two of them of position not match, the function will return (2, 2)

```python
import unittest
from guess_game import compare_numbers

def test_compare_numbers(self):
    self.assertEqual(compare_numbers("1234", "5678"), (0, 0))
    self.assertEqual(compare_numbers("1234", "1234"), (4, 0))
    self.assertEqual(compare_numbers("1234", "4321"), (0, 4))
    self.assertEqual(compare_numbers("1234", "1243"), (2, 2))
    self.assertEqual(compare_numbers("1122", "2211"), (0, 4))  # Test repeated digits
    self.assertEqual(compare_numbers("1234", "5671"), (0, 1))  # Test partial match
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Software-Unit-Testing-Report> python -m unittest test_compare_number.py

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
PS D:\Software-Unit-Testing-Report>
```

4. Test the main program

Object:
The test goal is to verify that the game program executes correctly in the loop of the game, and handles player input, compares guesses, displays results

Requirement:

- Use “unittest.mock.patch” to simulate the behavior of input and print functions
- Enter "1234" followed by "q" to simulate the player wishing to quit the game after two guesses

```python
import unittest
from guess_game import main

def test_main(self):
    with unittest.mock.patch("builtins.input", side_effect=["1234", "q"]):
        with unittest.mock.patch("builtins.print"):
            main()  # Test the main loop
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS D:\Software-Unit-Testing-Report> python -m unittest test_main_program.py

----------------------------------------------------------------------
Ran 0 tests in 0.000s

OK
```