**ME400 - Computer Applications in Mechanical Engineering**

**Exercise #7**

**Instructions:**

**Complete each of the following steps. When you have completed all steps assigned, upload the files associated with the exercise on Canvas. Make sure the filenames you upload as part of your exercise submission have the exact names specified in each steps of this exercise. If the filenames do not match exactly, then your submission will not be graded.**

## Background

Simon is an electronic game of short-term memory skill invented by Ralph H. Baer and Howard J. Morrison and programmed by Lenny Cope. The game is implemented using a combination of hardware and software, and the physical interface consists of four quadrants as shown in Figure 1.


*Figure 1: Image of Simon Game*

When the game starts, a random sequence is presented to the user by lighting the associated quadrants and playing a tone that is specific to each quadrant. Then the user is required to repeat the sequence. If the user succeeds, then the series becomes progressively longer and more complex. Once the user fails to enter the correct sequence, the game is over.

A simple version of this game will be implemented as part of this exercise. This task will test your understanding of incorporating and writing functions, passing arguments to functions, implementing logic in loops, implementing branching logic, and working with arrays.

## Generate Random Numbers – Raspberry Pi Pico

To generate a pseudo-random number in a MicroPython application, we need to add the following import:

- ***from random import randrange, seed***

And then use the following functions:

- ***randrange(min, max)*** - generates a pseudo-random integer between ***min*** and ***max-1***. ***A bit easier to setup our range of numbers.***
- ***seed(n)*** – instructs MicroPython to start generating a random number sequence at a particular point

If ***seed*** is called with no arguments, then a random hardware number, typically system time, will be used as the starting point for generating random numbers each run.

This function (i.e., seeding) should only be performed once upon program startup, and such a call looks similar to the following:

***seed()***

Typically, we want to generate a random integer on a fixed range (say 0 to 3). To do so, we might implement code similar to the following:

***x = randrange(0, 4)***

Notice the upper limit must be one greater than the upper value of the range we want to generate. Also, notice that the ***randrange()*** function only returns the integers 0, 1, 2, and 3 for the specified range.

## Problem Statement

Write a program, which meets the requirements listed below:

### Requirements

1. Create a sub directory named "***exercise_07***" in your ME 400 folder on the Z: drive. Download the files ***ir_remote_driver.py, lcd_display_driver.py,*** and ***exercise_07_starter.py*** from the exercise resources on the homepage in Canvas. ***DO NOT make changes to the files ir_remote_driver.py and lcd_display_driver.py*** Changes should only be made to the file ***exercise_07_starter.py.***

2. Connect the Willie Says interface board to the Pico. The orientation is critical and must be oriented as shown in Figure 1. Also, use two Male-to-Male jumper wires to connect one side of a push button to ground and the other side to digital pin 21 as shown in Figure 1.

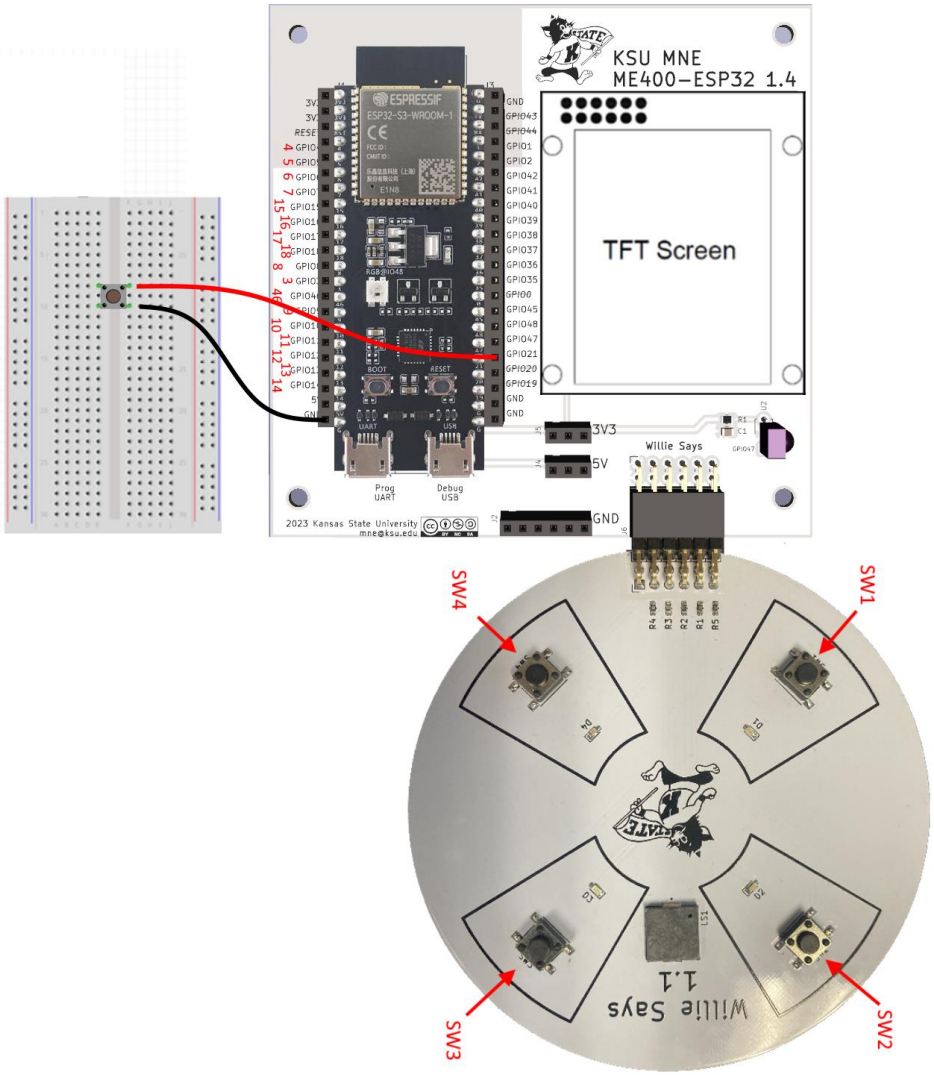


Figure 1: Willie Says Connections

3. The code supplied in the file ***exercise_07_starter.py*** is the solution to the exercise you completed Week #6. We could start using your version of the exercise, but to avoid potential problems with your original submission and to avoid confusion we will all be starting with the same code.

4. The code written in Exercise #6 represented a preliminary step in the development of a Simon Says game. The program you wrote collected user input from the four switches on the Willie Says board, blinked the corresponding LED on the Willie Says board, played a tone that was unique for each tone and updated the TFT display to indicate which of the four buttons was pressed. It also, setup the code to start a new game when an external button was pressed and to end a game when the menu key is pressed. Furthermore, you explored combining program functionality and using a function to more efficiently manage you code. Now, we want to modify this program to generate an initial set of random sequences and prompt the user to enter the sequences that were displayed. To implement these changes, perform the changes outlined in the following steps.

5. Create a global array of ***integers*** named ***sequences***. Leave the list empty for now. *Note: Global variables should be defined at the top of the program before the any function calls. Any variable*

*included inside a function such as will be declared with local scope and will not be accessible in another function.*

6. Create a global ***integer*** named ***sequence_count*** and set it to ***0***. This variable will be used to track the number of random sequences to generate using the Willie Says board.

7. Create a global integer named ***current_index*** and set it to ***0***.

8. So in Exercise #6, you setup the function ***process_move()*** to display the user selection on the TFT screen, blink the associated LED, and play a unique tone. You also setup the code to display the message ***NEW GAME*** when an external button was pressed. The associated ***main loop*** routine from exercise #6 is shown below:

```
1.      while True:
2.          # Get the last input from IR remote
3.          last_key_processed = ir_remote.get_last_key_press()
4.
5.          # Return to main menu when MENU key is pressed
6.          if last_key_processed == "MENU":
7.              show_display("MAIN")
8.
9.          # Restart game if new game interrupt is hit
10.         if new_game:
11.             # Display New Game screen
12.             show_display("NEW_GAME")
13.             time.sleep(1)
14.             display.fill_rectangle(0, 0, x, y, color565(255, 0, 255))
15.             new_game = False
16.         else:
17.             if not button_pins[0].value():
18.                 process_move(0)
19.             elif not button_pins[1].value():
20.                 process_move(1)
21.             elif not button_pins[2].value():
22.                 process_move(2)
23.             elif not button_pins[3].value():
24.                 process_move(3)
```

Figure 2: Basis for Code Revisions

9. Modify the code such that when a new game is started that the sequence count is set to 4 and four random sequences are generated. With this change in mind insert code between lines ***14*** and ***15*** of the code snippet shown in Fig 2 that performs the following:

   a. Set the global variable ***sequence_count*** to ***4***.

   b. Populate the first four elements of the ***sequences*** array with a random integer between ***0*** and ***3*** using the ***randrange()*** function. This must be implemented in a ***for*** or ***while*** loop to support potentially changing the initial ***sequence_count*** to a value greater than or less than ***4***.

   c. Loop through the ***sequences*** array and call the ***process_move()*** function for each element of the array. Set the quadrant argument to the random sequence stored in the ***sequences*** array for the associated element.

d. Set the global variable ***current_index*** to zero. The ***current_index*** indicates the first sequence in the array that needs to be validated by the user.

***Run the code at this point*. It should display the message "Welcome to Willie Says" on the TFT. Press the external button. It should display the message NEW GAME, delay for one second, and then generate 4 random sequences that display on the TFT, illuminate the corresponding LEDs, and play the corresponding tones.**

10. So, this is great, but now we need to provide a way for the user to enter the 4 sequences that were displayed and check to see if the 4 sequences were entered correctly.

    a. If all of the sequences were entered correctly, then a fifth random sequence should be generated and then the 5 sequences should be displayed to the user.

    b. Step 10 should repeat until one of the sequences is entered incorrectly or twenty sequences are displayed to the user (this is an arbitrary maximum that we use to indicate the user won the game). Each time Step 10 is repeated the number of sequences will increase by one (i.e., 6 sequence are generated, then 7 sequences are generated, and so on).

11. Upon completion of Step 10, a message should be displayed that indicates if the user won or lost the game. Then the user should be able to start a new game by pressing the external button again. Of course, we still have work to do to get all of this done.

12. Let's tackle the task of collecting the user's input to see if they enter the correct response first. Start by adding a ***integer*** variable named ***user_action*** with local scope in the ***main loop*** function. Initialize it with a value of -1. Ideally this new line of code should be inserted between lines 2 and 3 of the code snippet shown in Figure 2.

    In general, we are adding the variable ***user_actio***n because we want to determine whether the user pressed sw#1, sw#2, sw#3, or sw#4. We don't need a global variable because we don't plan to use the variable outside of the ***main loop*** routine unless it is passed as an argument to a function.

13. To implement the second part of the change described in Step 12, change the logic such that when sw#1 is pressed the variable ***user_action*** is set to 0, when sw#2 is pressed the variable ***user_action*** is set to 1, when sw#3 is pressed the variable ***user_action*** is set to 2, and when sw#4 is pressed the variable ***user_action*** is set to 3. This should be done by modifying the existing code shown in lines 17 to 24 of the code snippet shown in Figure 2.

14. So now that we have a set a variable to store the user selection, we need to check the associated sequence to see if it is correct. To facilitate this functionality, after like 24 of the code snippet shown in Figure 2. Make sure the new code is outside of the elif but inside of the else from line 16.

```
1.              # If any button was pressed, then process it
2.              if user_action > -1:
3.                  # Call to process_user_selections goes here
4.                  user_action = -1
```

Figure 3: Implementation of action checks

This sets the variable ***user_action*** back to ***-1*** so that any code that we want to execute when the button is pressed is only executed ***once***. This logic is required because when a button is pressed it does not block program execution and the ***main loop*** routine might execute thousands of times when the button is pressed.

***Run the sketch to verify we did not create any errors.***

15. Before continuing let's consider an example to illustrate the desired program functionality. Consider the case where 4 random sequences of 0, 2, 2, and 1 have been displayed to the user and you want the player to enter the 4 sequences in order using the buttons Willie Says board. For such a case the ***sequences*** array contains the values 0, 2, 2, and 1, and the variable ***sequence_count*** is equal to 4. Let's consider a possible scenario that unfolds as follows:

    a. The user presses sw#1 such that the variable ***user_action*** is set to 0. The program needs to compare this value to the values stored in the 1st element of the sequences array. Namely, ***sequences[0]***. If the values do not match, then the game should end. Otherwise, the program should process the next button pressed.

    b. The user subsequently presses sw#3 such that the ***user_action*** is set to 2. The program needs to compare this value to the values stored in the 2nd element of the sequences array. Namely ***sequences[1]***. If the values do not match, then the game should end. Otherwise, the program should process the next button pressed.

    c. The user subsequently presses sw#3 again such that the ***user_action*** is set to 2. The program needs to compare this value to the values stored in the 3rd element of the sequences array. Namely ***sequences[2]***. If the values do not match, then the game should end. Otherwise, the program should continue to the next step:

    d. The user subsequently presses sw#2 such that the ***user_action*** is set to 1. The program needs to compare this value to the values stored in the 4th element of the sequences array. Namely ***sequences[3]***. If the values do not match, then the game should end. Otherwise, the program should continue to the next step:

    e. Generate an additional random sequence in the sequences array and increment the value of ***sequence_count*** to t. Repeat steps 15a through 15d and also check to see if the fifth button pressed matches the 5th entry in the ***sequences*** array.

16. To simplify the implementation of the process outlined in Step 15, implement a function named ***process_user_selections()*** that meets the following specifications:

    a. Returns a ***Boolean***. If the value is true, then game play should continue. If the value is false, then game play should be suspended.

    b. Accepts a single argument ***integer*** argument named ***action***.

    c. Checks to see if the user action matches the current sequence. In other words, check if ***action*** equals ***sequences[current_index]***. If the values match then increment the variable ***current_index*** by one and return ***True***. Otherwise, return ***False***.

    When implementing this step recall that the variable ***current_index*** was declared as a global variable in Step 7. It is used to track the sequence that is currently being validated. For each round we always check the first sequence and then check the subsequent sequences as they are entered by the user.

17. Revisit the code entered in Step 14 to call the function ***process_user_selections ()***. Pass the variable ***user_action*** as an argument to the function. The code should be inserted between lines 3 and 4 of the code snippet shown in Figure 3. Set the function call so you can process the return value of the function and then add the following functionality based on the return value:

    a. If ***process_user_selections ()*** returns ***True***, then print the current value of ***currentindex*** to the TFT screen so the user knows how many sequences they have entered correctly so far. This can be accomplished by adding the following code:

```
    # Display the current sequence count to the end user
```

```
        display.draw_text(115, 130, str(current_index), espresso_dolce,
                          color565(255, 255, 255), color565(255, 0, 255))
```

b. If ***process_user_selections()*** returns ***False***, then perform the following actions:

   i. Print text to the TFT screen that indicates they player has lost the game

   ii. Play a tone for 1 second at 100 Hz to the speaker.

   iii. Pause execution until the user presses the button attached to the interrupt pin.

   These three tasks can be accomplished by adding the following to the code:

```
display.draw_text(75, 100, "You lose.", espresso_dolce,
                  color565(255, 255, 255), color565(255, 0, 255))
display.draw_text(70, 120, "Play again?", espresso_dolce,
                  color565(255, 255, 255), color565(255, 0, 255))
playtone(50, 1000)
time.sleep(1)
playtone(50, 0)
while(interrupt_button.value()): pass
```

18. ***Run the sketch to verify we did not create any errors***.

19. Now we want to add the ability to see if the user got all four messages correct, and if they did, add another sequence to the sequence count. To do this, create a function called ***check_state()*** that takes no input and returns no value.

    a. Create a global Boolean named ***next_round*** and initialize it to ***False***.

    b. The first thing the function should do is check to see if the game has been won. To win the game, you must get a sequence of 20 in a row. If the ***current_index*** is 20, then display a winning message and blink the LEDs.

    c. If the game has not been won, but the ***current_index*** is equal to the ***sequence_count***, then set ***next_round*** to ***True*** because a new round needs to be started.

20. Once ***check_state*** is complete, a new option needs to be added before line 16 in Figure 2. Add an ***elif next_round:*** before the ***else*** statement the in the next line.

    a. Inside this elif, the ***sequence_count*** needs to be increased by one and a new sequence needs to be appended to the ***sequences*** array.

    b. A message needs to be displayed to the user indicating that the next round is about to start. The number of sequences should also be displayed.

    c. Wait 2 seconds to give the user time to prepare, then use a ***for*** loop and ***process_move()*** to play the next round.

    d. Set ***next_round*** to ***False***.

21. Lastly, the ***check_state()*** function needs to be run every time a ***process_user_selections()*** returns ***True***, so add a call to ***check_state()*** after the code shown in step 17a.

22. ***Run the code to verify that we did not create any errors and the game plays as expected.*** Our program should perform as follows:

    a. When the program starts or the menu key on the remote is pressed, the program should display the message WELCOME TO WILLIE SAYS to the TFT screen.

b. If the user presses the external button the program should display the message NEW GAME to the TFT screen followed by a delay of one second. Then the program should generate four random sequences, display them on the TFT, illuminate the associated LEDs, and play the associated tones.

c. After the 4 initial sequences are displayed if the user pressed the buttons on the Willie Says board in the correct sequence the sequence count should increment in the center of the TFT screen. If the user enters an incorrect sequence, then a message that indicates the user lost the game should be displayed, a tone should be displayed, and the program should pause until the external button is pressed.

d. If all 4 input sequences are correct, a message should indicate to the user that the next round will be starting with 5 sequences. This should repeat until the player gets all 20 correct.

## Submission

Upload the files ***exercise_07.ino*** to Canvas once the exercise has been completed. The code will be evaluated as follows:

1. It will be evaluated to make sure the proper coding techniques were used to implement the associated functionality.
2. It will be evaluated to ensure the associated code runs successfully.
3. It will be evaluated to ensure the associated code returns accurate and complete results.
4. It will be evaluated to ensure that the correct file names specified for each problem were uploaded to Canvas.
5. ***It will be evaluated to make sure the implementation uses correct indentation.***
6. ***It will be evaluated to determine if suitable comments have been included.***