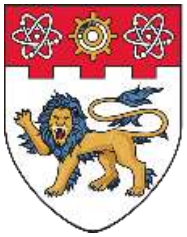


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Laboratory Manual
for
CE/CZ1103
Introduction to Computational Thinking and Programming

Practical Exercise #6:
Decomposition

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Ex. #6 – Decomposition

Learning Objectives

The manual provides information and exercises to let you use the concept of decomposition, the divide and conquer technique to break a problem into smaller and simpler parts, and together with other concepts and techniques that you have learnt in this course, develop an algorithm to implement a game.

Intended Learning Outcomes

At the end of this exercise, you should be able to

- know how to use the decomposition strategy to break a problem into a set of component parts and implements them in the form of functions, which together lead to the solution of the problem.

Equipment and accessories required

i) Raspberry Pi 3 Model B (RPi3) board with Sense HAT add-on display module/board.
ii) A USB power source to power the RPi3 board (E.g. Power Bank, Adaptor or USB port of a desktop computer).
iii) A computer (desktop PC or notebook) with Ethernet port and cable for remote access of RPi3. Software (open source) to be installed on the computer – PuTTY, VNC Viewer and WinSCP

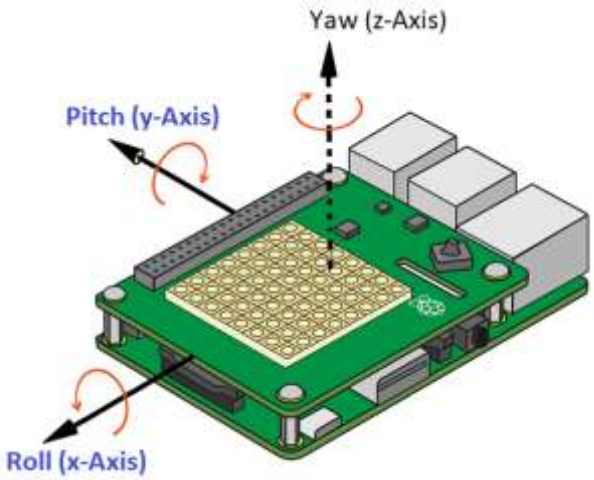
---

1. Decomposition – Divide and Conquer

When you are tasked to solve a problem of significant complexity, the approach is to first decompose the problem into smaller sub-parts that can be tested and solved independently. The solutions to the sub-parts can then be combined to solve the original problem. In this exercise, you will be guided to develop various functions for the raspberry Pi board, and combine them to implement games of various complexity.

2. Detecting Pitch and Roll using the Sense Hat

As briefly introduced in the earlier exercise (5d), the Sense HAT display board contains sensors integrated in the form of an IMU (Inertial Measurement Unit) chip, which can be used to detect the orientation of the board along the 3 axes known as Roll (x-axis), Pitch(y-axis) and Yaw (z-axis) as shown in the figure below.



In this exercise, you will only be required to detect the roll and the pitch of the board to implement a Marble game on the Sense Hat display. The Sense Hat library provides the method `get_orientation()` that can be used to detect the roll and pitch (and yaw) of the board, and returns the values in the form of a dictionary data structure with three entries

associated with the 3 keys: {'roll', 'pitch' and 'yaw'}. The following code illustrates how you can detect the roll and pitch of the board using this method (through the usual **sense** object you create when using the SenseHat module).

```
pitch = sense.get_orientation()['pitch']
roll = sense.get_orientation()['roll']
```

Coding Exercise 6a

Key in and execute the following code on the RPi board. Observe the range of values of the roll and pitch detected by the Sense Hat as you tilt the RPi board along these two axes.

```
while True:
   pitch = sense.get_orientation()['pitch']
   roll = sense.get_orientation()['roll']
   print("pitch {0} roll {1}".format(round(pitch,0), round(roll,0))
   sleep(0.05)
```

3. Two-Dimension data structure

You will now extend the program to display a 'marble' on the Sense Hat Display board, such that it will roll on the board as you tilt the RPi along the roll and pitch axes.

First you define the size of the board using a list-within-a-list data structure as follows (using black/blank colour only).

```
board = [[b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b],
         [b,b,b,b,b,b,b,b] ]
```

With this data structure, each roll of entry corresponds to the y coordinate of the board, while the entry within each roll corresponds to the individual x coordinate of the row, i.e. this is a 2-dimension description of the board. With this data structure, you can specify the position of a white 'marble' by using its y-x coordinate as follows:

```
y=2                  # y coordinate of marble
x=2                  # x coordinate of marble
board[y][x]=w        # a white marble
```

To display the marble on the board using the **set_pixels** method, you need to convert the 2-dimension data structure for the board into a 1-dimension list data structure. This can be done by applying the **sum** operation to the list (together with an empty list) as follows:

```
board_1D=sum(board,[])          # convert to 1-dimension list
print(board_1D)                 # for code debugging
sense.set_pixels(board_1D)      # display it
```

Coding Exercise 6b

Write the program to display a marble on the Sense Hat board by using the code given above.

3. Moving the marble

You will now move the marble on the board by detecting the roll and pitch of the board. The following code shows a function, **move_marble()** that can be used to compute the new coordinate of the marble based on the roll and pitch values gathered from the board.

```
# This function checks the pitch value and the x coordinate
# to determine whether to move the marble in the x-direction.
# Similarly, it checks the roll value and y coordinate to
# determine whether to move the marble in the y-direction.
def move_marble(pitch,roll,x,y):
    new_x = x    #assume no change to start with
    new_y = y    #assume no change to start with
    if 1 < pitch < 179 and x != 0:
        new_x -= 1    # move left
    elif 359 > pitch > 179 and x != 7:
        new_x += 1    # move right

    if 1 < roll < 179 and y != 7:
        new_y += 1    # move up
    elif 359 > roll > 179 and y != 0:
        new_y -= 1    # move down

    return new_x, new_y
```

This function can be called from the main body of the program as follows.

```
while True:
    pitch = sense.get_orientation()['pitch']
    roll = sense.get_orientation()['roll']
    x,y = move_marbla(pitch,roll,x,y)
    board[y][x] = w
    sense.set_pixels(sum(board,[]))
    sleep(0.05)
```

Coding Exercise 6c

- Write the program using the code given above and execute it on the RPi board to observe the operation of the program while you tilt the board along the x and y axes.
- You would have observed that while the marble is moving according to the tilt of the board, it leaves a trail of its movement along the path.
  - Analyse the program code to find a solution to this problem. (Hint: the solution consists of one additional line of code.)

4. Adding a wall

You will now add boundary walls on the board that the marble cannot cross. A simple boundary wall can be as follows, which is defined by using red colour pixels.

```
board = [[r,r,r,r,r,r,r,r],
         [r,b,b,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [r,r,r,r,r,r,r,r] ]
```

The boundary can hence be checked by determining whether the new position calculated by the `move_marble()` function is of the red colour, using a function such as the following `check_wall()` which can be called from the `move_marble()` function as shown below.

```
def check_wall(x,y,new_x,new_y):
    if board[new_y][new_x] != r:
        return new_x, new_y
    elif board[new_y][x] != r:
        return x, new_y
    elif board[y][new_x] != r
        return new_x, y
    else
        return x,y

def move_marble(pitch,roll,x,y):
    new_x = x     #assume no change to start with
    new_y = y     #assume no change to start with
      :
    new_x,new_y = check_wall(x,y,new_x,new_y)
    return new_x, new_y
```

Coding Exercise 6d

- Add the new function `check_wall()` to your program. Execute the program and observe that the marble will only move within the boundary and cannot cross the wall as you tilt the board.
- Once the program is working properly, you can modify the wall design to make it into a maze. An example is given below.

```
board = [[r,r,r,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [b,b,b,b,b,r,b,r],
         [b,r,r,b,r,r,b,r],
         [b,b,b,b,b,b,b,b],
         [b,r,b,r,r,b,b,b],
         [b,b,b,r,b,b,b,r],
         [r,r,b,b,b,r,r,r] ]
```

Run you program with your new wall design and check that it continues to operate as expected.

5. Adding a target

To complete the game, you will now add a target in your maze design, such as by using a green color to indicate its location as shown below.

```
board = [[r,r,r,b,b,b,b,r],
         [r,b,b,b,b,b,b,r],
         [b,b,b,b,g,r,b,r],
         [b,r,r,b,r,r,b,r],
         [b,b,b,b,b,b,b,b],
         [b,r,b,r,r,b,b,b],
         [b,b,b,r,b,b,b,r],
         [r,r,b,b,b,r,r,r] ]
```

The objective of the game is to guide the marble to capture this target, and when successful, display a message or a picture.

Coding Exercise 6e

Modify your program and add the necessary code that will terminate the game once the target is captured by the Marble.

```
while not game_over:
    pitch = sense.get_orientation()['pitch']
    roll = sense.get_orientation()['roll']
    x,y = move_marble(pitch,roll,x,y)
      :                # add your code here

sense.show_message('yay!!')
```

6. Challenge (Optional)

Add a function to your program that will change the position of the target every 10 seconds. The idea is the player is given only 10 seconds to capture the target at a location before it moves away.