Report on
Python for Scientific Computation and Control

CTU in Prague, 2017

Arne Hulsdunker

Simon Memory Game

1. Introduction

This project is about creating a simple ‚Simon Memory Game'. The original game was invented by Ralph H. Baer and Howard J. Morrison (Baer a Morrison 1980) and launched in 1978 in the USA.

In this project, the game is implemented in python in an object orientated manner and visualized with the help of the Tkinter toolkit.

2. What is the 'Simon Memory Game' about?

In general, the game is about memorizing a specific order of colors and sounds that is created by the computer and becomes longer in every round so that the difficulty increases.

Figure 1: original simon game (ANON. 2016)

As you can see on picture (Figure 1:) the game mainly consists of four buttons colored in blue, red, yellow and green. When you start the game, one of the buttons will light up and a specific sound will be played. The task of the player is to copy the behavior of the computer by pushing the same button. After the correct copy of the player the computer adds one more color and corresponding sound to the order. The task of the player is again to copy this order of now two colors.
The list of colors increases each round until the player makes a mistake in copying the right order. In this case, the game is over and the player has to start the game again.

3. The GUI of the game

In the Figure 2 you can see the four colored buttons described in chapter What is the 'Simon Memory Game' about?. These are the buttons the player has to press to copy the randomly created order of colors. Furthermore there are two buttons to start and to restart the game, while the text field shows the current score of the player.
With the two checkboxes in the middle of the figure the player can activate or deactivate special additional functions like turning the sound on or off and including the stroop effect to the game. More information about the stroop effect are in chapter (…).

To start the game, the player has to press the start button. Failing in copying the correct order of colors or clicking on the reset button will lead to the appearance of a message box, informing that the game is over and showing the current score of the player.

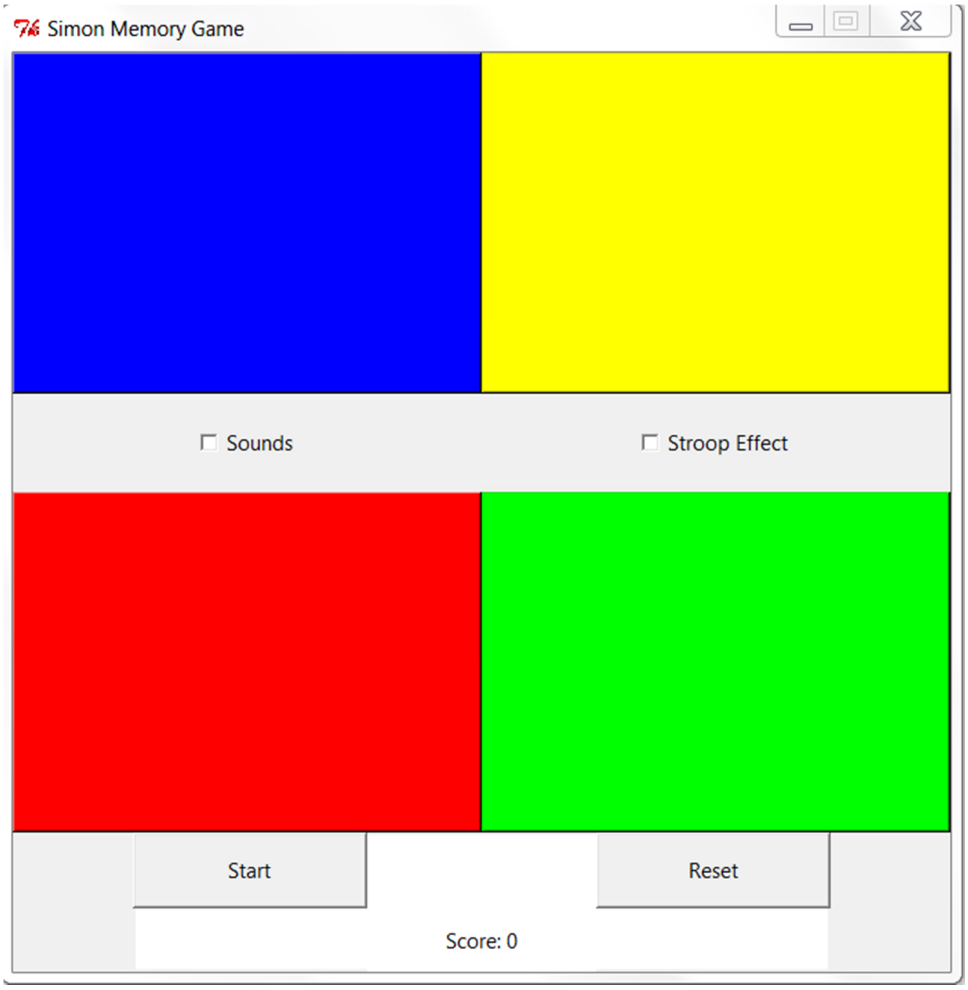

Figure 2: GUI of the simon memory game implemented in python

4. The code

4.1. Import of necessary packages

```
1  import Tkinter
2  import random
3  import time
4  import tkMessageBox
5  import winsound
```

The first lines of the code load all the necessary packages for the program, following by the start of the class.

4.2. The initial function

```
def __init__(self):

    self.colors = ['blue', 'red', 'yellow', 'green']        # list of colors
    self.winsounds = ['c1', 'e1', 'a1', 'c2']               # list of sounds in .wav format

    self.colorOrder = []        # fill in with the numeric equivalents of the colors randomly
    self.score = 0              # counter for the successfully completed rounds
    self.listOfPressedButtons = []      # fill in with the numeric equivalents of the colors after button press

    self.mainFrame = Tkinter.Tk()                   # create the top level figure called mainFrame
    self.mainFrame.title('Simon Memory Game')       # set the title of the figure to "Simon Memory Game"
    self.mainFrame.resizable(width=False, height=False) # fix the size of the figure

    self.buttons = ['blue', 'red', 'yellow', 'green']   # create the four colored buttons with a for loop
    for n in range(0, len(self.buttons), 1):
        self.buttons[n] = Tkinter.Button(self.mainFrame,  bg=self.colors[n], font=1, height=10, width=30, command=lambda color=n:(self.push
        self.buttons[n].grid(row=n*2 % 4, column=n*2 / 4)

    self.lowerField = Tkinter.Canvas(self.mainFrame, bg='white', height=10)     # field for additional buttons or textfields
    self.lowerField.grid(row=3, columnspan=2)

    self.startButton = Tkinter.Button(self.lowerField, height=2, width=20, text='Start', command=lambda: self.start())  # button to start t
    self.startButton.grid(row=0, column=0)

    self.scoreField = Tkinter.Label(self.lowerField, bg='white', height=2, width=20, text='Score: '+str(self.score))   # textfield to sho
    self.scoreField.grid(row=1, column=1)

    self.resetButton = Tkinter.Button(self.lowerField, height=2, width=20, text='Reset', command=lambda: self.reset())  # button to reset t
    self.resetButton.grid(row=0, column=2)

    self.varCheckboxSounds = Tkinter.IntVar()
    self.checkboxSounds = Tkinter.Checkbutton(self.mainFrame, text="Sounds", variable=self.varCheckboxSounds, height=3, width=20)   # check
    self.checkboxSounds.grid(row=1, column=0)

    self.varCheckboxStroop = Tkinter.IntVar()
    self.checkboxStroop = Tkinter.Checkbutton(self.mainFrame, text="Stroop Effect", variable=self.varCheckboxStroop, height=3, width=20)
    self.checkboxStroop.grid(row=1, column=1)
```

The initial function consists of all the different buttons, text fields and checkboxes, which are displayed in the figure. They are given their properties and they are arranged in a grid. Moreover the attributes of the class are created with the help of the initial function when executing the program.

4.3. pushButton function

```
def pushButton(self, color):                                    # the colors are here the numbers from 0 to 3
    if self.varCheckboxSounds.get() == 1:                       # check, if the sounds are activated
        winsound.PlaySound(self.winsounds[color], winsound.SND_NODEFAULT)   # play sound
    self.listOfPressedButtons.append(color)                     # extend the listOfPressedButtons
    for counter in range(0, len(self.listOfPressedButtons), 1):     # going through the listOfPressedButtons
        if self.listOfPressedButtons[counter] == self.colorOrder[counter]:   # if the button color is equal to the current color at every entry
            print 'right button'

        else:
            print 'wrong input'
            self.reset()            # call the reset method
            break

    if len(self.colorOrder) == len(self.listOfPressedButtons):  # if the length of the lists are equal = all colors are correct
        self.updateScore()                  # call method to update the score
        self.newColor()                     # call method to randomly add a new color
        self.listOfPressedButtons = []      # empty the listOfPressedButtons
        time.sleep(0.2)                     # wait for some time
        self.showColorOrder()               # call the method to show the new color order in the figure
```

The initial function is followed by a function called *pushButton*. It is the function that is called if one of the colored buttons is clicked. This function checks if the checkbox for the sound is set and if this is

true plays the specific tone. If the checkbox is cleared the program directly jumps to line 52, where the color of the pushed button is added to the list *listOfPressedButtons*.

The following for-loop compares the button presses with the color order, randomly created by the computer. If the player succeeds, the computer will clear the *listOfPressedButtons* and call different methods to update the score, add a new color to the list *colorOrder* and display the new order in the figure.

If the player fails copying the right order, the program will call the reset function and break out of the loop.

4.4. showColorOrder function

```
70  def showColorOrder(self):
71      for numericEntries in self.colorOrder:                      # for all numeric color entries (between 0 and 3) of the list "colorOrder"
72          self.buttons[numericEntries].flash()                    # flash of the proper button
73          if self.varCheckboxSounds.get() == 1:                   # check, if the sounds are activated
74              winsound.PlaySound(self.winsounds[numericEntries], winsound.SND_NODEFAULT)  # play sound
75          if self.varCheckboxSounds.get() == 0:
76              time.sleep(0.2)                                         # wait for some time
```

The *showColorOrder* method displays the current color order in the figure by flashing one button after another. If the checkbox *sounds* is set, the player will additionally hear the specific tone to each button.

4.5. reset function

```
80      def reset(self):
81          tkMessageBox.showwarning('Game Over', 'Game over! \n'                           # show a message box
82                                                'Your Score: ' + str(self.score) + '\n'
83                                                'Please click on start to restart the game')
84          self.colorOrder = []            # empty the list of the colors
85          self.score = 0                  # set the score to zero
86          self.scoreField.config(text='Score: ' + str(self.score))    # update the new score in the text field
```

The *reset* method is called either when the player makes a mistake or if the *reset* button is pressed. In both cases a message box appears telling that the game is over and showing the current score.

4.6. start function

```
89      def start(self):
90          self.newColor()                     # call the method to randomly add a new color
91          self.showColorOrder()               # call the method to show the new color order in the figure
92          self.listOfPressedButtons = []      # empty the listOfPressedButtons
93          print 'start'
```

To start or restart the game, the player has to push the *start* button. The computer will randomly choose the first color and flash the appropriate button.

4.7. newColor function

```
96      def newColor(self):
97          self.nextColor = random.randint(0, 3)       # randomly generate a number between 0 and 3 (--> reference to colors)
98          self.colorOrder.append(self.nextColor)      # add the new color to the list "colorOrder"
99          self.stroopEffect()                         # call the method for the stroop effect
```

When the method *newColor* is called, it randomly chooses a new color and adds it to the list *colorOrder*. It also calls the *stroopEffect* function.

4.8. updateScore function

```
102     def updateScore(self):
103         self.score += 1                                             # increment the score by one
104         self.scoreField.config(text='Score: '+str(self.score))      # update the textfield
```

The *updateScore* function simply increments the value of the variable *score* by one and updates the text string of the text field in the figure.

4.9. stroopEffect function

```
107     def stroopEffect(self):
108         if self.varCheckboxStroop.get() == 1:               # if the checkbox is set
109             for n in range(0, len(self.buttons), 1):
110                 self.buttons[n].config(bg='black', fg=self.colors[n], text=random.choice(self.colors))
111         if self.varCheckboxStroop.get() == 0:               # if the checkbox is cleared
112             for n in range(0, len(self.buttons), 1):
113                 self.buttons[n].config(bg=self.colors[n], text='')                          # set the
```

The last method of the class is called *stroopEffect*. It checks if the corresponding checkbox is set or cleared. Depending on the state of the checkbox, strings of the four colors will be shown in the middle of the colored buttons. More information about the stroop effect are given in the next chapter The stroop effect

5. The stroop effect

The stroop effect is named after JohnRidley Stroop, who worked as a psychologist in America. He published a test showing the effect of the interference of two different cortex pathways on the reaction time.

The human brain is fast in processing simple data like seeing squares of colors. But if the color of printed names of colors has to be processed, the processing time of the brain increases significantly.

In this project, the player can change the colored squares to color names, painted in the same color like the squares and see, if there is a difference in the memory capacity.

References:

ANON., 2016. Simon *(game)* [online]. [vid. 2017-01-19]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Simon_(game)&oldid=751021018

BAER, Ralph H. a Howard J. MORRISON, 1980. Microcomputer controlled game [online]. US4207087 (A). [vid. 2017-01-19]. 10. červen 1980. Dostupné z: https://worldwide.espacenet.com/publicationDetails/biblio?FT=D&date=19800610&DB=&locale=en_EP&CC=US&NR=4207087A&KC=A&ND=1

Contents