

## Short Description

The Memory game has appeared in many different contexts, either as a computer game or as a card game that uses a partial or full deck of playing cards. The player tries to find two matching tiles by selecting tiles from a rectangular grid. This game also appeared as a popular American TV show called Concentration. We will create a single person game that tracks the score of the player as the time taken to complete the game, where a lower score is better. Multiple players can take turns playing the game and compete by comparing their scores.

## Learning Outcomes

- Practice basic programming skills
- Discover new language features<sup>1</sup>
- Learn how to organize your software according to classes and functions
- Use pygame to create graphical game programs
- Practice programming with concepts:
  - nested collections, n-dimensional sequences, files

## Detailed Explanation and Things To Do

- Study the following video to understand how the game Memory is supposed to look like and work:
  - [Video](#) of the "Memory" game with a second game starting after the first game ends. Clicking on an exposed tile or clicking in the right (black) panel has no effect.
- Design and write code for a program that implements Memory as shown in the above video.

---

<sup>1</sup> To create this game, you may need to use a few programming language features that have not been used yet in class. Discovering new language features and how to use them is an integral part of problem-solving in computing science and an essential skill that you should learn. Think about what you need to do, search the web for python3 programming examples, and/or use the [Python documentation](#) and [pygame documentation](#) to help you find the programming constructs that you need. If you get stuck, ask your TA for help/hints about the programming constructs you need to use.

- Your code must be based on our final [pre-poke framework](#), which imports pygame, and implements the game framework in its own class. You are not allowed to import any additional modules, other than modules that are in the Python Standard Library, if needed.
- In addition to the Game class, your code must contain a Tile class.
- You may add new methods to your Game class as needed, but you must also use all methods supplied in the pre-poke framework.
- The draw method of your Tile class may only be called from the draw method of your Game class.
- The only call to the draw method of your Game class is within the Game's play method.
- You will need to use `pygame.image.load()` to create an image (Surface) object by reading it from a file.
- There are 8 images that must be placed randomly into 16 tiles. Here is a way to create a list of 16 images that can be used in the tiles.
  - Start with an empty list of images in the Game class.
  - For each of the 8 image files (image1.bmp - image8.bmp), read the image and append it to the list of images.
  - Concatenate the images list with itself to obtain a list of 16 images - each image is in the list twice.
  - Call a function to randomize the images in the list (the `random.shuffle` function).
  - When you create a Tile object, assign it both the "hidden tile" image ('image0.bmp') and one of the other images from the images list, so that a Tile can draw itself either as hidden or as exposed.
- Each Tile object should know whether it is currently exposed or not so that it can draw itself properly when it is in its hidden or exposed state. To achieve this, there must be a corresponding instance attribute.
- Since the Game object needs to tell a tile to expose itself, there must be a method in the Tile class that, when called, changes the tile's state to exposed.
- You will need to write a method to check whether two tiles contain the same image. This method should be in your Tile class and should take a second tile (the "other" tile) as an argument.

- You must follow the code quality standards outlined in the [Software Quality Requirements for code with classes](#).

## Hints

- A tile in memory, like a tile in Tic Tac Toe, has two states ("exposed" and "hidden"), depending on which it should draw itself with the associated image or the "hidden tile" image.
- Start by implementing a version that only displays the grid with "exposed" images shown in each tile.
- Then, extend the functionality by displaying the "hidden tile" image in each tile, and exposing the image associated with a tile when the player clicks a mouse button inside a tile. When the player clicks on a tile, the tile should become permanently exposed.
- Then, extend the functionality by adding the score. The game should end (the score should stop changing) when all 16 tiles have been exposed (in any order).
- Then, extend the functionality by implementing the correct behaviour of the tiles when clicked. That means, when a tile is selected it should be exposed until a second tile is selected. If the image on the second selected tile is the same as the image on the first selected tile, then both tiles should stay permanently exposed. If the images do not match, both tiles should become hidden after a short delay.
- Note that a single Tile object cannot know whether there is another tile currently selected or not. Whether a single tile is currently selected, or whether two tiles are currently selected represents states of the Game, not the Tile class. How to record whether there is currently no selected tile, one selected tile, or two selected tiles, and how to respond when two tiles have been selected (i.e., what to do if they match, and what to do if they do not match) must be implemented in the Game class.

## Resources

- Your code must be based on our [pre-poke framework](#).
- Download copies of [the images](#) that will be used for your game.

## Submission Information

- Please submit the solution for this mini-project by the due date for this project. For submission purpose, the file with your code should be named:

**memory.py**

## COMPLETE THE FOLLOWING SOLUTION:

# TTT Version 1

import pygame

# User-defined functions

def main():

# initialize all pygame modules (some need initialization)

pygame.init()

# create a pygame display window

pygame.display.set\_mode((500, 400))

# set the title of the display window

pygame.display.set\_caption('Memory')

# get the display surface

w\_surface = pygame.display.get\_surface()

# create a game object

game = Game(w\_surface)

# start the main game loop by calling the play method on the game object

game.play()

# quit pygame and clean up the pygame window

pygame.quit()

# User-defined classes

class Game:

# An object in this class represents a complete game.

def \_\_init\_\_(self, surface):

# Initialize a Game.

# - self is the Game to initialize

# - surface is the display window surface object

# === objects that are part of every game that we will discuss

self.surface = surface

self.bg\_color = pygame.Color('black')

self.FPS = 60

self.game\_Clock = pygame.time.Clock()

self.close\_clicked = False

```

self.continue_game = True

# === game specific objects
self.board_size = 4
self.image_list=[]
self.load_images
self.board = [] # will be represented by a list of lists
self.create_board()

def load_images(self):
    #do the following steps 8 times image1-image8
    image=pygame.image.load("image1.bmp")
    self.image_list.append(image)

def create_board(self):
    #width = self.surface.get_width()//self.board_size
    #height = self.surface.get_height()//self.board_size
    for row_index in range(0,self.board_size):
        row = []
        for col_index in range(0,self.board_size):
            image=self.image_list[0] #zero bcz here we hav only 1 image
            width=image.get_width()
            height=image.get_height()
            x = col_index *width
            y = row_index * height
            tile = Tile(x,y,width,height,image,self.surface)
            row.append(tile)
        self.board.append(row)

def play(self):
    # Play the game until the player presses the close box.
    # - self is the Game that should be continued or not.

    while not self.close_clicked: # until player clicks close box
        # play frame
        self.handle_events()
        self.draw()
        if self.continue_game:
            self.update()
            self.decide_continue()
        self.game_Clock.tick(self.FPS) # run at most with FPS Frames Per Second

def handle_events(self):

```

```
# Handle each user event by changing the game state appropriately.
# - self is the Game whose events will be handled
```

```
events = pygame.event.get()
for event in events:
    if event.type == pygame.QUIT:
        self.close_clicked = True
```

```
def draw(self):
    # Draw all game objects.
    # - self is the Game to draw
```

```
self.surface.fill(self.bg_color) # clear the display surface first
# draw the board
for row in self.board:
    for tile in row:
        tile.draw()
pygame.display.update() # make the updated surface appear on the display
```

```
def update(self):
    # Update the game objects for the next frame.
    # - self is the Game to update

    pass
```

```
def decide_continue(self):
    # Check and remember if the game should continue
    # - self is the Game to check

    pass
```

```
class Tile:
    # A class is a blueprint --- > Properties and behavior
```

```
def __init__(self,x,y,width,height,image,surface):
    self.rect = pygame.Rect(x,y,width,height)
    self.color = pygame.Color('white')
    self.border_width= 3
    self.hidden_image=pygame.image.load("image0.bmp")
    self.content = image
    self.hidden=True
    self.surface = surface
```

```
def draw(self):
```

```

# draw the coordinates of each Tile objects
#string = str(self.rect.x) + ',' + str(self.rect.y)
#font = pygame.font.SysFont("",40)
#text_box = font.render(string,True, self.color)
#location = (self.rect.x,self.rect.y)
#self.surface.blit(text_box,location)
location=(self.rect.x,self.rect.y)
if self.hidden==False:
    self.surface.blit(self.hidden_image,location)
else:
    self.surface.blit(self.content,location)
pygame.draw.rect(self.surface,self.color,self.rect, self.border_width)

```

```

#self.draw_content()
def draw_content(self):
    font = pygame.font.SysFont("",133) # height of the surface is 400 //3 = 133
    text_box = font.render(self.content,True,self.color)
    # text_box is a pygame.Surface object - get the rectangle from the surface
    rect1 = text_box.get_rect()
    #rect1 <----> self.rect
    rect1.center = self.rect.center
    location = (rect1.x,rect1.y)
    self.surface.blit(text_box,location)

```

```

main()

```