Game: Memory

Version: 1

**Description**

**When the game is started, a 500px by 400px window opens, with a black background with title ‘Memory’. There is a 4x4 grid on the left side. The grid is made up of tiles. Each tile is 100px by 100px** and has a blue background with a red question mark in the middle. There is a (40px white font black background) text timer in the top right, increasing with every second since the game started. **There is a 100px wide black bar on the left side of the screen unoccupied by tiles. Each tile is set to one of 16 images randomly positioned. However, there are only 8 different images, and there are two copies of each image.** When a tile is clicked, the image changes to the image it is set to. When another tile is clicked, it also changes to the image it is set to, and then the game checks to see if the tiles are the same image. When the player selects two images that are matching, they both turn over and stay flipped over for the remainder of the game. If the two boxes the player selects does not contain matching images the tiles stay flipped for two seconds, then change back to the question mark image. **Clicking on a tile that has already been flipped over will do nothing.** The game ends and the timer stops increasing when all tiles have been matched. **Clicking a tile at this point will do nothing. When the X is clicked, the game ends and the window closes.**

# Exemplar Graphics-4

# This is an example program that contains graphics, using

# modules pygame and uaio and responds to a close window event.

# It contains these kinds of statements: expression, assignment,

# import, function definition, while, return, class definition

# It contains these kinds of expressions: identifier, literal,

# attribute reference, function call, binary operator, expression

# list

# It uses these types:

# str, int, float, bool, NoneType, function, module, tuple

# pygame.Surface, pygame.Color, pygame.Rect, Game

import pygame

import time

import random

from pygame.locals import \*

# User-defined functions

def main():

surface = create\_window()

game = Game(surface)

game.play()

pygame.quit()

def create\_window():

# Open a window on the display and return its Surface

title = 'Memory'

size = (500, 400)

pygame.init()

surface = pygame.display.set\_mode(size, 0, 0)

pygame.display.set\_caption(title)

return surface

# 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 window's pygame.Surface object

self.surface = surface

Tile.set\_surface(surface)

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

self.pause\_time = 0.04 # smaller is faster game

self.close\_clicked = False

self.continue\_game = True

self.tiles = []

self.unclicked\_image = pygame.image.load('question.bmp')

self.image\_list = []

for index in range(0, 8):

image = pygame.image.load('image'+str(index)+'.bmp')

self.image\_list.append(image)

self.image\_list = self.image\_list + self.image\_list

self.create\_grid()

def create\_grid(self):

tile\_width = self.surface.get\_width() // 5

tile\_height = self.surface.get\_height() // 4

for row in range(0, 4):

row\_list = []

for column in range(0, 4):

x\_coord = column \* tile\_width

y\_coord = row \* tile\_height

tile\_type = random.randint(0, len(self.image\_list)-1)

image = self.image\_list[tile\_type]

self.image\_list.remove(image)

self.value = tile\_type // 2

tile = Tile(x\_coord, y\_coord, tile\_width, tile\_height, image, self.value)

row\_list.append(tile)

self.tiles.append(row\_list)

def play(self):

# Play the game until the player presses the close box.

# - self is the Game that should be continued or not.

self.draw()

while not self.close\_clicked: # until player clicks close box

# play frame

self.handle\_event()

if self.continue\_game:

self.update()

self.decide\_continue()

self.draw()

time.sleep(self.pause\_time) # set game velocity by pausing

def handle\_event(self):

# Handle each user event by changing the game state

# appropriately.

# - self is the Game whose events will be handled.

event = pygame.event.poll()

if event.type == QUIT:

self.close\_clicked = True

def draw(self):

# Draw all game objects.

# - self is the Game to draw

self.surface.fill(self.bg\_color)

for row in self.tiles:

for tile in row:

tile.draw()

pygame.display.update()

def update(self):

# Update the game objects.

# - 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:

@classmethod

def set\_surface(cls, surface):

cls.surface = surface

border\_width = 3

fg\_color = pygame.Color('black')

def \_\_init\_\_(self, x, y, width, height, image, value):

self.rectangle = Rect(x, y, width, height)

self.image = image

self.value = value

def draw(self):

# Draw the tile

pygame.draw.rect(Tile.surface, Tile.fg\_color, self.rectangle, Tile.border\_width)

# Draw the image

image\_x = self.rectangle.left + self.border\_width

image\_y = self.rectangle.top + self.border\_width

self.surface.blit(self.image, (image\_x, image\_y))

main()