OZONE

Programmer's Guide

CS 11 AC2-M2-HRU
MACHINE PROBLEM 2

2018-04225	Aldemita, Marieve Gillian L.
2018-00866	Bassig, Lance Raphael D.
2018-01306	Ceniza, Leana Eunice G.
2018-00615	Crisostomo, Coleen Anne F.

Github Page: *https://github.com/lgceniza/Odd1Out*

OZONE

Ozone is a game where you have to be quick and meticulous! A grid of icons will be displayed, and it is up to you to find the different negative panda, or the different happy shiba! Can you find the odd one out?

This game needs the player to sharpen their senses to zone in on the odd one out in the board. The player may opt to choose between three difficulties: easy, medium, and hard. Each level differs only with the play time allotted for the player. (HINT: HARD MODE IS HARD.)

This game tracks the name and score of the previous player in the game, so you'll know what and who to beat.

This game is comprised of four Python (.py) files: **game.py**, which contains the engine and also runs the game; **elements.py**, which contains classes that make important game objects; **interface.py**, which contains the interface elements; and **text_input.py**, which handles text input for the end of the game. This game also uses media files which also contribute to the gameplay.

# how the program is designed # how it works #

game.py

This is the main file for running the game. This script imports four modules: **elements.py**, **interface.py**, **text_input.py**, **random.py**, and also imports the **pyglet** library. This file contains the game engine and directly incorporates that into the program.

Methods:

Play()

This function creates the title screen whenever needed.

HowTo()

This function creates the how to play screen whenever needed.

Difficulty()

This function creates the ‘select difficulty’ screen whenever needed.

Confirm(*dependentstring*)

This function creates the 'confirm difficulty' screen whenever needed.

PARAMETERS

dependentstring : str

the game mode choice to be confirmed by the player

YourScore(*dt*)

This function creates the ‘your score’ screen whenever needed. It also resets the game, initializing all elements all over again and saves the score.

Scoreboard()

This function creates the scoreboard screen. It displays up to three previous scores. If the number of scores is less than three, then it only shows scores that are present.

timer_deplete(*dt*)

This function creates the timer in each level of the game. It accepts a float as an argument which is the rate of function calls.

The four following functions make up the game loop, which is the highlight of the program. Here is the algorithm of the game loop:

1. Pick a set of tiles from a set of sets of tiles, and then pick a tile from the set: this will be the common tile. Make a list filled with copies of this common tile.
2. From the same set, pick a different tile. Append this to the list filled with common tiles. This will be the odd tile. Shuffle the list. Return both tiles to their set to be used again.
3. Make a dictionary wherein the common tile has a value of 0, and the odd tile has a value of 1. This will be used to check, upon mouse click, if the clicked image is the odd one out.
4. From the list of tiles, make clickable buttons and append each of them to a new list. For each button in the list, assign an x- and y- coordinate: this will be the board.
5. For each tile in the board, check if that tile is clicked by the user, and is the odd one out. If both conditions are true, then one point will be added the their score, and the board is initialized again. The tiles may or may not change, but the coordinates of the odd one out most certainly will.
6. Number 5 is repeated until the timer is depleted, at which point all the tiles in the board will be made invisible.

tileset_pick(*list*)

This function accepts a list as an argument which contains the set of images to be used as tiles in the game board. It returns a new list of random tiles containing the images it used to create the game board and a dictionary containing only 1s and 0s as values, where the keys with a value 0 are common tiles and the key with the value 1 is the odd one out.

board_create(*list*)

This function creates the game board and accepts a list as an argument. The list needed will be the list containing the image names to be made into buttons created by the tileset_pick() function. It also initializes each game button with the x- and y- coordinates. It returns a list of button objects with their corresponding sprites and positions.

initialize()

This function calls the function tileset_pick() and board_create() to make one game screen and to initialize the odd one out checker. It returns a list of button objects with their corresponding sprites and positions, another list of images used to create the game board, and lastly a dictionary containing only 1s and 0s as values, where the keys with a value 0 are common tiles and the key with the value 1 is the odd one out.

gameloop(*x, y*)

This function runs the game loop. It accepts integers as parameters, x being the horizontal position of the cursor, and y being the vertical position of the cursor. With this function, the game runs recursively until the in-game timer runs out. It checks whether or not the player has clicked on the correct odd tile, and then tallies the score. After each correct answer, this function re-initializes the game buttons on the board, their corresponding images, and the checker that comes with it. It changes the sprites on the board and the coordinates of the odd tile, and then calls itself.

pyglet.gl.glClearColor (**interface.bgcolor*)

This specifies the background color of the defined game window.

music_player.play ()

This function starts the background music loop.

Events:

on_mouse_motion (*x, y, dx, dy*)

This event is generated whenever the mouse cursor moves. This is mainly used for displaying button hover state. It accepts the position and the change in position of the mouse cursor.

PARAMETERS

x, y : int

current position of the mouse cursor. x is the horizontal position of the cursor on the screen, while y is the vertical position.

dx, dy : int

change in position of the mouse cursor. dx is the change in the horizontal position of the cursor on the screen, while dy is the change in vertical position.

on_mouse_press (*x, y, button, modifiers*)

This event is generated whenever the mouse button is pressed. This is mainly used for displaying pressed button state. It accepts the position of the cursor, the pressed mouse button, and the modifiers for any pressed button. In this program, the mouse button pressed and the modifiers do not matter.

PARAMETERS

x, y : int

current position of the mouse cursor. x is the horizontal position of the cursor on the screen, while y is the vertical position.

on_mouse_release (*x, y, button, modifiers*)

This event is generated whenever the mouse button is released. This is mainly used for displaying unpressed button state, and for calling functions to clear the window to draw the next game scene. As in the on_mouse_press event, this accepts the position of the cursor, the pressed mouse button, and the modifiers. The mouse button pressed and the modifiers also do not matter.

PARAMETERS

x, y : int

current position of the mouse cursor. x is the horizontal position of the cursor on the screen, while y is the vertical position.

on_draw ()

This event is generated whenever the window is drawn. This draws all the buttons, labels, and sprites from the moment the program begins running.

Attributes:

window : window obj

This defines the window object with a width and height.

gametilebatch : batch obj

This defines the batch where the game tiles are to be collected. This batch is then drawn at the moment the game timer starts.

board, random_tiles, check_tile : list, list, dict

This defines the game board to be drawn, and the list random_tiles and the dictionary to check the common tiles and the odd tile.

scene : str
mode : str
game_start : bool
score : int

These attributes define variables at the start of the program, to be used for the game. These are necessary to define so that essential game values, such as the score and the game tiles, are newly initialized every time a game starts.

During beta testing, mode used to determine the size of the game grid and amount of images to be displayed, but due to multiple bugs, this feature was cancelled.

sound : media obj
looper : media obj
music_player : player obj

These attributes are necessary to set up the background music that will be looping while the program is running.

elements.py

This module contains classes which makes the most important elements that are essential to make the game: the buttons, the game tiles, and the timer. This module requires **pyglet** to be installed and uses the pyglet library.

Classes:

Button

This class creates off-game button objects. These objects are assigned functions which are called to represent the animation of the buttons upon fulfillment of an event.

Attributes:

name : str

name of the button image file. This is used to fetch the filename of the image.

x, y : int

position of the button relative to its lower left corner

width, height : int

dimensions of the button. These are used to determine the coordinates over which the button is drawn.

Methods:

__init__ (*name, x, y, batch*)

This initializes the name and position of the button. The name is used to fetch the unpressed and hovered images to be interchangeably placed into a sprite, which will be drawn on the window. It also sets the visibility for the buttons at the start of the program--making visible only those buttons which are immediately placed in the title screen. Finally, it attaches all buttons made into a batch to be drawn all at once at the beginning of the program.

when_pressed ()

This changes the appearance of the button sprite when it is pressed. The hue of the sprite is turned into a darker blue.

when_not_pressed ()

This method changes the sprite back to its neutral appearance--the normal, unadulterated unpressed image of the button with no hue changes.

when_hovered (*xpos, ypos*)

This method checks whether or not the cursor is hovering over the button. This is used in mouse click events to make sure that cursor was meant to press the button before it performs the button's assigned functionalities. The button's properties: its position, width, and height, are used in running this function.

PARAMETERS
xpos, ypos : int
current position of the cursor in the window

RETURNS
boolean
a truth value of whether or not the cursor is hovering over the button

when_hovering (*xpos, ypos*)

This method makes use of the when_hovered method to perform mouse motion events. When the mouse cursor is hovering over a button, this method changes the button's appearance to that of its hover state.

PARAMETERS
xpos, ypos : int
current position of the cursor in the window

button_show ()

This method changes the visibility of the button. This makes the button appear in the window. This is used for setting the game screen in each scene.

button_clear ()

This method changes the visibility of the button. This makes the button disappear from the window. This is used for setting the game screen in each scene.

GameButton

This class creates in-game button objects. Unlike the objects from the Button class, this class does not make use of any methods for animation. This class is used in making clickable game tiles, and so have use only for methods which determine whether or not the cursor is hovering over the button.

Attributes:

name : str

name of the button image file. This is used to fetch the filename of the image.

batch : obj

batch object to which the button will be appended. Doing this helps the performance in sprite rendering, since the nature of this game is to draw multiple sprites per screen and change them immediately upon clicking the correct button.

width, height : int

dimensions of the button. These are used to determine the coordinates over which the button is drawn.

Methods:

__init__ (*name, batch*)

This initializes the name and the batch to which the button will belong. The name is used to fetch the game tile image to be placed into a sprite, which will be drawn on the window. It also sets the visibility for the buttons at the start of the program--making them invisible before the game play scene begins, and also after it ends upon depletion of the timer.

when_hovered (*x, y, xpos, ypos*)

This method checks whether or not the cursor is hovering over the game tile. This is used in mouse click events to make sure that cursor was meant to press that particular game tile before it performs the next events, which hinges on if that game tile is the odd tile or not. The button’s properties: its width and height, are used in running this function.

PARAMETERS
x, y : int

position of the game tile. These properties are initialized outside of this class, for the positions of the tiles are to be randomized before they are given coordinates.

xpos, ypos : int

current position of the cursor

RETURNS

boolean

a truth value of whether or not the cursor is hovering over the button

button_show ()

This method changes the visibility of the button. This makes the button appear in the window. This is used for setting the game screen in each scene.

button_clear ()

This method changes the visibility of the button. This makes the button disappear from the window. This is used for setting the game screen in each scene.

GameTimer

This class creates the in-game timer. It contains functions which initializes the starting minute or second, which depletes the timer, and which resets it.

Attributes:

minutes : int

initial minute mark of the timer. The minute count of when the timer begins.

second : int

Initial second mark of the timer. The seconds count of when the timer begins.

start : str

initial timer string. The string that will initially be displayed before the timer will begin to deplete.

Methods:

__init__ (*minute, second*)

This initializes the duration of the timer in minutes and seconds.

RunMinutes ()

This decreases the minute mark by 1 if the minute mark is greater than 0.

RunSeconds ()

This decreases the second mark by 1. If there are 0 seconds left and more than 0 minutes left, then it calls RunMinutes and then sets the seconds mark to 59.

TimerReset ()

This restores the timer to its initial state.

interface.py

This module contains the elements which are necessary to create the interface of the game: text labels, images, colors, and button objects. This module imports the **elements.py** module, and requires **pyglet** to be installed.

Attributes:

bgcolor : tuple
width, height = tuple of int, int

> These attributes specify the background color of the window, as well as its dimensions.

title : resource img
ozone : sprite

> These define the image that will be displayed in the title screen, including its position.

click_sound : resource media, audio wav
timeout_sound : resource media, audio wav
correct_sound : resource media, audio wav

> These define the sound files that will be used and played throughout the game.

Cats, Dogs, Octopi, Raccoons, Pandas : lists of str
tileset_list = [Cats, Dogs, Octopi, Pandas, Raccoons]

> These define the filenames of the game tile images that can be used.

watermark : resource img
watermark_sprite : sprite

> These define the image that will be displayed during the game play screen, to show credit to the authors of the game tile images used.

instructions : list

> This is used in the batch_fill and batch_clear methods to display or remove the paragraph in the 'How To Play' screen. The list will be filled with label objects to draw on the window, and the text from these label objects comes from the opened file.

labelbatch : batch obj
howtoplay_label : label obj
timelabel : label obj
selectdiff_label : label obj
confirm_label : label obj
score_display : label obj
scoreboard_label : label obj
name_label : label obj
score_label : label obj
scoreboard_label : label obj

These label objects are defined to be drawn from the main game script. Each label is a caption to their appropriate game screens, indicating the current scene of the game, if needed be. Each label has programmer-defined text, font styles, and positions. They are attached to the batch object to be drawn all at once.

scorelabelbatch : batch obj
one : label obj
two : label obj
three : label obj
name1 : label obj
name2 : label obj
name3 : label obj
score1 : label obj
score2 : label obj
score3 : label obj

These label objects are defined to be drawn from the main game script. Each label displays information about the players' names and their scores. Each label has programmer-defined text, font styles, and positions. They are attached to the batch object to be drawn all at once.

buttonbatch : batch obj
playbutton : button obj
nextpage : button obj
nextchoice : button obj
previouschoice : button obj
easydiff : button obj
mediumdiff : button obj
harddiff : button obj
yeschoice : button obj
nochoice : button obj
diffselect : button obj
exitbutton : button obj

These button objects are defined to be drawn from the main game script. Each button has programmer-defined positions, placed appropriately on the screen where they are to appear to get the game running. These buttons are placed under mouse events. They are attached to the batch object to be drawn all at once.

labellist : list
scorelabellist : list
buttonlist : list

These lists are made so that redrawing and reclearing labels and buttons are easier.

easytime = elements.GameTimer(1, 30)
mediumtime = elements.GameTimer(1, 00)
hardtime = elements.GameTimer(0, 30)

These game timer objects are defined to be displayed during the game. The timer_deplete method in the main game script utilizes these objects. Each timer has a programmer-defined duration, which depends on the game mode for which the timers are made.

Methods:

pyglet.font.add_file (*"assets/MontserratEL.ttf"*)

This imports the font to pyglet so that it will be usable in the program.

pyglet.font.load *("Montserrat ExtraLight", bold = True*)

This loads the font Montserrat ExtraLight, which was added through the add_file method, to be used in the game.

batch_fill ()

This function fills the empty instruction list with label objects, where the text is taken from instructions.txt. The label objects in this list are then drawn from the main game script.

batch_clear ()

This clears all the items in the instructions list.

load_leaderboard ()

This loads the player's name and score during the end of each playthrough in the game. It opens the leaderboard text file and returns a list of all of the lines.

get_scores ()

This takes a string containing a player's name and score and separates them into two variables. This function is used in set_scores.

set_scores ()

This takes the list returned by load_leaderboard and uses the get_scores method to distribute the string variables into the labels defined earlier.

text_input.py

This module contains the classes that create a new window to handle text user-input. This module requires **pyglet** to be installed and uses it.

Classes:

Rectangle

Attributes:

x1, y1, x2, y2 : int

the vertices of the rectangle

batch : obj

batch object to which the button will be appended. This allows it to be drawn together with others when called in Text_Input Batch()

Methods:

__init__ (*x1, y1, x2, y2, batch*)

This initializes the vertices of the object to be drawn later on by the batch.

TextWidget

Attributes:

x, y : int

the position of Rectangle

width : int

the width of the Rectangle

batch : obj

batch object to which the button will be appended. This allows it to be drawn together with others when called in Text_Input Batch ().

pad : int

adjusts the height and width of the Rectangle

Methods:

__init__ (*text, x, y, width, batch*)

This initializes the layout of the text which includes the space, font, font size, caret, and the Rectangle.

Text_Input

Attributes:

labels : list

contains all class Label() that appears as texts

widgets : list

contains all class TextWidget() that appears as rectangles where the user provides inputs

text_cursor : class

tells the type of cursor that is used.

focus : object / value

initializes the focus to None as default. This is position of the cursor inside the rectangle where the user provides input.

font : object / value

initializes the type of font used.

string_name : str

initializes an empty string which will soon be edited when the user provides input

batch : obj

batch object to which the button will be appended. This is where the Batch () is called to draw all batch elements.

Methods:

on_draw ()

This draws the background of the Text_Input window with all of the batch elements drawn at once which includes Rectangle() and TextWidget()

on_text ()

This sets the location where the caret appears.

on_text_motion ()

This takes motion values from keyboard such as BACKSPACE key. Currently, the motion accept BACKSPACE as the only key to edit string_name changes before saving this string to file with the function save_name().

on_key_press ()

This handles all key_press on keyboard. Currently, the symbol that were used were limited to the alphabet letter keys as the only keys to edit string_name.

on_close ()

This is called when the user attempts to close the window. This is used to handle no given input name when this window is closed. As the user attempts to close the window at the same time does not give any input, it will raise a statement using the Label() class to be drawn on the screen.

set_focus ()

This sets the caret location as the user provides input and moves to the last character as the string_name gets appended.

Methods:

save_name (*file, name*)

This function saves the name to a file by first, opening it and uses “a+” to read and write at the end of file and a file is created if it does not exist--then closes the opened file.

PARAMETERS
file : variable placeholders
name : str

Sources: Assets Used

The assets used in the program were either made by the programmers or taken from online sites. All resources taken online are royalty-free and under Fair Use.

2D Navigation Buttons

All navigation buttons were created by the programmers specifically designed for the game.

2D Sprites

All sprites used in the game were taken from the site *thenounproject.com*. The sprites, together with their artists, are listed below.

Panda

All panda icons are from the collection created by Nick Bluth. These were retrieved from:
https://thenounproject.com/nickbluth/collection/pandas/?i=495401

Dog

All dog icons are from the collection created by AomAm. These were retrieved from:
https://thenounproject.com/aomam/collection/siberian-husky-emoticons/

Cat

All cat icons are from the collection created by AomAm. These were retrieved from:
https://thenounproject.com/aomam/collection/cat-emoticons-line/

Octopus

All octopus icons are from the collection created by AomAm. These were retrieved from:
https://thenounproject.com/aomam/collection/octopus-emoticons-line/

Raccoon

All raccoon icons are from the collection created by AomAm. These were retrieved from:
https://thenounproject.com/aomam/collection/raccoon-emoticons-line/

Sound Effects

All sound effects used in the game were taken from online sources. All sound effects used are listed below.

Background song

This song is played on loop all throughout the entirety of the game. This was created by Alexander. This sound effect is permitted for non-commercial use under license “Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)“. No changes were made to the sound effect.

It was retrieved from:
http://www.orangefreesounds.com/8-punk-8-bit-music/?fbclid=IwAR3_I5LMhN4tjxat-KoYVNfXtQ7R50kCv9A5ph5TBcnOQ4LHJ1zwDqBK-rs.

A description of the license can be found in:
https://creativecommons.org/licenses/by-nc-sa/3.0/

Clicking sound effect

This sound effect is used whenever the player presses a navigation button/icon. This was created by Alexander. The sound effect is permitted for non-commercial use under license “Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)”. No changes were made to the sound effect.

It was retrieved from:
http://www.orangefreesounds.com/button-sound-wrong/?fbclid=IwAR2Isl6EQofWYxlIiAr1cqGeK2Gno4MdCdnCFFrNEHEafFdgZNsqw34-CPg

A description of the license can be found in:
https://creativecommons.org/licenses/by-nc/4.0/

Correct sound effect

This sound effect instantly plays whenever the player garners one point. This was created by Alexander. The sound effect is permitted for non-commercial use under license “Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)”. No changes were made to the sound effect.

It was retrieved from:
http://www.orangefreesounds.com/error-sound-effect/?fbclid=IwAR3w18eN2SYst-LKbYLQOF5giIAlhAcPAh_3wZiY4QRPnmtexZGB9bl97uY

A description to the license can be found in:
https://creativecommons.org/licenses/by-nc/4.0/

Time out sound effect

This sound effect is used in the end of every level where the timer runs out. This was created by Alexander. This sound effect is permitted for non-commercial use under license “Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)”. No changes were made to the sound effect.

It was retrieved from:
http://www.orangefreesounds.com/retro-game-sound-effect/?fbclid=IwAR1JZpUQtVp7f-IuF7s-FDbx-VsWa3sdRj5L_LZiR9KV_xRHrfR9JQQ_mtM

A description of the license can be found in:
https://creativecommons.org/licenses/by-nc/4.0/