BINUS UNIVERSITY INTERNATIONAL

BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter
(Individual Work)

Student Information:		Surname	Given Names	Student ID Number
	1.	Suratno	Ivan Ezechial	2101693920

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1BC**	**Name of Lecturer(s)**	: 1. Minaldi Loies 2. Jude Josheph Lamug Martinez
Major	**: CS**		
Title of Assignment (if any)	: Pygame "Matching Cards"		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 6-11-2017**	**Submission Date**	**: 6-11-2017**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Bina Nusantara International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Bina Nusantara International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Ivan Ezechial Suratno

(Ivan Ezechial suratno)

"Matching Cards"
Name: Ivan Ezechial Suratno
ID: 2101693920

I. Project Specification

a. The function of this program:

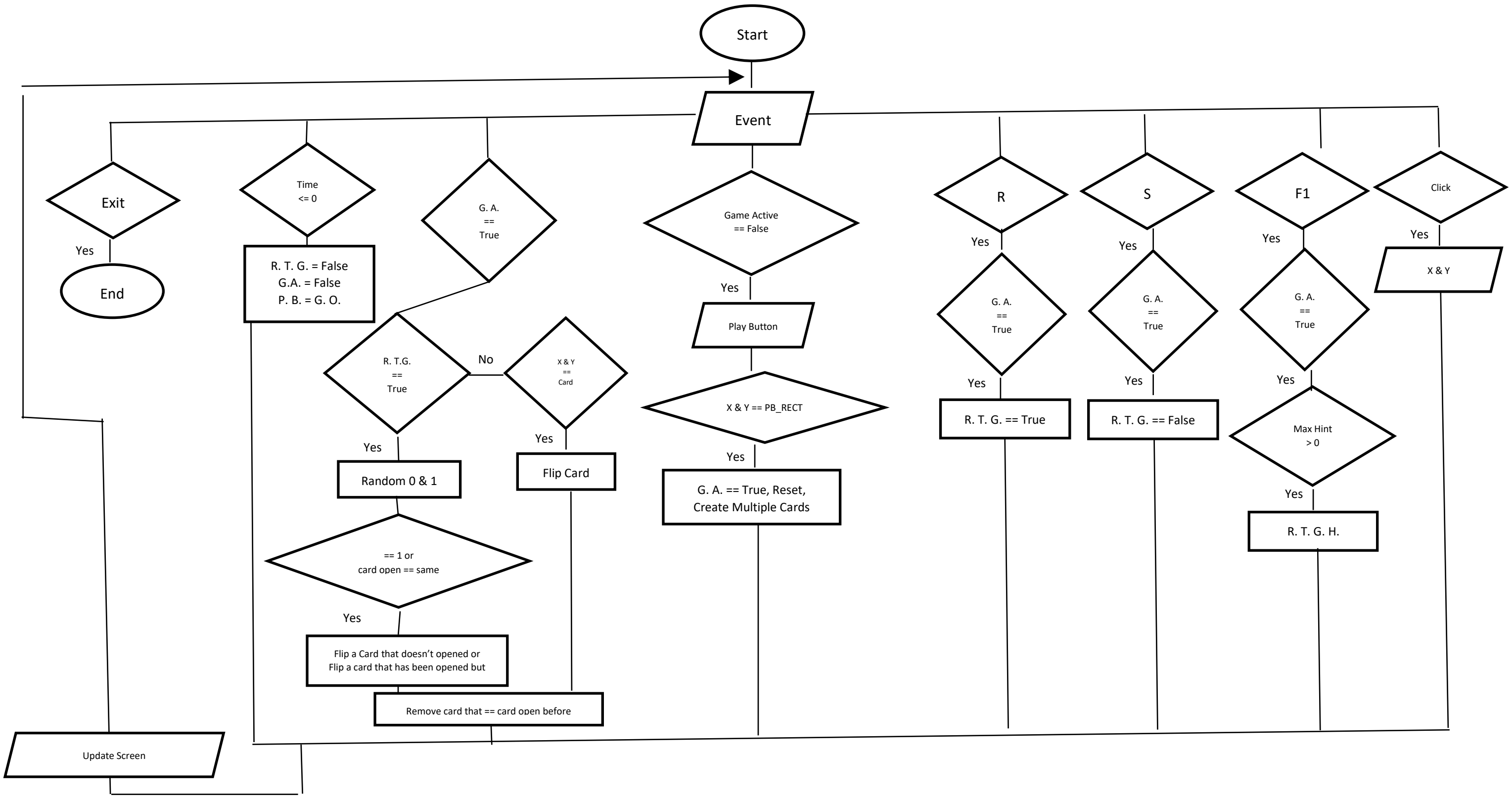
This program is a game based on PyGame and Python. The program code is based on Alien Invasion "Python Crash Course". This game is a matching card game with time based scale program. The objective of the game is to finish matching 52 cards before the time runs out. The card that need to be match should be the same card value such as King of Dimond and King of Heart and so on until 52 cards has been solved. You may use any tool to reach the objective including the hint. After finishing with matching 52 cards, you can get to the next level which the time will got a reduction, the score point will be increase, and you'll get another hint. You'll play this game until you can't handle the time any more. Then, If and only if your score higher than the remaining high score, your score will be save by the game.

b. Set of instruction to Play the Game (Manual)

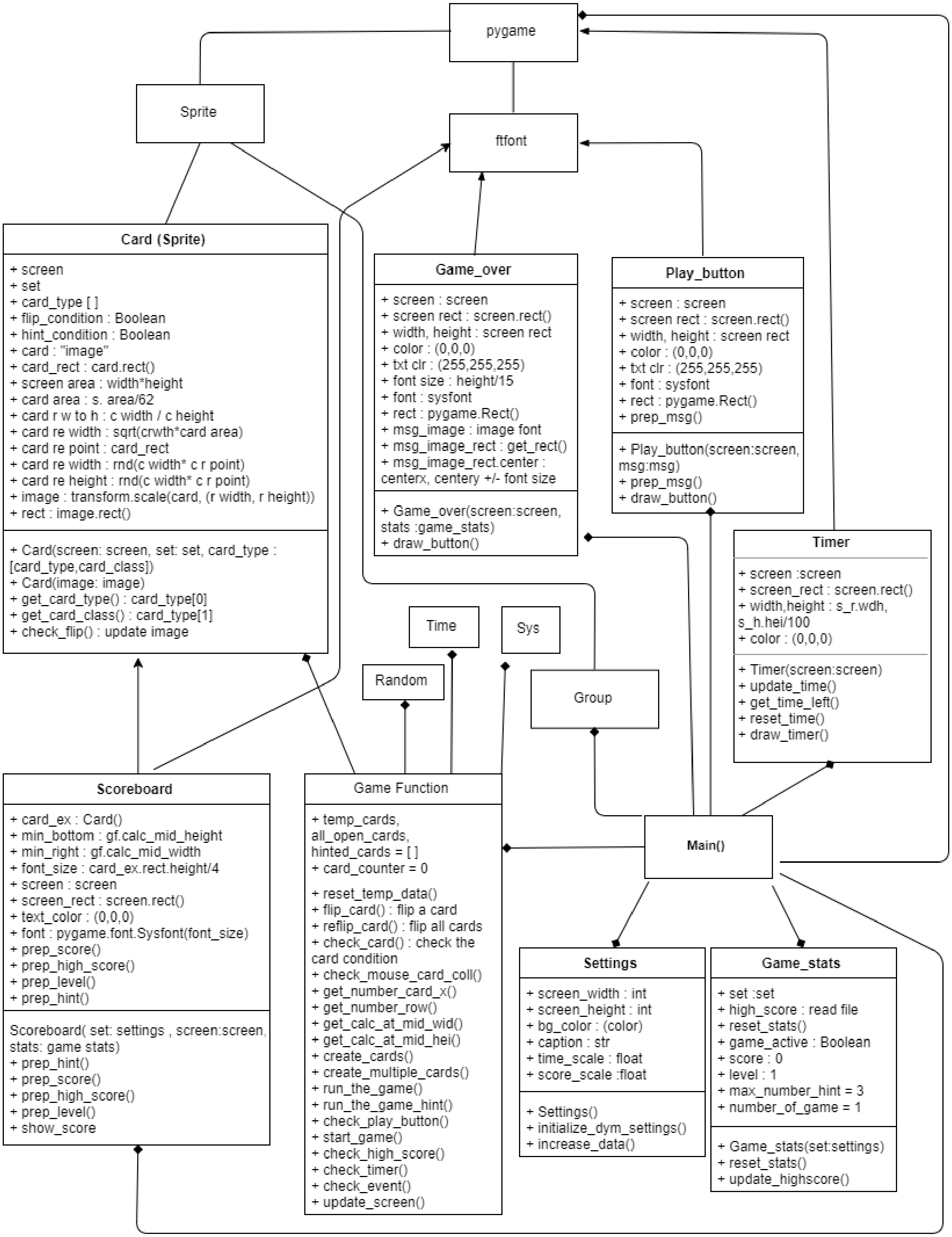
i.	Play Button ("Click to Play")	– Click with your mouse
ii.	Game Over Button	– Click with your mouse
iii.	Flip a Card	– Click with your mouse
iv.	Hint	– Click F1/Help Key
v.	Tell the computer to Play	– Click R Key
vi.	Tell the computer to Stop Play	– Click S Key

II. Solution Design

a. Design/Plan (Flow Chart)



b. Design/ Plan (UML Diagram)


pygame
Sprite
ftfont
Card (Sprite)
+ screen
+ set
+ card_type [ ]
+ flip_condition : Boolean
+ hint_condition : Boolean
+ card : "image"
+ card_rect : card.rect()
+ screen area : width*height
+ card area : s. area/62
+ card r w to h : c width / c height
+ card re width : sqrt(crwth*card area)
+ card re point : card_rect
+ card re width : rnd(c width* c r point)
+ card re height : rnd(c width* c r point)
+ image : transform.scale(card, (r width, r height))
+ rect : image.rect()
+ Card(screen: screen, set: set, card_type : [card_type,card_class])
+ Card(image: image)
+ get_card_type() : card_type[0]
+ get_card_class() : card_type[1]
+ check_flip() : update image
Game_over
+ screen : screen
+ screen rect : screen.rect()
+ width, height : screen rect
+ color : (0,0,0)
+ txt clr : (255,255,255)
+ font size : height/15
+ font : sysfont
+ rect : pygame.Rect()
+ msg_image : image font
+ msg_image_rect : get_rect()
+ msg_image_rect.center : centerx, centery +/- font size
+ Game_over(screen:screen, stats :game_stats)
+ draw_button()
Play_button
+ screen : screen
+ screen rect : screen.rect()
+ width, height : screen rect
+ color : (0,0,0)
+ txt clr : (255,255,255)
+ font : sysfont
+ rect : pygame.Rect()
+ prep_msg()
+ Play_button(screen:screen, msg:msg)
+ prep_msg()
+ draw_button()
Timer
+ screen :screen
+ screen_rect : screen.rect()
+ width,height : s_r.wdh, s_h.hei/100
+ color : (0,0,0)
+ Timer(screen:screen)
+ update_time()
+ get_time_left()
+ reset_time()
+ draw_timer()
Time
Sys
Random
Group
Scoreboard
+ card_ex : Card()
+ min_bottom : gf.calc_mid_height
+ min_right : gf.calc_mid_width
+ font_size : card_ex.rect.height/4
+ screen : screen
+ screen_rect : screen.rect()
+ text_color : (0,0,0)
+ font : pygame.font.Sysfont(font_size)
+ prep_score()
+ prep_high_score()
+ prep_level()
+ prep_hint()
Scoreboard(set: settings , screen:screen, stats: game stats)
+ prep_hint()
+ prep_score()
+ prep_high_score()
+ prep_level()
+ show_score
Game Function
+ temp_cards, all_open_cards, hinted_cards = [ ]
+ card_counter = 0
+ reset_temp_data()
+ flip_card() : flip a card
+ reflip_card() : flip all cards
+ check_card() : check the card condition
+ check_mouse_card_coll()
+ get_number_card_x()
+ get_number_row()
+ get_calc_at_mid_wid()
+ get_calc_at_mid_hei()
+ create_cards()
+ create_multiple_cards()
+ run_the_game()
+ run_the_game_hint()
+ check_play_button()
+ start_game()
+ check_high_score()
+ check_timer()
+ check_event()
+ update_screen()
Main()
Settings
+ screen_width : int
+ screen_height : int
+ bg_color : (color)
+ caption : str
+ time_scale : float
+ score_scale :float
+ Settings()
+ initialize_dym_settings()
+ increase_data()
Game_stats
+ set :set
+ high_score : read file
+ reset_stats()
+ game_active : Boolean
+ score : 0
+ level : 1
+ max_number_hint = 3
+ number_of_game = 1
+ Game_stats(set:settings)
+ reset_stats()
+ update_highscore()

III. Implemented and How It Works

a. Main Menu *(main.py)*

i. This is the main program where the program run started.

ii. **# Outside Main Function #**

“The first line of my code here is importing the other class, function, module, and package.”

```
import pygame
```

“Here is where I import pygame package”

```
from pygame.sprite import Group
```

“Here is where I import Group from a package pygame and module sprite for creating a Group”

```
import game_fuction as gf
```

“Here is where I import game_function module from my directory”

```
from settings import Settings as settings
from scoreboard import Scoreboard as Sb
from stats import Game_stats as Gs
from timer import Timer
from playbutton import Play_button
from game_over import Game_over as Go
```

“From my directory I import those classes and function”

iii. **# Inside Main Function #**

```
def main():
```

“Create a function call main”

```
   pygame.init()
```

“This is for initializing pygame”

```
   set = settings()
```

“Call the setting class as a variable called set”

```
   stats = Gs(set)
```

“Call the Game Stats Class as a variable called stats”

```
   pygame.display.set_caption(set.caption)
```

“Call the pygame package then implement the display module then set_caption module, this is to set caption in the left corner of the program beside the pygame logo”

```
    screen = pygame.display.set_mode((set.screen_width, set.screen_height))
```

“Create screen by calling pygame.display module inside package then implement the set_mode module, this is to set width and height of the screen than call that as a variable call screen.”

```
    sb = Sb(set,screen,stats)
```

“Create scoreboard by calling scoreboard class then set that as a variable sb, then I input the parameter of the class Sb which is settings, screen, and stats”

```
    timer = Timer(screen)
```

“Create timer by calling Timer class input the parameter required which is screen then set it to a variable”

```
    play_button = Play_button(screen, "Click to Play")
```

“Create a play button by calling play_button class then input the parameter which is screen and a massage then set it to a variable”

```
    cards = Group()
```

“Create a group that we call from pygame.sprite this is a list containing a class call card that later will be added to it”

```
while True:
```

“Create a loop to run the pygame continuously”

```
        if stats.run_the_game and stats.game_active:
            gf.run_the_game(cards,stats,sb)
```

“This condition is to let the computer play the game by itself, if and only if the game is active and the run the game is active”

```
       gf.check_event(set,cards,stats,sb,play_button,screen)
```

“This for checking the event within the game itself”

```
      gf.update_screen(set, screen, cards, play_button, stats, sb, timer)
```

“This for updating the screen after checking event in the game”

```
        if stats.game_active == False:
            if stats.number_of_game > 1:
                play_button = Go(screen,stats)
        stats.run_the_game = False
```

“So, when the Game not active the play button changes to game over button if the number of game higher than 1 (which is after playing the game) and if the computer plays the game it make sure it doesn’t continue”

```
    if stats.game_active:
        gf.check_card(cards, set, sb, stats, screen, timer)
```

“This is for checking the card after it open either by a human player or computer player only if the game is active”

```
main()
```

“Call the function main”

b. **Card *(card.py)***

   i. **# Outside Card Class #**

```
import pygame
import math
```

“Here is where I import pygame and math package”

```
from pygame.sprite import Sprite
```

“Here is where I import Sprite class from a package pygame and module sprite for creating a Group”

   ii. **# Inside Card Class #**

```
class Card(Sprite):
```

“Create a Card class which inherited from a class called sprite”

```
   def __init__(self, set, screen, card_type):
```

“Create a function that initialized the class with the parameter of the class”

```
      super(Card, self).__init__()
```

“Initialized card class in the super class which is the sprite”

```
      self.screen = screen
```

“Set the screen of the class”

```
      self.set = set
```

“Call the setting from the parameter”

```
      self.card_type = card_type
```

“Call the card type of the card, the card type parameter is a list which contain the type of the card and the class of card which for example [“As”,”Hearts”]”

```
        self.flip_condition = True
```

“Create a condition of flip inside the initializer of the class, this for checking the flip condition of the card”

```
        self.hint_condition = False
```

“Create a condition of hint inside the initializer of the class, this for checking the hint condition of the card”

```
        self.card = pygame.image.load(
'Database\\Cards\\CardCover.png')
```

“Since the first-time card appended is a closed card, I only load the cover card at this time.”

```
        self.card_rect = self.card.get_rect()
```

“Get the rect of a card, since pygame knows the picture as a rectangle. So, by getting the rect we get the width and the height of the card it self”

```
        self.screen_area = set.screen_width * set.screen_height
```

“Get the screen area of the screen”

```
        self.card_area = self.screen_area / 62
```

“To get the card area I calculate the screen area and divide by 62, 62 is a number that get from 52 cards + 10 that for calculating more than the needed cards area.”

```
        self.card_ratio_w_to_h =
self.card_rect.width/self.card_rect.height
```

“Calculate the ratio width to height of the card”

```
        self.card_re_rect_width = math.ceil( math.sqrt(
self.card_ratio_w_to_h * self.card_area))
        self.card_re_rect_height = math.ceil( math.sqrt(
(1/self.card_ratio_w_to_h) * self.card_area))
```

“Get the new width and height from square root of ratio times card area then round the number of it”

```
    def get_card_type(self):
        return self.card_type[0]
```

“The function, get card type is to return the card type which is like “As,1,2,3, etc.”

```
    def get_card_class(self):
        return self.card_type[1]
```

“This function, get card class is to return the card class which is like “Hearts, Diamond, Clubs, and Spade”

```
    def check_flip(self):
```

“Create a module fuction that check the flip of the card”

```
        if self.hint_condition:
            self.card = pygame.image.load('Database\\Cards\\CardCover(hint).png')
```

“*If the card hint_condition true then return the card as hinted”*

```
        elif self.flip_condition:
            self.card = pygame.image.load('Database\\Cards\\CardCover.png')
```

“*If the card flip_condition true then return a cover card”*

```
        else:
            self.card = pygame.image.load('Database\\Cards\\{0}of{1}.png'.format(self.card_type[0], self.card_type[1]))
```

“*Else, load the card as based on the card type and card class such as [“As”, “Diamond”] it open the file from the Database then the path to card after that the path to AsofDiamond.png.”*

```
        self.image = pygame.transform.scale(self.card, (self.card_re_rect_width,self.card_re_rect_height))
```

” Set the image of the sprite class to the card from the right condition every time.”

c. **Play Button *(playbutton.py & game_over.py)***

i. **# Outside Play Button and Game_over Class #**

```
import pygame.ftfont
```

“Importing the font based from the pygame font module”

ii. **# Inside Play Button Class # *(playbutton.py)***

```
class Play_button:
    def __init__(self, screen, msg):
        self.screen = screen
        self.screen_rect = self.screen.get_rect()
```

”As pygame interpret everything as a rectangle, the screen interpret as a rectangle, by get_rect() we get the dimension of the screen itself”

```
        self.width, self.height = self.screen_rect.width, self.screen_rect.height
```

“I’m setting the width and the height of the button to the size of the screen itself”

```
        self.button_color = (0, 0, 0)
        self.text_color = (255, 255, 255)
```

“Setting the color of the button to (0,0,0) means black because in RGB light sequence 0 == Off that means Red light is off, Green light is off, and Blue light is off, there is no color so it’s just displaying black the none color. For the text color set it to (255,255,255), same as before, combination or all RGB at max it’s white.”

```
        self.font = pygame.font.SysFont(None, int(self.screen_rect.height/15))
```

“I’m setting the font to System Font with a None type, It means the basic system font, the set the font size to screen height divided by 15 as I want the font 1/15 of the screen height.”

```
        self.rect = pygame.Rect(0, 0, self.width, self.height)
```

“I’m creating a rectangle by telling the pygame.Rect() with a x axis, y axis, the width of the rectangle and the height of it as the parameter. I’m setting the width and the height as I called the self.width and self.height. and 0,0 as the starting point.”

```
        self.rect.center = self.screen_rect.center
```

“Here is where we set the x and y axis of the rectangle, we are assigning the center of the rect to the center of the screen.”

```
        self.prep_msg(msg)
```

“This is initializing the prep_msg function at the initializer of the class”

```
    def prep_msg(self,msg):
        self.msg_image = self.font.render(msg, True, self.text_color, self.button_color)
```

“Render the font based on the self.font with the msg parameter will be displayed and the text color and the background color the same as the button color so it looks like transparent, (you can also None as the background color).”

```
        ---snip---
        self.msg_image_rect.center = self.rect.center
    def draw_button(self):
        self.screen.fill(self.button_color, self.rect)
```

“Here I fill the screen with the button color and the rect location of the button.”

```
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

“Here I blit the msg and the rect of the msg. Which it blit only in the specific location in the screen so I don’t need to flip all the display.”

iii. **# Inside Game_over Class # *(game_over.py)***

```
class Game_over:
---snip---
                self.font_size = int(self.screen_rect.height/15)
```

“This is where I set the font size to be calculated”

```
                self.font = pygame.font.SysFont(None,
self.font_size)
                self.rect = pygame.Rect(0, 0, self.width,
self.height)
```

msg 0 : “Game Over”; msg 1: “Score : {:,}”.format(stats.score)
msg 2: “High Score : {:,}”.format(stats.high_score)
msg 3: “Level : {:,}”.format(stats.level)
msg 4: “Click to Continue”

“{:,} this is to set every 3 character follow by a coma”

```
*********************msg 0 – 4 *********************
                self.msg0_image = self.font.render("Game Over",
True, self.text_color, self.go_color)
                self.msg0_image_rect =
self.msg0_image.get_rect()
                self.msg0_image_rect.center =
(self.rect.centerx,self.rect.centery - self.font_size*2)
```

“Setting the msg image to the center of the screen with a little bit of change in the y axis (for displaying 5 msg) I calculated by the font size times the placement of the word.”

```
        def draw_button(self):
                self.screen.fill(self.go_color)
**********************msg 0 – 4 *********************
                self.screen.blit(self.msg0_image,
self.msg0_image_rect)
```

d. **Timer *(timer.py)***

i. *“This class of timer not really based on time, instead for every loop it minus the length of the width.”*

ii. **# Outside Timer Class #**

```
import pygame
```

iii. **# Inside Timer Class #**

```
class Timer:
--- snip ---
        self.height = self.screen_rect.height/100
```

“Here are where I set the height to the 100 of the screen height.”

```
        self.timer_color = (0,0,0)
    def update_time(self,min_fr_time):
        if self.width >= 0:
            self.width -= self.screen_rect.width*min_fr_time
```

“When updating the time, time minus the presentence ok decreasing.”

```
    def get_time_left(self):
        return self.width
```

“Return the width of the timer since it continuously minus it”

```
    def reset_time(self):
        self.width = self.screen_rect.width
```

“Reset the time to the right width.”

```
def draw_timer(self):
--- snip --
        self.rect.top = self.screen_rect.top
```

“Set the rectangle to the top center of the screen.”

```
        self.screen.fill(self.timer_color,self.rect)
```

e. **Game Stats *(stats.py)***

--- snip ---

```
        self.high_score = int(open('Database\\high_score.txt','r').read())
```

“Create a high score that store in a text flie so the high score would be saved.”

```
        self.reset_stats()
        self.game_active = False
```

“Stat that hold the state of the game if it is active or not”

```
        self.run_the_game = False
```

“Stat that hold the state of computer run the game or not”

```
        self.number_of_game = 1
```

“Stat that count the number played, this stat is for stop to create a multiple game over button in the main function.”

```
    def reset_stats(self):
        self.max_number_hint = 3
```

“Stat that hold the max number of hint.”

```
        self.score = 0
```

“Stat that hold the score”

```
        self.level = 1
```

“Stat that hold the level”

```
    def update_highscore(self):
        open('Database\\high_score.txt','w').write(str(self.high_score))
```

“Update the high score after it been change by playing the game.”

f. **Game Scoreboard *(scoreboard.py)***

---snip---

```
            self.card_ex = Card(set, screen, ['As', 'Hearts'])
```

“This is to get an example of a card to calculate later”

```
            self.min_bottom = gf.get_calc_at_mid_hei(set,self.card_ex)
            self.min_right = gf.get_calc_at_mid_wid(set,self.card_ex)
```

“Get the calculation of the number of add to get the card to the middle, I get the number to put the scoreboard at the border that it made by calculating it.”

---snip---

g. **Game Function *(game_function.py)***

”This all set of function that can be executed for the game itself”

i. **# Outside All the Function #**

---- snip ----

```
import sys
```

“Import the system itself.”

```
import random
```

“Import random functionality such as random number between 1 to 10 and the output is such as ‘7’.”

```
import time
```

“Import time functionality such as time.sleep(int) to delay the system.”

```
temp_cards = []
all_open_cards = []
hinted_card = []
card_counter = 0
```

“This is a set of temporary data to be used in the game function as a temporary basis to calculate.”

ii. **# Function #**

1.
```
def reset_temp_data():
    global temp_cards, card_counter, all_open_cards,
hinted_card
```

“This is needed to get access to the temporary data”

```
    temp_cards = []
    all_open_cards = []
    hinted_card = []
    card_counter = 0
```

“It reset all the temporary data.”

2.
```
def flip_card(card):
    global ---snip---
    temp_cards.append(card)
```

“As it being already open it has got to go to temp_cards, which at this point it register the temp cards list as a list of open temporary card.”

```
    card.hint_condition = False
```

“This is for registering the card that the hint return to false if the card is flip, because you don’t need hint if you already got the card open.”

```
    card.flip_condition = False
```

“The change condition to change the card cover to card picture.”

```
    if card not in all_open_cards:
        all_open_cards.append(card)
```

“First it check if the card it’s already open before or not then if it not then it append the card to all open cards.”

```
    card.check_flip()
```

“This recall the function inside the card class which check the flip condition then change the card according to the condition.”

```
    card_counter += 1
```

“Since it flip 1 card it add to the card counter which to count the card that already been opened.”

3.

```
def reflip_card(cards):
    global ---snip---
    for card in cards:
        card.flip_condition = True
        card.check_flip()
```

“What this do is to flip all the card that exist in cards group.”

```
    temp_cards = []
    hinted_card = []
    card_counter = 0
```

“Then reset partial data of the temporary data. Only for all open cards it dismissed that because it still need to remember all the card that has been open before and the group still has it.”

4.

```
def check_card(cards, set, sb, stats, screen, timer):
    global ---snip---
```

```
  check_time(set, timer, stats)
```

"This is for checking the time if it ok or not"

```
  if card_counter == 2 and
temp_cards[0].get_card_type() ==
temp_cards[1].get_card_type():
    stats.score += set.card_points * card_counter
    sb.prep_score()
    check_high_score(stats, sb)
    for card in cards:
      if card == temp_cards[0] or card ==
temp_cards[1]:
        time.sleep(0.3)
```

"Time delay before removal of the cards so the card could be shown."

```
        cards.remove(card)
        all_open_cards.remove(card)
```

"Check whether the card in the temp cards the same, if it is, it will add the score then preparing the score then continuing check the score corelate with high score then check the card to all the list of the cards and the remove it from all open cards and the cards group ."

```
    temp_cards = []
    card_counter = 0
```

"Reset the temporary data of temporary cards and counter, so it can be used again in the next check card."

```
  elif card_counter == 2 and
temp_cards[0].get_card_type() !=
temp_cards[1].get_card_type():
    time.sleep(0.3)
    reflip_card(cards)
```

"If it not then reflip cards."

```
  if len(cards) == 0:
    start_game(set, screen, stats, sb, cards, timer)
```

"Re run the game for the new level if the length of cards is equal to 0"

```
    if card_counter > 2:
        reflip_card(cards)
```

“This is to prevent the cards for opening more then 2 cards at once.”

5.
   ```
   def check_mouse_card_collisions(cards):
       for card in cards:
   ```

   “For checking every card in the cards group.”

   ```
           if card.rect.collidepoint(pygame.mouse.get_pos()):
               card.flip_condition = False
               card.check_flip()
   ```

   “Then check whether the card is colliding with the mouse or not.”

   ```
               if card not in temp_cards:
                   flip_card(card)
   ```

   “Then check whether the card already exist or not if not the flip the card.”

   ```
               elif card in temp_cards:
                   reflip_card(cards)
   ```

   “If it already exist it reclose the card.”

6.
   ```
   def get_number_cards_x(set, card_width):
       number_cards_x = int(set.screen_width / card_width)
       return number_cards_x
   ```

   “Get the amount of card in a row that could fit the card.”

7. get_number_rows ***---snip---***

   “It basically the same as the number of card x, instead of width use height.”

8.
   ```
   def get_calc_at_mid_wid(set,card):
       return int((set.screen_width - get_number_cards_x(set, card.rect.width) * card.rect.width) / 2)
   ```

   “This is how to calculate the number that should be added to get all the card to the middle of the screen.”

9. get_calc_at_mid_hei ***---snip---***

   “The same as before, instead of width use height.”

10.

```
def create_cards(set, screen, cards, card_number, row_number):
global ---snip---
    type_card = ['As', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King']
    card_model = ['Diamond', 'Hearts', 'Clubs', 'Spade']
```

“All the card combination.”

```
    while True:
        type_card_num = random.randint(0, 12)
        card_model_num = random.randint(0, 3)
```

“Random the card that need to be shown.”

```
        if [type_card_num, card_model_num] not in temp_cards :
            temp_cards.append([type_card_num, card_model_num])
            break
```

“While in a loop, check if the card already exists or not, if it not exist then append the card and break the loop.”

```
    card = Card(set, screen, [type_card[type_card_num], card_model[card_model_num]])
    card.rect.x = card.rect.width * card_number + get_calc_at_mid_wid(set,card)
    card.rect.y = card.rect.height * row_number + get_calc_at_mid_hei(set,card)
    cards.add(card)
```

“Then create a card base on the card class, then change the rect based on the number of row and the number of card the card is plus the amount to push it to the middle of the screen.”

11.

```
def create_multiple_cards(set, screen, cards):
    global ---snip---
    reset_temp_data()
```

“This just making sure it reset all data first before create a multilayer card in the screen(display).”

```
    card = Card(set, screen, ['As', 'Hearts'])
```

“Creating one sample card to calculate all the things needed.”

```
    number_cards_x = get_number_cards_x(set,
card.rect.width)
    number_rows = get_number_rows(set,
card.rect.height)
```

“To get the number of row and the number of card that can be put in the screen.”

```
    for row_number in range(number_rows):
        for card_number in range(number_cards_x):
            if len(temp_cards) != 52:
                create_cards(set, screen, cards, card_number,
row_number)
```

“Create a card in a row and card number until the card equal to 52.”

```
            else:
                break
    temp_cards = []
```

“Then, reset the temporary data of the temp cards.”

```
12.def run_the_game(cards,stats,sb):
    global ---snip---
    x = 0
```

“For breaking the loop purposes”

```
    for card in cards:
        y = random.randint(0,1)
        m = random.randint(0,500)
        if len(hinted_card) != 0 temp_cards == 0 :
```

“It will run the hinted card first, if there any hinted card.”

```
            if hinted_card[0] == card:
                flip_card(card)
```

“It just need to flip one card in the hinted card.”

```
                hinted_card.remove(card)
```

“Then remove the hinted card from the temp data.”

```
                break
```

```
        elif m == 1 and stats.max_number_hint != 0 and len(temp_cards) == 0:
“So the computer get a hint by itself if the likeliness of 1 in 500 changes.”
            run_the_game_hint(cards,stats,sb)
“Then run the hint.”
            break
        elif y and card not in temp_cards and card not in all_open_cards:
“This is for not to open the same card again whether in temporary card or in all open card, cause every card has its own type and model/class.”
            flip_card(card)
            break
        elif y and len(all_open_cards) == len(cards) and len(temp_cards) == 0:
“If the remaining of all the card has been open it will equal to the length of all card that remain intact, then it just need to flip any card to match the other card, so the game can still continue.”
            flip_card(card)
            break
        else:
            for card2 in all_open_cards:
                if y and len(temp_cards) != 0 and len(all_open_cards) != 0:
                    if temp_cards[0].get_card_type() == card2.get_card_type() and temp_cards[0] != card2:
                        flip_card(card2)
                        x = 1
“This last part is for checking the card that has been open and located in the temp cards, then open the location where it’s from that already exist in all open cards.”
                        break
                else:
                    break
            if x:
                break
```

“For the y part, is for randomly say Boolean number so yes or no that run or pass to the next. Basically, The program choose what to open as random as possible.”

```
13.def run_the_game_hint(cards,stats,sb):
    global ---snip---
    if stats.max_number_hint>0:
        stats.max_number_hint -= 1
        sb.prep_hint()
```

” First it check if the hint still exist, then it minus since the hint is being used the prep the hint.”

```
        if len(cards) != 0:
            for card in cards:
                x = random.randint(0,1)
                if x and card not in hinted_card:
                    card.hint_condition = True
                    hinted_card.append(card)
                    card.check_flip()
```

“Change the card hinted condition as it being choose random between yes or no in the cards.”

```
                    for card2 in cards:
                        if card.get_card_type() == card2.get_card_type() and card != card2 and card2 not in hinted_card:
                            card2.hint_condition = True
                            hinted_card.append(card2)
                            card2.check_flip()
```

“Change the card2 that the same card type then change hinted condition as it being choose random between yes or no in the cards.”

```
                            break
                    break
```

14.
```
def check_play_button(set, screen, stats, sb, play_button, cards, mouse_x, mouse_y):
    button_clicked = play_button.rect.collidepoint(mouse_x, mouse_y)
```
“Check if the play button being click by the mouse then the state if it True or False”
```
    if button_clicked and not stats.game_active:
        set.initialize_dynamic_settings()
```
“Initializing the setting that are dynamic that can change between the game, so after a new game start that setting can reset also.”
```
        stats.reset_stats()
        stats.game_active = True
        sb.prep_score()
        sb.prep_high_score()
        sb.prep_level()
        cards.empty()
        create_multiple_cards(set, screen, cards)
```
“Restart the game then create the first card set.”

15.
```
def start_game(set, screen, stats, sb, cards, timer):
    cards.empty()
    reset_temp_data()
    timer.reset_time()
```
“Reset data”
```
    set.increase_data()
    stats.level += 1
```
“Add the level”
```
    if stats.level != 1:
        stats.max_number_hint += 1
```
“Add the hint if it not the first level”
```
        sb.prep_hint()
    sb.prep_level()
    create_multiple_cards(set, screen, cards)
```
“Start the game for the next level”

```
16.def check_high_score(stats, sb):
    if stats.score > stats.high_score:
        stats.high_score = stats.score
        sb.prep_high_score()
        stats.update_highscore()
```

“This function check the score to the last high score, if the score is higher to the high score the high score must be equal to the score then update high score to the text file in the database.”

```
17. def check_time(set, timer, stats):
    if timer.get_time_left() < 0:
        stats.game_active = False
        stats.number_of_game += 1
        timer.reset_time()
```

“It check the time if it less than zero, if its correct then the number of game increases and the game goes to inactive mode the reset the time.”

```
    else:
        timer.update_time(set.min_fr_time)
```

“So if it not yet below zero, it going to minus the time by each check (each frames).”

```
18.def check_event(set, cards, stats, sb, play_button, screen):
    global ---snip---
    for event in pygame.event.get():
```

“Check every event that happened in the pygame event.”

```
        if event.type == pygame.QUIT:
            sys.exit()
```

“If it equal to the Quit function then it exit the sys terminal.”

```
        elif event.type == pygame.KEYDOWN:
            if stats.game_active:
                if event.key == pygame.K_F1 :
                    run_the_game_hint(cards,stats,sb)
```

“If F1 pressed the the hint would run.”

```
        elif event.key == pygame.K_r:
          stats.run_the_game = True
```

“If R Key pressed the the condition of the computer running the game started.”

```
        elif event.key == pygame.K_s:
          stats.run_the_game = False
```

“If R Key pressed the the condition of the computer running the game ended.”

```
    elif event.type == pygame.MOUSEBUTTONDOWN:
      if stats.game_active:
        if card_counter < 2:
          check_mouse_card_collisions(cards)
```

“Only if the card_counter less than 2, it will check the card mouse collisions.”

```
      else:
        mouse_x, mouse_y = pygame.mouse.get_pos()
        check_play_button(set, screen, stats, sb,
play_button, cards, mouse_x, mouse_y)
```

“For game is not active, it’ll check for the mouse play button collisions.”

```
19.def update_screen(set, screen, cards, play_button,
   stats, sb, timer):
     screen.fill(set.bg_color)
     cards.draw(screen)
     if not stats.game_active:
       play_button.draw_button()
     sb.show_score()
     timer.draw_timer()
     pygame.display.flip()
```

“It basically blit than flip all the screen.”

```
     if stats.run_the_game:
       time.sleep(0.3)
       timer.update_time(set.min_fr_time*3)
```

“And only for the game run by computer it get a sleep

function for every update screen so it don't go as rapid as it could be."

h. **Settings *(settings.py)***

---snip---

```
        self.screen_width = 600
        self.screen_height = 600
```

"Setting the screen width and height"

```
        self.bg_color = (230, 230, 230)
        self.caption = 'Matching Cards'
```

"Setting the caption name here"

```
        self.time_scale = 1.2
        self.score_scale = 1.5
```

"Setting the factor speed for each level up"

```
    def initialize_dynamic_settings(self):
        self.card_points = 5
        self.min_fr_time = 0.00005
```

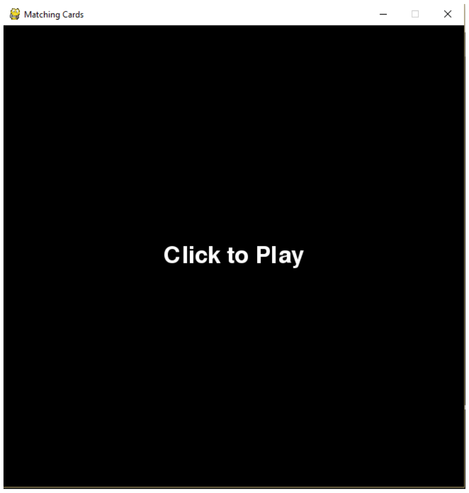
"This function is initializing the settings that can change within the game."

```
    def increase_data(self):
        self.card_points = int(self.card_points*self.score_scale)
        self.min_fr_time = self.min_fr_time*self.time_scale
```

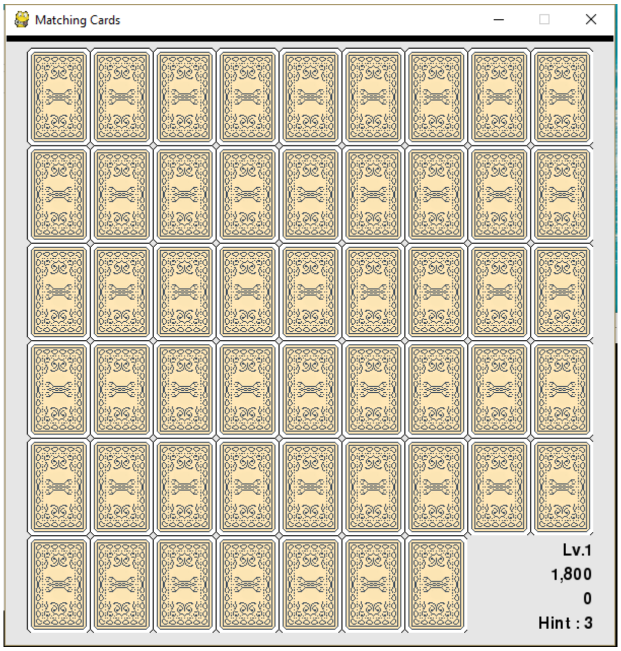
"This function is for increasing the data within the game itself at the next level."

IV. Evidence of Working Program.

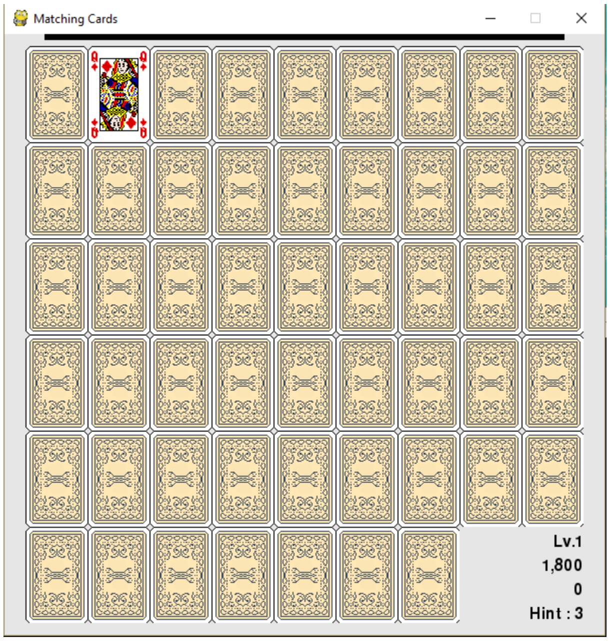
a. **Play Button Screen**

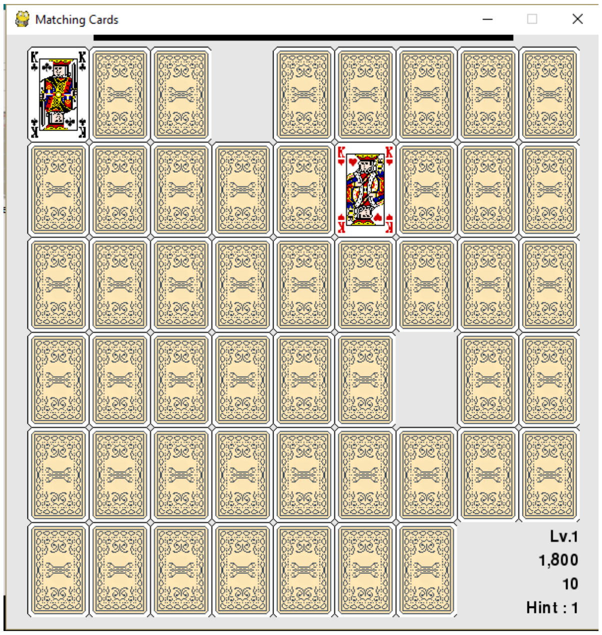


b. Game Screen (Start a Game)



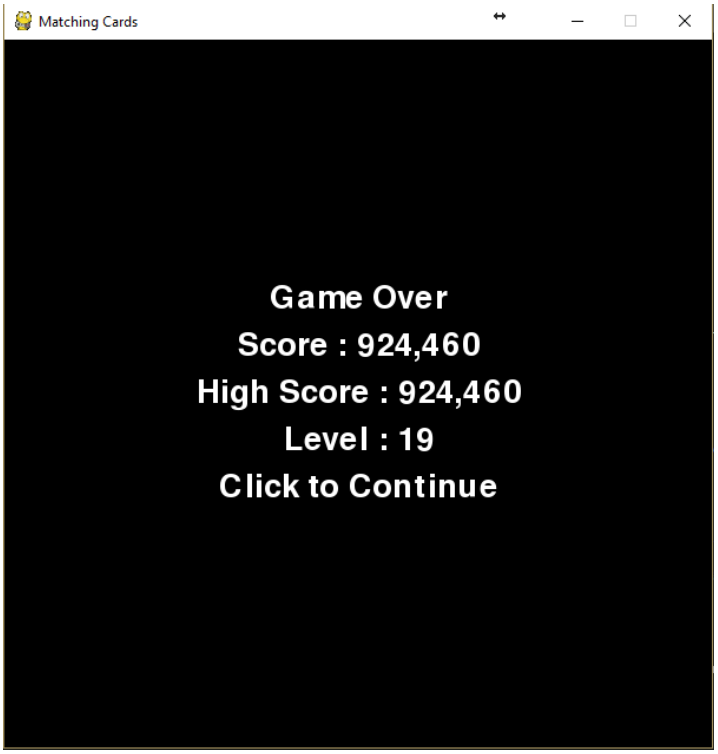
c. Open Card





"This is the evidence of card opening and card matching with a used of hint and the increasing property of score and 2 match card."

d. **Game Over Button/Screen**



Game Over
Score : 924,460
High Score : 924,460
Level : 19
Click to Continue

“This is the evidence that the computer playing, the level 19 and the impossible high score and also for the game over screen.”

V. Source

a. CITE A WEBSITE - CITE THIS FOR ME
Final Project: 1. Cite a Website - Cite This For Me. Installmeinfo. 2017.
Available at: http://installme.info//wp-content/uploads/2016/02/5_suited_deck_of_cards_28908_1027_615.png.
Accessed October 29, 2017.

b. MATTHES, E.
Python crash course
Final Project: 2. Matthes E. Python Crash Course.