Odd Semester (2023)

BINUS UNIVERSITY INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:	Surname	Given Names	Student ID Number
1.	Yowen	Yowen	2301902390

Course Code	**: COMP6056**	**Course Name**	**: Introduction to Programming**
Class	**: L1AC**	**Name of Lecturer(s)**	Ida Bagus Kerthyayana
Major	**: CS**		
Title of Assignment (if any)	: **Higher-Lower Game**		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 17-01-19**	**Submission Date**	**: 13-01-19**

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

BiNus 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 BiNus 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. Yowen Yowen

"Higher-Lower Game"

Name : Yowen

ID : 2301902390

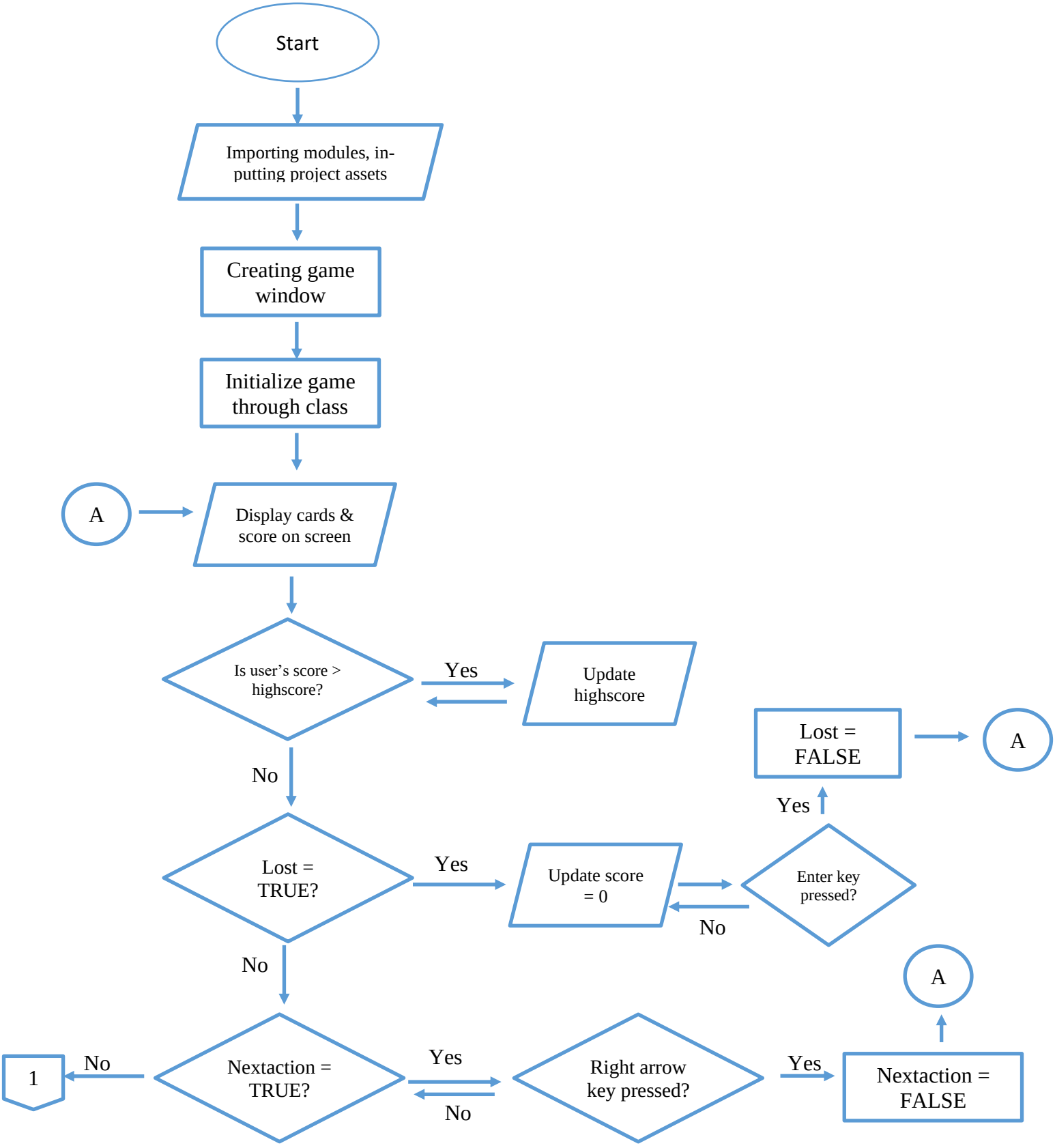
I. Description

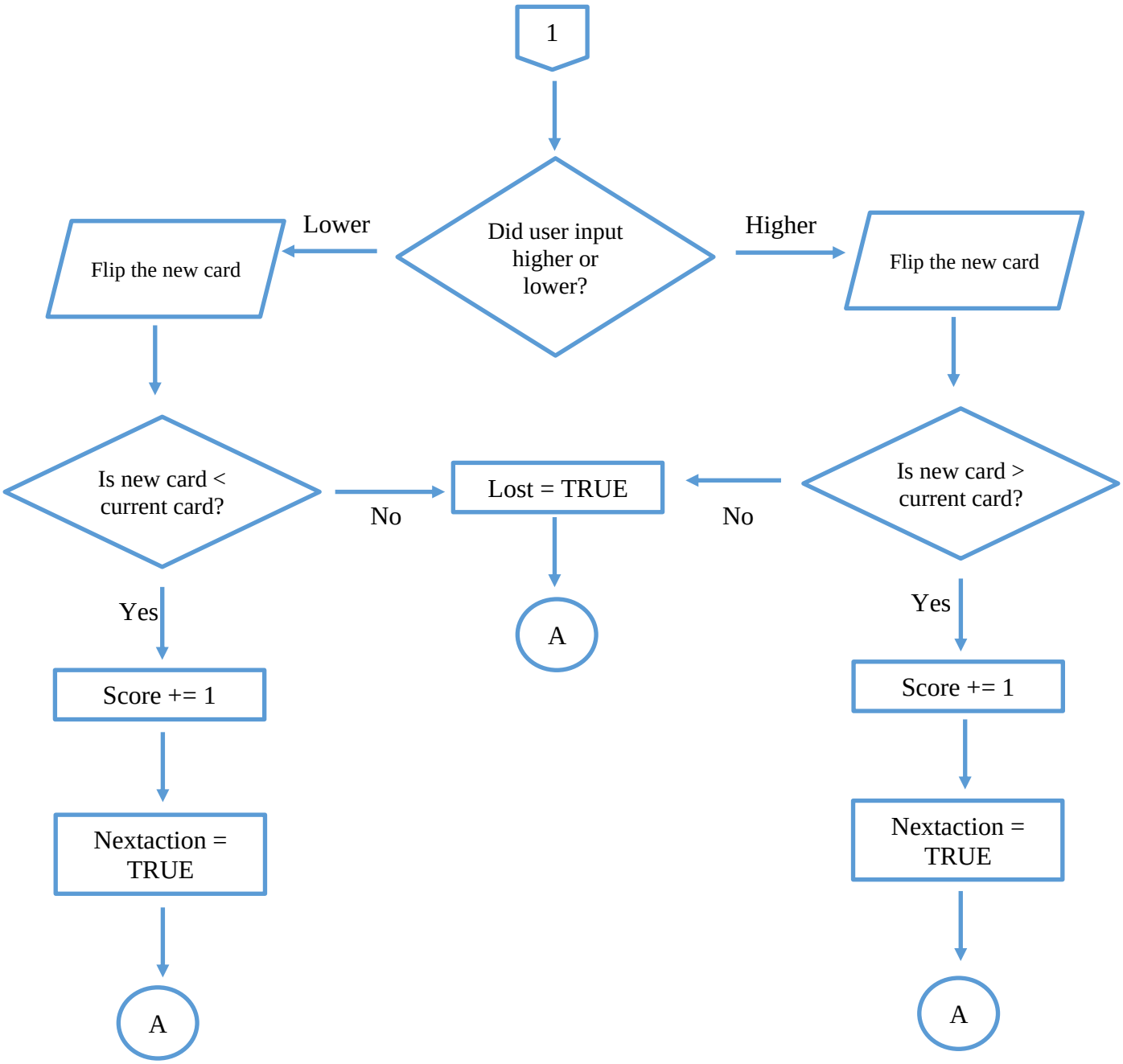
The function of this program:

The purpose of this program is to simulate a game whereby the user needs to predict the value of the next card. There will be a card on the left referred to as 'current card' and a card on the right referred to as 'new card'. The current card will always be face up and the new card will always be face down. The objective of the game is to predict the value of the next card. The user can hit the up arrow key to predict the next card's value is higher than the current card or the user can hit the down arrow key to predict the next card's value is lower than the current card. Every time the user guesses correctly, they gain 1 score. Their total score is displayed on the bottom right along with their all time highscore. On the event where the value of next card is the same as the value of the current card, the program will count that as a win and the user will gain 1 point. If the user guesses incorrectly, they lose the game and their score goes back down to 0.

II.a. Design/Plan

Project's Hierarchy Chart




1
Did user input higher or lower?
Lower
Higher
Flip the new card
Flip the new card
Is new card < current card?
Is new card > current card?
No
No
Lost = TRUE
A
Yes
Yes
Score += 1
Score += 1
Nextaction = TRUE
Nextaction = TRUE
A
A

II.b. Explanation of Each Function Inside the Class

- **__init__(self):**
    - This init function sets up the self.current_card and self.next_card in the format (cardvalue-cardsymbol) this format is necessary for the next step
    - From self.current_card and self.next_card, using the split method, obtain index 0 of splitting with '-' separator to obtain only the card value
    - self.score keeps track of the user's score throughout the game, its default value is set to 0
    - self.lost and self.nextaction are flags used to check whether the user has won or lost. It is by default set to False
- **higher(self) :**
    - This function is called when the user choses to hit the up-arrow key thus going higher
    - This function firstly displays the hidden next card by blitting its image into the screen
    - The function then checks the outcome whether the user won or lost
    - Depending on the outcome, the user will be redirected to the function self.win() or self.lose()
- **lower(self) :**
    - This function is called when the user choses to hit the down-arrow key thus going higher
    - This function firstly displays the hidden next card by blitting its image into the screen
    - The function then checks the outcome whether the user won or lost
    - Depending on the outcome, the user will be redirected to the function self.win() or self.lose()

- **win(self) :**
    - This function is called when the outcome of user's action is a win
    - Firstly, it plays a sound effect
    - It then blits in the image to show that the user has won
    - It then does self.score += 1
    - It then sets self.current_card = self.next_card to pass the revealed card to the left side thus making it the new current card
    - Then it choses a new card value and card icon to be used as self.next_card randomly
    - It then updates the self.raw_value_current and self.raw_value_next for the new card
    - Finally, it sets self.nextaction = True, the image that was blitted prompts the user to hit the right-arrow key. When the user hits the right-arrow key, self.nextaction will turn back to False thus allowing the user to continue the game
- **lose(self) :**
    - This function is called when the outcome of user's action is a lost
    - Firstly, it plays a sound effect
    - It then blits in the image to show that the user has lost
    - It then saves the user's highscore into a txt file called 'highscore.txt'
    - Then it sets self.score back to 0
    - Finally, it sets self.lost = True, the image that was blitted prompts the user to hit the enter key. When the user hits the enter key, self.lost_shuffle() will be run. While self.lost is True, the program won't continue unless the user hits the enter key.

- **lost_shuffle(self) :**
    - This function is called when the user hits the enter key after self.lost = True
    - It acts as the function that resets the game
    - First it obtains a new pair of self.current_card and self.next_card
    - Using the new self.current_card and self.next_card, it obtains new values for self.raw_value_current and self.raw_value_next
    - Lastly, it sets self.lost = True, allowing the user to continue playing

- **set_score(self,score) :**
    - This function is used to update the user's score live
    - It is called everytime the main gameloop function loops
    - The new score is placed in the parameter thus changing the value of self.score

- **symbol_hierarchy(self) :**
    - This function is used to check when both self.raw_value_current and self.raw_value_next have equal value
    - It will go by poker's rule whereby diamonds, clubs, hearts and spades is the order of power
    - It splits self.current_card and self.next_card into self.current_symbol and self.next_symbol
    - Using these new symbols, the program will determine whether the next card is higher or lower

- **reshuffle_check(self) :**
    - This function is used to ensure both self.current_card and self.next_card will never be equal
    - When both values are equal, it sets self.randomize = True
    - While self.randomize is true, the program will obtain a new randomized value for self.next_card
    - When the value of self.next_card is not equal to self.current_card, self.randomize is set to False and the loop ends

Class Diagram

Higher_Lower
- val_list -icon_list -highscore -current_card -next_card -raw_value_current -raw_value_next -score -lost -nextaction -randomize
- higher() -lower() -win() -lose() -lost_shuffle() -set_score(score) -symbol_hierarchy() -reshuffle_check()

III.a. Lessons that Have Been Learned

1. The module 'random':

This module is capable of producing random elements from a given list thus making it perfect for assigning the card's value and symbol from val_list and icon_list respectively.

2. Audio with pygame:

```
# Setting video game assets
win_sound = pygame.mixer.Sound("cards\\music\\winsound.flac")
lose_sound = pygame.mixer.Sound("cards\\music\\losesound.wav")
pygame.mixer.music.load("cards\\music\\bensound-dance.mp3")
```

I learned how to add background music and sound effects to my game using pygame. I also learned how to adjust the volume of the music added.

3. Blitting instead of writing:

I learned that it's much easier to blit text instead of adding them. It also keeps my code shorter and simpler

4. Adding text into pygame window:

```
# Function to display text on the screen
def display_message(msg, font, color, pos):
    screen_text = font.render(msg, True, color)
    gameDisplay.blit(screen_text, pos)
```

```
display_message("Score: "+str(player.score),scorefont,white,scorepos)
display_message("Highscore: "+str(player.highscore),scorefont,white,highscorepos)
```

Although I previously mentioned that it's much easier to blit in text through images instead of writing them, I was unable to use this technique for score and highscore as both data are dynamic thus I needed to find out how to actually add text into pygame.

III.b. Problem that Have Been Overcome

I ran into some problems when creating this program. Initially, I did not use the variable self.raw_value_current and self.raw_value_next to obtain the value of each cards. Instead, I used the index 0 of self.current_card and self.next_card. At the time I did not realized how big of a mistake this was as I assigned jack,queen, king and ace as 11, 12, 13 and 14 respectively. This meant the program would read it's value as '1' and thus thinking the card 3 was higher than Jack. I had to overcome this problem by instead setting up the format 'cardvalue-cardsymbol'. This way I am able to obtain the pure value of the card by using the split method.

Resources :

-freesound.org (royalty free sound effects)

-bensound.com (royalty free background music)

- https://stevepython.wordpress.com/2018/11/09/python-gui-card-game/ (card game assets)

V. Source Code

Github link: https://github.com/dankpanda/finalprojectpython

```
import pygame
import random

# This program assumes that you have all required assets on a file named 'cards' on your directory

pygame.init()

# Setting values
display_width = 800
display_height = 600
card1pos = (300,200)
card2pos = (400,200)
scorefont = pygame.font.SysFont(None, 30)
loseFont = pygame.font.SysFont(None,115)
losepos = (400,300)
scorepos = (665,550)
highscorepos = (665,575)
white = (255,255,255)
```

```
blue = (0,0,255)
bg_color = (34,177,76)
score_fill = (730,550,30,20)
highscore_fill = (770,575,30,20)

# Setting video game assets
win_sound = pygame.mixer.Sound("cards\\music\\winsound.flac")
lose_sound = pygame.mixer.Sound("cards\\music\\losesound.wav")
pygame.mixer.music.load("cards\\music\\bensound-dance.mp3")
blank_card_img = pygame.image.load('cards\\blank.png')
bg_img = pygame.image.load('cards\\bg.png')
continue_img = pygame.image.load('cards\\continue.png')
continue2_img = pygame.image.load('cards\\continue2.png')
win_img = pygame.image.load('cards\\win.png')
win2_img = pygame.image.load('cards\\win2.png')
retry_img = pygame.image.load('cards\\retry.png')
retry2_img = pygame.image.load('cards\\retry2.png')
music_credit_img = pygame.image.load('cards\\music_credit.png')

# Game window
gameDisplay = pygame.display.set_mode((display_width,display_height))
pygame.display.set_caption('Higher Lower')
clock = pygame.time.Clock()
gameDisplay.blit(bg_img,(0,0))

# Game mechanics
class Higher_Lower():

    val_list = [2,3,4,5,6,7,8,9,10,11,12,13,14]
    icon_list = ['diamonds', 'clubs', 'hearts', 'spades']
    randomize = False

    with open("cards\\highscore.txt",'r') as f:
        f_read = f.read()
    if f_read == '': # Avoid errors in the case where the highscore.txt file is empty
        f_read = 0
    highscore = int(f_read)

    def __init__(self):
        self.current_card = str(random.choice(self.val_list)) + "-" + random.choice(self.icon_list)
        self.next_card = str(random.choice(self.val_list)) + "-" + random.choice(self.icon_list)
        self.reshuffle_check()
        self.raw_value_current = int(self.current_card.split('-')[0])
        self.raw_value_next = int(self.next_card.split('-')[0])
        self.score = 0
        self.lost = False
        self.nextaction = False


    # This function checks the outcome when the user goes higher
    def higher(self):
        next_card_img = pygame.image.load('cards\\'+player.next_card+'.png')
        gameDisplay.blit(next_card_img,card2pos)
```

```
    if self.raw_value_current == self.raw_value_next:
        self.symbol_hierarchy(self.current_card,self.next_card)
        if self.hierarchy_check == 'Higher':
            self.win()
        else:
            self.lose()

    elif self.raw_value_current < self.raw_value_next:
        self.win()
    else:
        self.lose()

# This function checks the outcome when the user goes lower
def lower(self):
    next_card_img = pygame.image.load('cards\\'+player.next_card+'.png')
    gameDisplay.blit(next_card_img,card2pos)

    if self.raw_value_current == self.raw_value_next:
        self.symbol_hierarchy(self.current_card,self.next_card)
        if self.hierarchy_check == 'Lower':
            self.win()
        else:
            self.lose()

    elif self.raw_value_current > self.raw_value_next:
        self.win()
    else:
        self.lose()

# This function will be called when the outcome of user's action is a win
def win(self):
    pygame.mixer.Sound.play(win_sound)
    gameDisplay.blit(win_img,(255,305))
    self.score += 1
    self.current_card = self.next_card
    self.next_card = str(random.choice(self.val_list)) + "-" + random.choice(self.icon_list)
    self.reshuffle_check()
    self.raw_value_current = int(self.current_card.split('-')[0])
    self.raw_value_next = int(self.next_card.split('-')[0])
    self.nextaction = True


# This function will be called when the outcome of user's action is not a win
def lose(self):
    pygame.mixer.Sound.play(lose_sound)
    gameDisplay.blit(retry_img,(255,305))
    with open("cards\\highscore.txt","w") as f: # Saves the new highscore
        f.write(str(self.highscore))
    self.score = 0
    self.lost = True

# This function will be called when the user loses and decides to play again
```

```
    def lost_shuffle(self):
        self.current_card = str(random.choice(self.val_list)) + "-" + random.choice(self.icon_list)
        self.next_card = str(random.choice(self.val_list)) + "-" + random.choice(self.icon_list)
        self.reshuffle_check()
        self.raw_value_current = int(self.current_card.split('-')[0])
        self.raw_value_next = int(self.next_card.split('-')[0])
        self.lost = False

    # Updates the current score
    def set_score(self,score):
        self.score = score

    # This function is used to check the outcome when both cards have equal value us-
ing poker rules
    def symbol_hierarchy(self,x,y):
        self.current_symbol = x.split('-')[1]
        self.next_symbol = y.split('-')[1]
        if self.icon_list.index(self.current_symbol) < self.icon_list.index(self.next_symbol):
            self.hierarchy_check = 'Higher'
        else:
            self.hierarchy_check = 'Lower'

    # Ensures it is not possible for both current and next card to be the exact same card
    def reshuffle_check(self):
        if self.current_card == self.next_card:
            self.randomize = True

        while self.randomize:
            self.next_card = str(random.choice(self.val_list)) + '-' + random.choice(self.icon_list)
            if self.next_card != self.current_card:
                self.randomize = False

# Function to display text on the screen
def display_message(msg, font, color, pos):
    screen_text = font.render(msg, True, color)
    gameDisplay.blit(screen_text, pos)

player = Higher_Lower()
current_card_img = pygame.image.load('cards\\'+player.current_card+'.png')
next_card_img = pygame.image.load('cards\\'+player.next_card+'.png')

# Main game loop
def gameloop():
    pygame.mixer.music.set_volume(0.3)
    pygame.mixer.music.play(-1)
    player.set_score(player.score) # Refreshes the player's score live
    run = True

    while run:
        if player.score > player.highscore: # Updates the highscore live
            player.highscore = player.score
        gameDisplay.fill(bg_color,score_fill)
        gameDisplay.fill(bg_color,highscore_fill)
```

```
        display_message("Score: "+str(player.score),scorefont,white,scorepos)
        display_message("Highscore: "+str(player.highscore),scorefont,white,highscorepos)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                with open("cards\\highscore.txt","w") as f: # Saves the new highscore
                    f.write(str(player.highscore))
                run = False

        # Game flow if player have not lost
        if player.lost == False:
            if player.nextaction == False:

                current_card_img = pygame.image.load('cards\\'+player.current_card+'.png')
                gameDisplay.blit(music_credit_img,(5,0))
                gameDisplay.blit(continue2_img,(275,375))
                gameDisplay.blit(blank_card_img,card2pos)
                gameDisplay.blit(current_card_img,card1pos)
                gameDisplay.blit(win2_img,(255,305))
                gameDisplay.blit(retry2_img,(255,305))

                if event.type == pygame.KEYDOWN:

                    if event.key == pygame.K_UP:
                        player.higher()

                    elif event.key == pygame.K_DOWN:
                        player.lower()

            # This ensures the user does not accidentally choose an action twice and in-
stead prompts for the input 'right' before proceeding
            elif player.nextaction  == True:
                gameDisplay.blit(continue_img,(275,375))
                if event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RIGHT:
                        player.nextaction = False

        # This block of code will run if the user loses
        else:
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:

                    player.lost_shuffle()
        pygame.display.update()
        clock.tick(15)
gameloop()
pygame.quit()
```

Screenshots of working program