Pixel Jack

Creating a Black Jack Game Application using Python and Pygame

BINUS University International

Algorithm and Programming

Final Project 2024

Jonathan Mulyono - 2802537054

Introduction to Pixel Jack

Pixel Jack is a simple and fun digital card game based on the well-known card game Blackjack. The “Pixel” name of the game name is based on the interface of the game which is mostly pixelated and the “Jack” name of the game name is based on the card game, which is Blackjack. This game is built by using Python and also the Pygame library, which allows for easy creation of graphical applications. The goal of this card game is simple, the player must get a hand of cards that adds up to 21 or as close to it as possible, without going over. The player competes against a virtual dealer, and the game tracks wins, losses, and ties over multiple rounds. In this project, the focus was not only replicating the rules and algorithm of Blackjack, but also I am focusing on making the game that is visually appealing and interactive. There are buttons, sound effects, and a user-friendly graphical interface to make the game enjoyable for anyone who plays and not making many confusions.

Problem Analysis

This project can be correlated to the Sustainable Development Goals (SDG), this project primarily correlated to SDG 9: Industry, Innovation, and Infrastructure, because the use of Pygame and Python for game development can promote innovation and the creation of interactive entertainment technologies among society.

Features

Feature	**Description of feature**
Multiple Decks	This game uses 4 decks of cards for added randomness, but the symbol of each deck will not be included in the randomization.
Card Dealing	Cards are dealt randomly from the deck, with one card hidden for the dealer.
Score Calculation	The game calculates the score for each

	hand, including handling Aces (1 or 11).
Win/ Loss Tracking	The game tracks the player's wins, losses, and draws throughout the game.
Interactive UI	This game has buttons for actions (Hit, Stand, Deal Hand, Quit) and visual feedback.
Sound Effects	This game includes sounds for deal hand, hit, stand, and new hand button.
Music Control	The player can turn game music on/off with a dedicated button.

Application Walkthrough

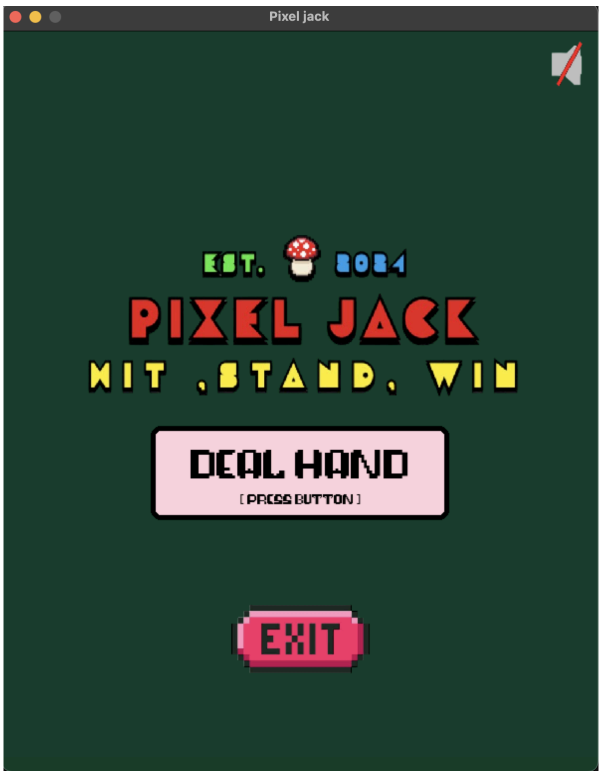


Figure 0. Homepage.

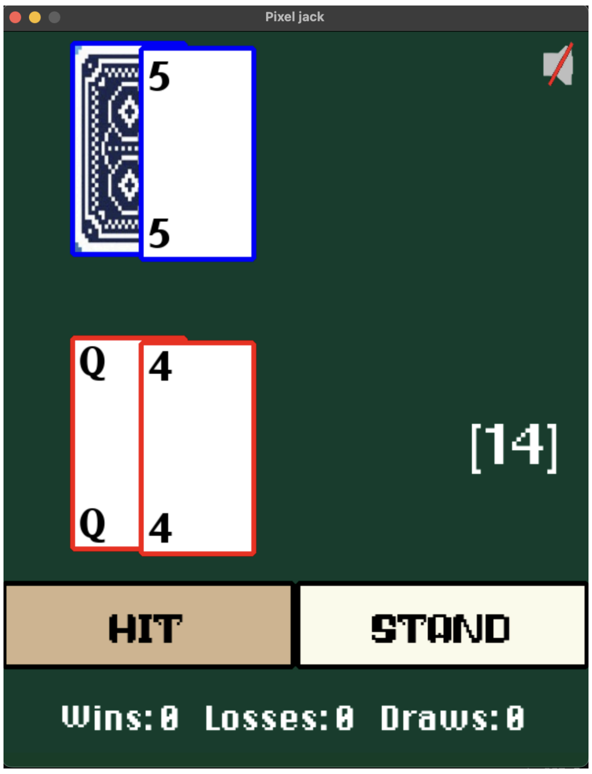


Figure 1. Card Dealing Page.

When the users/players open the game, they will be greeted with an interactive welcome screen displaying the game's logo and buttons for the following actions (Figure 0). The first one is the “Deal Hand” button which will allow the player to begin a new game round. Other buttons in the Homepage include the “Quit Game” button and the music toggle button, if the quit button is pressed, it will allow the player to exit the game at any time, in addition, there is a music toggle button at the top-right corner. If the music is not already playing, the player can click the button to start background music. If music is playing, clicking the button will pause it. Now, if the “Deal Hand” clicked, it will be directed to the Card Dealing Page (Figure 1), the game starts by dealing cards to the player and the dealer. On the Card Dealing Page, the player is dealt two cards, and the dealer is also dealt two cards, but one of their cards remains hidden (the card is facing down) to maintain the mystery and make the game more fun and interactive. This setup mirrors the traditional Blackjack rules, in which the deck is shuffled at the start of each round, ensuring the random card distribution. The cards are drawn and displayed in the following rule, for the player's cards, the player’s cards appear at the bottom of the screen. The cards are displayed with a red outline and white shape, and the card values (such as “3”, “4”, “5”, “6”, “7”, “8”, “9”, “10”, “J”, “Q”, “K”, “A”, “2”) are shown on the top left and right of each card. Cards are drawn in a sequential manner, spaced evenly across the screen, so it makes the interface more appealing. For the Dealer's Cards, they are displayed at the top of the screen, with a blue outline and white shape. The first card is face up, and the second card is hidden behind a card back (represented by the unknown card image or remain mystery). This adds an element of suspense, as the player cannot see the dealer's second card until they stand. The next element on the Figure 1 includes the “HIT” and “STAND” button. If the player wishes to draw another card, they can click the HIT button, located at the bottom-left side of the screen. Each time the player clicks "HIT," a new card is added to their hand. The card is displayed next to the previous cards, and the player's score is recalculated. This continues until the player either decides to Stand or their hand exceeds a total score of 21, which results in a "BUST". The next button is “STAND”, if the player is satisfied with their hand and does not wish to take another card, they can click the STAND button. The player's turn will end, and the dealer is prompted to reveal their second card and proceed with their turn. Each time the player presses the “HIT” or “STAND” button, it will play the sound effect. There are also indicators that count player’s wins, losses, and ties.

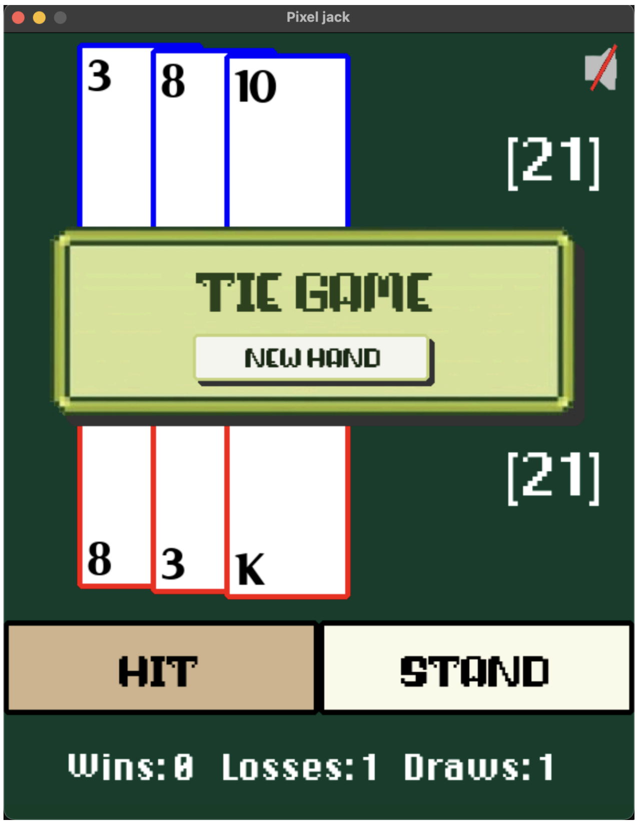


Figure 2. Tie Game Page.

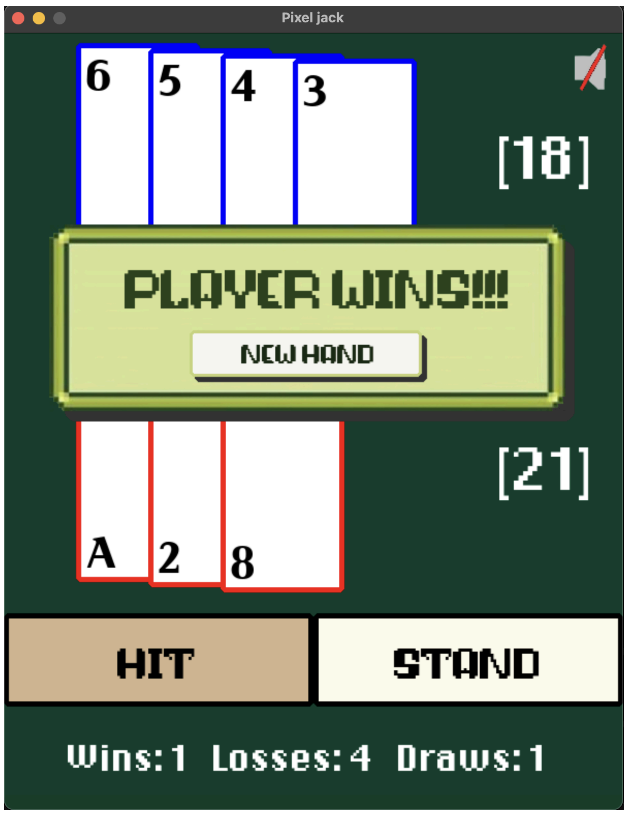


Figure 3. Player Wins Page.

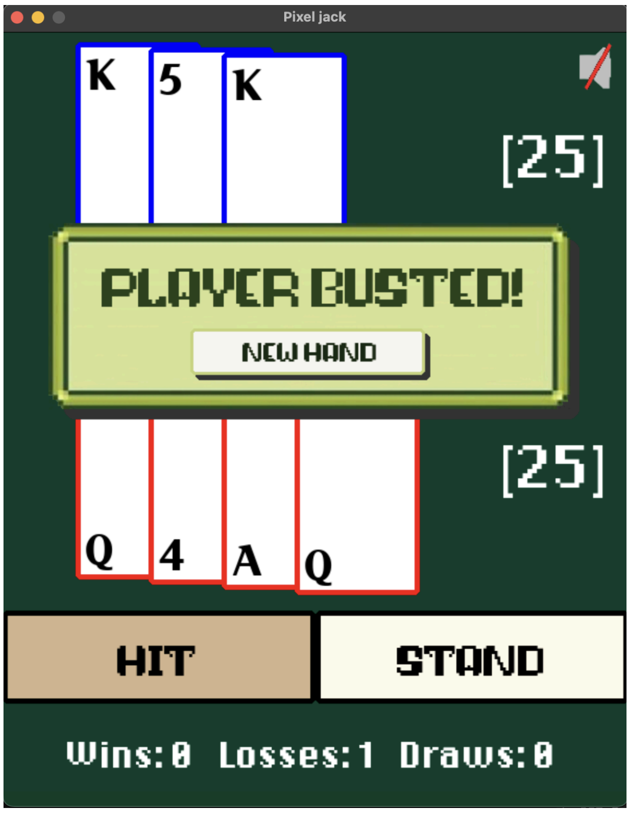


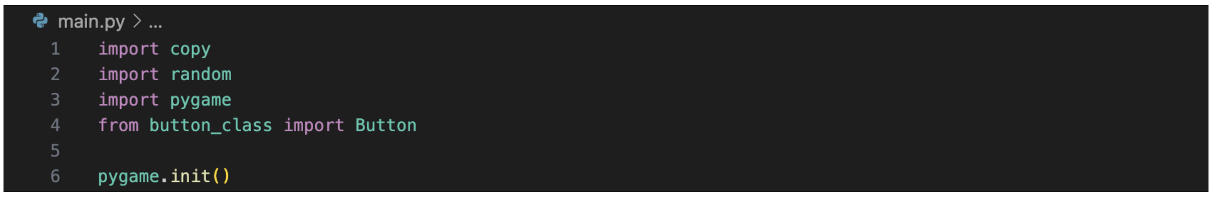
Figure 4. Player Busted Page.

On the other hand, after both the player and dealer have completed their turns, the game evaluates the outcome based on the following conditions, it will give a popup “PLAYER BUSTED!”, as shown in Figure 3, and then the “Wins” indicator will add up one each time. If the player’s hand exceeds 21 points, the player loses the round. The popup will show “PLAYER WINS!!!” (Figure 2), and then the “Losses” indicator will add up one each time., if the player’s hand is higher than the dealer's hand without busting (i.e., closer to 21), the player wins. After that the popup will show “TIE GAME” (Figure 1), and then the “Draws” indicator will add up one each time., if the player's and dealer's hands are of equal value, the round results in a draw, often referred to as a "push." Each time the player presses the “NEW HAND” button, it will play the sound effect and then the page will redirect to the card dealing page again. There is also a toggle music button on the up-right corner, to play and pause the background music.

Program Walkthrough

As this game application uses Python and mainly Pygame, this is the breakdown flow and solution scheme of the main Pixel Jack game program. The program uses Pygame to handle the graphical elements and user interactions, and Python's built-in modules (random, copy) to manage the game logic. After opening the python file, it is a must to install the pygame before, by running: pip install pygame.

1. The first one is initialization, the game starts by importing the important libraries and initializing Pygame.

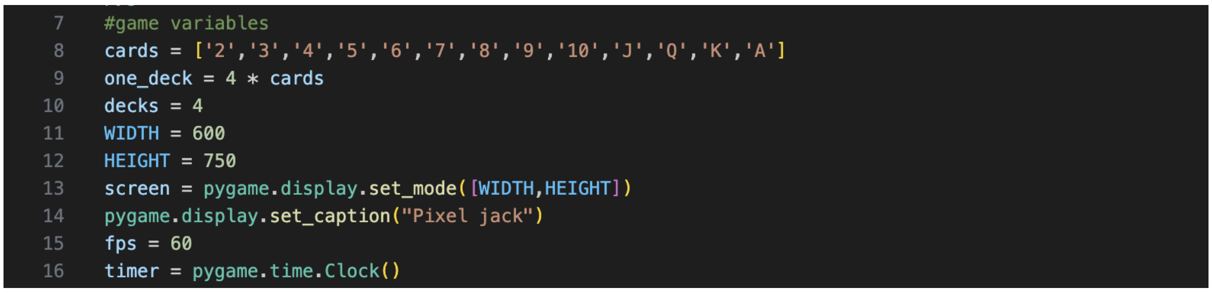


```
main.py > ...
import copy
import random
import pygame
from button_class import Button

pygame.init()
```

- pygame.init() : is used to initialize the Pygame library, so it will enable the use of its graphical, sound, and event-handling features.

2. After the initialization, there are several game variables and configurations to set up, including, the deck of cards, window dimensions (WIDTH, HEIGHT), fonts, and colors.

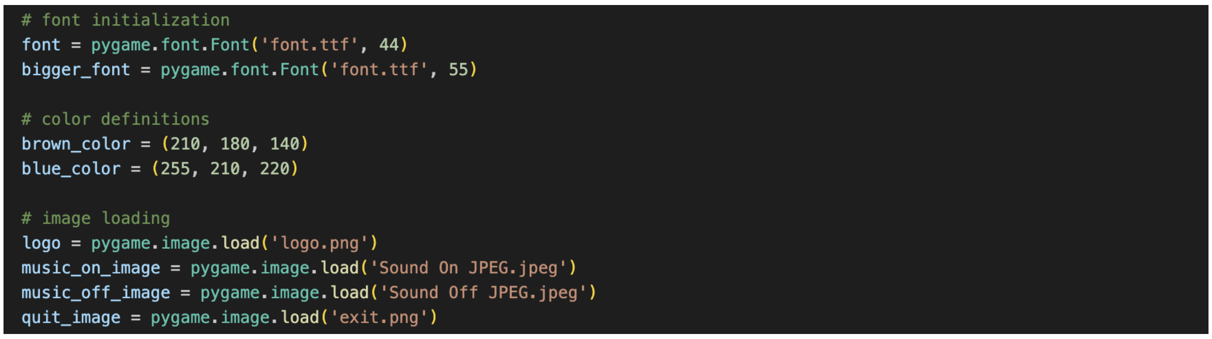


```
#game variables
cards = ['2','3','4','5','6','7','8','9','10','J','Q','K','A']
one_deck = 4 * cards
decks = 4
WIDTH = 600
HEIGHT = 750
screen = pygame.display.set_mode([WIDTH,HEIGHT])
pygame.display.set_caption("Pixel jack")
fps = 60
timer = pygame.time.Clock()
```

- cards : contains the list of the card values
- one_deck : represents a deck of 52 cards, repeated for the specified number of decks.
- screen : is the graphical window where everything is rendered.

- timer : ensures the game runs at 60 frames per second.

3. The program also loads custom fonts for text rendering images for buttons and game elements, to make the game application more attractive.



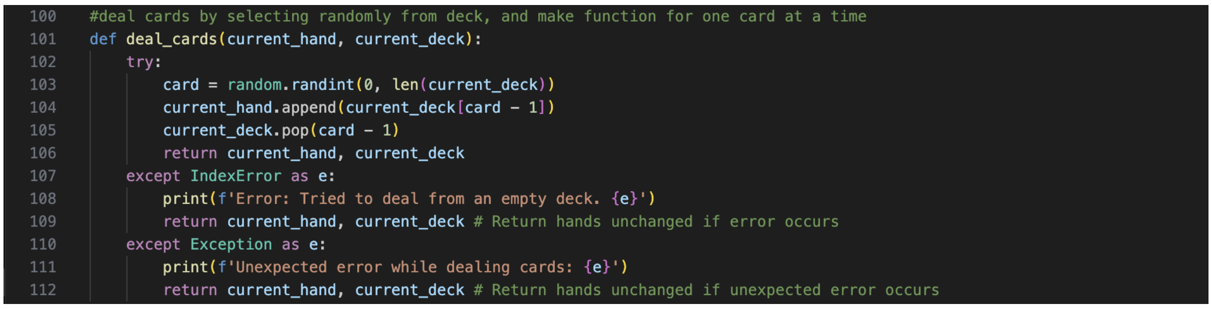
```python
# font initialization
font = pygame.font.Font('font.ttf', 44)
bigger_font = pygame.font.Font('font.ttf', 55)

# color definitions
brown_color = (210, 180, 140)
blue_color = (255, 210, 220)

# image loading
logo = pygame.image.load('logo.png')
music_on_image = pygame.image.load('Sound On JPEG.jpeg')
music_off_image = pygame.image.load('Sound Off JPEG.jpeg')
quit_image = pygame.image.load('exit.png')
```

- For the fonts section, they are used to render text at different sizes and also custom the font style.
- For this program, colors are defined using RGB values for various UI elements like buttons, card outlines, and text.
- Images are loaded for the game logo, music buttons, and the quit button.

4. The next one is the deck and card dealing for shuffling the deck and dealing cards to the player and dealer when the game starts or after a new round.

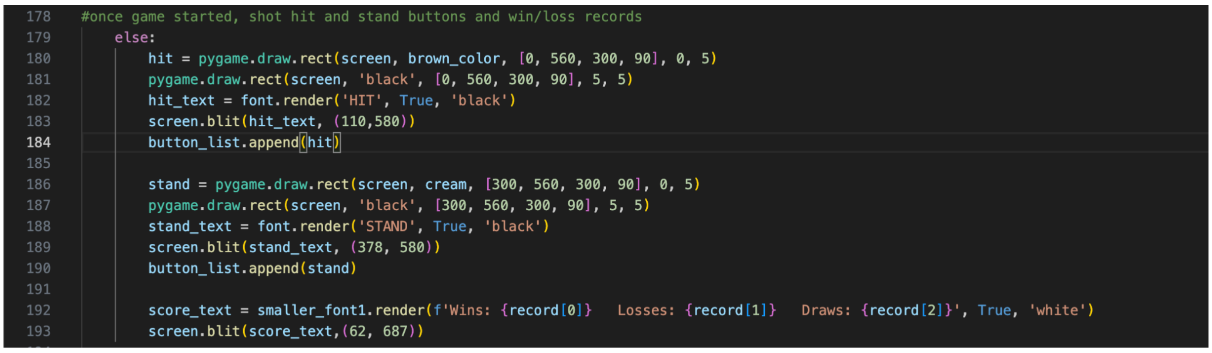


```python
#deal cards by selecting randomly from deck, and make function for one card at a time
def deal_cards(current_hand, current_deck):
    try:
        card = random.randint(0, len(current_deck))
        current_hand.append(current_deck[card - 1])
        current_deck.pop(card - 1)
        return current_hand, current_deck
    except IndexError as e:
        print(f'Error: Tried to deal from an empty deck. {e}')
        return current_hand, current_deck # Return hands unchanged if error occurs
    except Exception as e:
        print(f'Unexpected error while dealing cards: {e}')
        return current_hand, current_deck # Return hands unchanged if unexpected error occurs
```

- For the deck setup, the deck is created by repeating the base card set (cards) four times for four decks and the deck is shuffled to ensure the randomness using random.shuffle(), but on this game application there is no symbol to be randomized.

- The card dealing includes two cards sections, which is the player and also the dealer. The card dealing is using the deal_cards() function randomly picks a card, and then removes it from the deck, and adds it to the appropriate player's hand.
- The line 105 current_deck.pop(card - 1) has an important rule, which is that it will remove one card from the deck and cannot be dealt again, once that card is dealt. So it will prevent repeating the same card.

5. There is also the Hit and Stand button to handle the player's decisions to either draw more cards (Hit) or end their turn (Stand).

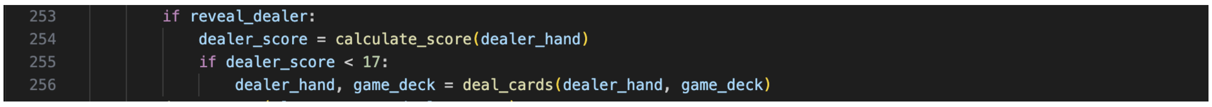


```
178  #once game started, shot hit and stand buttons and win/loss records
179      else:
180          hit = pygame.draw.rect(screen, brown_color, [0, 560, 300, 90], 0, 5)
181          pygame.draw.rect(screen, 'black', [0, 560, 300, 90], 5, 5)
182          hit_text = font.render('HIT', True, 'black')
183          screen.blit(hit_text, (110,580))
184          button_list.append(hit)
185
186          stand = pygame.draw.rect(screen, cream, [300, 560, 300, 90], 0, 5)
187          pygame.draw.rect(screen, 'black', [300, 560, 300, 90], 5, 5)
188          stand_text = font.render('STAND', True, 'black')
189          screen.blit(stand_text, (378, 580))
190          button_list.append(stand)
191
192          score_text = smaller_font1.render(f'Wins: {record[0]}   Losses: {record[1]}   Draws: {record[2]}', True, 'white')
193          screen.blit(score_text,(62, 687))
```

- For the "HIT" button, when the player clicks that button, a new card will be dealt to the player and the player's score section will be recalculated.
- For the "STAND" button, when the player clicks that button, the player's turn ends, and the dealer's turn begins.
- In the code above, there is "else" to make the hit and stand button active after the player hits the "DEAL HAND" button.
- The "screen.blit" is for drawing the "hit_text" (line 183) onto the screen and the button_list.append() function is used to add items (in this case, buttons) to

a list. This list can be used for various purposes, but in this case is for appending the hit and stand button.

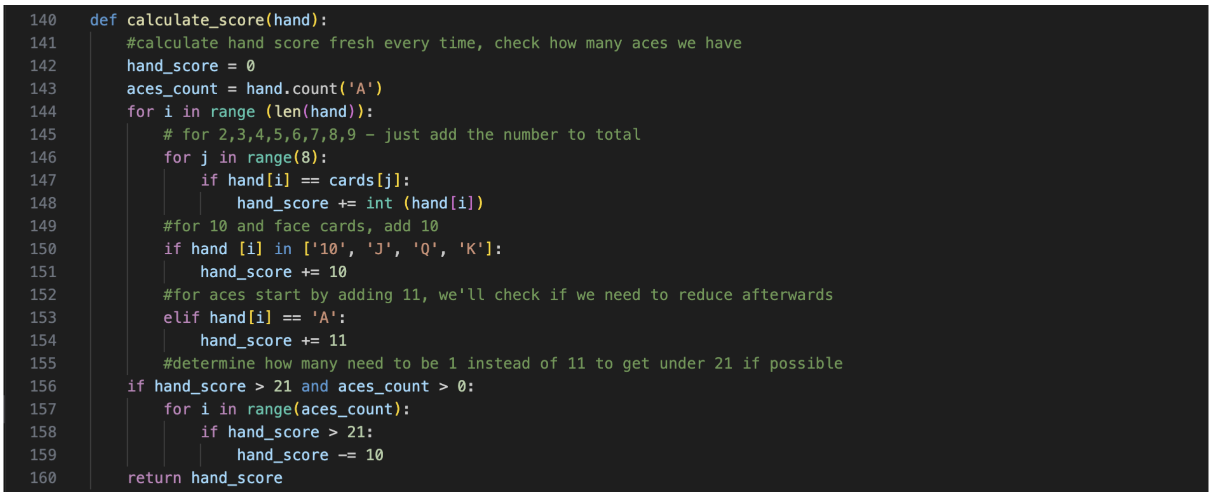
6. As this game application is applied from the basic Blackjack game rule, it has the dealer's turn logic section, which implements the dealer's turn after the player stands, ensuring the dealer draws cards until their score is at least 17. The code below works like this:



```
        if reveal_dealer:
            dealer_score = calculate_score(dealer_hand)
            if dealer_score < 17:
                dealer_hand, game_deck = deal_cards(dealer_hand, game_deck)
```

- if reveal_dealer : This code checks if it's the dealer's turn.
- dealer_score = calculate_score(dealer_hand) : This will calculate the dealer's total hand score.
- if dealer_score < 17 : If the score is less than 17, the dealer draws another card.
- dealer_hand, game_deck = deal_cards(dealer_hand, game_deck) : This is for dealing a new card to the dealer and updating the deck.

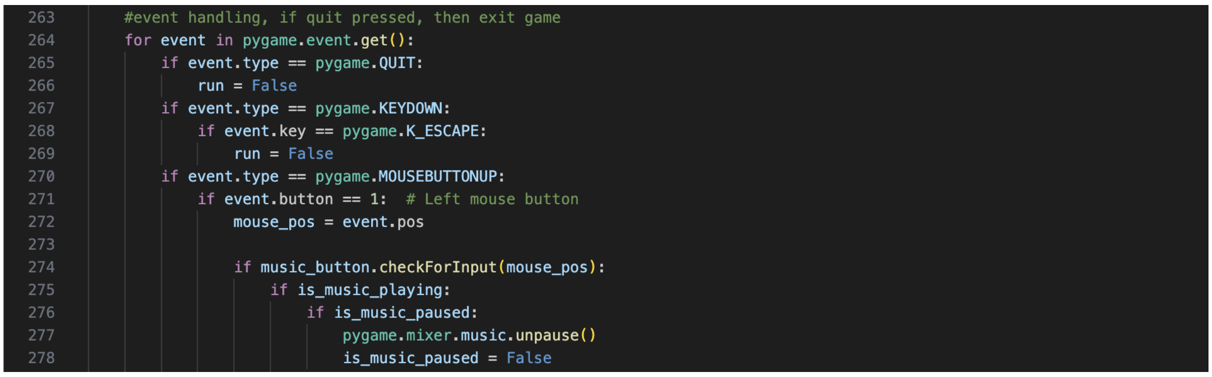
7. This code below is for the score calculation, this algorithm calculates the score of a hand, based on Blackjack rules.



```python
def calculate_score(hand):
    #calculate hand score fresh every time, check how many aces we have
    hand_score = 0
    aces_count = hand.count('A')
    for i in range (len(hand)):
        # for 2,3,4,5,6,7,8,9 - just add the number to total
        for j in range(8):
            if hand[i] == cards[j]:
                hand_score += int (hand[i])
        #for 10 and face cards, add 10
        if hand [i] in ['10', 'J', 'Q', 'K']:
            hand_score += 10
        #for aces start by adding 11, we'll check if we need to reduce afterwards
        elif hand[i] == 'A':
            hand_score += 11
        #determine how many need to be 1 instead of 11 to get under 21 if possible
    if hand_score > 21 and aces_count > 0:
        for i in range(aces_count):
            if hand_score > 21:
                hand_score -= 10
    return hand_score
```

- For number cards, each card from '2' to '10' has a value equal to its number, but for the face cards, like 'J', 'Q', and 'K' are worth 10 points. The special one is for Aces 'A', because it is worth either 1 or 11 points, depending on the hand's total score. If the hand exceeds 21, the Ace is counted as 1 to avoid a bust.
- The algorithm steps:
    a. Loop through the hand and calculate the total score.
    b. But, if the player got any Aces and the score exceeds 21, it will reduce the value of Aces from 11 to 1 to prevent busting or losing.

8. The programs will get an user input via the mouse and keyboard for the event handling.



```python
    #event handling, if quit pressed, then exit game
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                run = False
        if event.type == pygame.MOUSEBUTTONUP:
            if event.button == 1:  # Left mouse button
                mouse_pos = event.pos

                if music_button.checkForInput(mouse_pos):
                    if is_music_playing:
                        if is_music_paused:
                            pygame.mixer.music.unpause()
                            is_music_paused = False
```

- if event.type == pygame.QUIT:

  run = False

  This code is used for the quit button, if the player clicks the quit button, the game will exit.

- if music_button.checkForInput(mouse_pos):

  if quit_button.checkForInput(mouse_pos):

  run = False

  is_music_playing: tracks whether the music is playing or not.

  is_music_paused: tracks whether the music is paused.

  When the user clicks the music button, it will check the current state of the music (whether it's playing or paused).

  If Music is Playing:

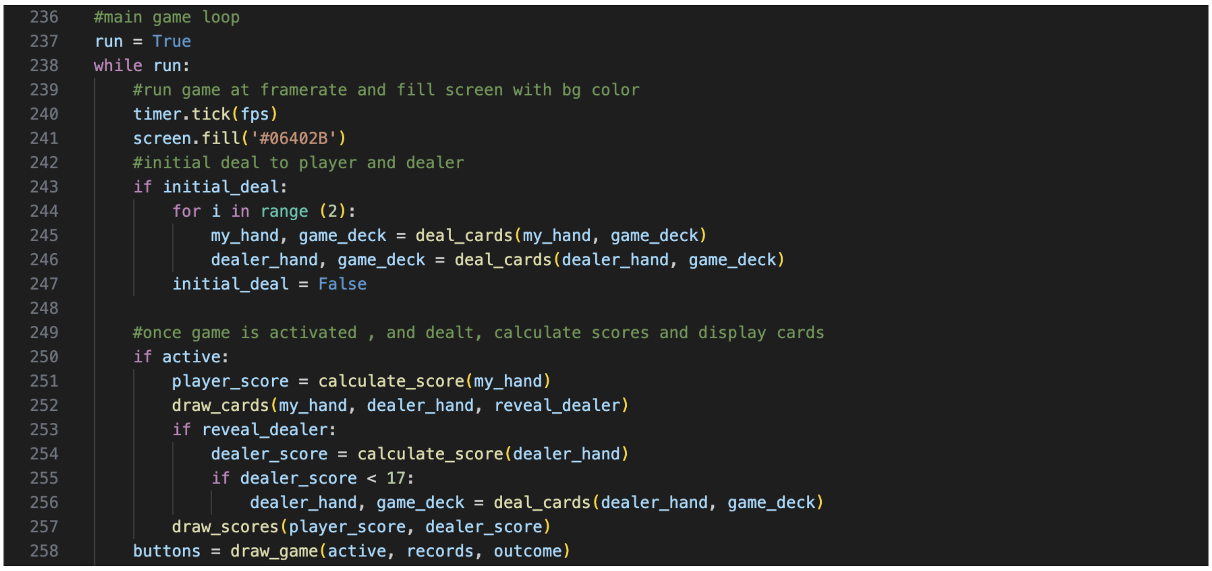
  a. If Paused, the music is unpaused and continues playing.
  b. If Not Paused, the music is paused.

  If Music is Not Playing:

  a. The music is loaded and starts playing, looping indefinitely.

  This code is used for the upper right corner music button, so the player can toggle background music on and off using the music button.

9. The core of the game loop is controlled by the [while run:] block, which it will ensures the game is continuously updated and also rendered, this is the code for the game flow control (from line 237):

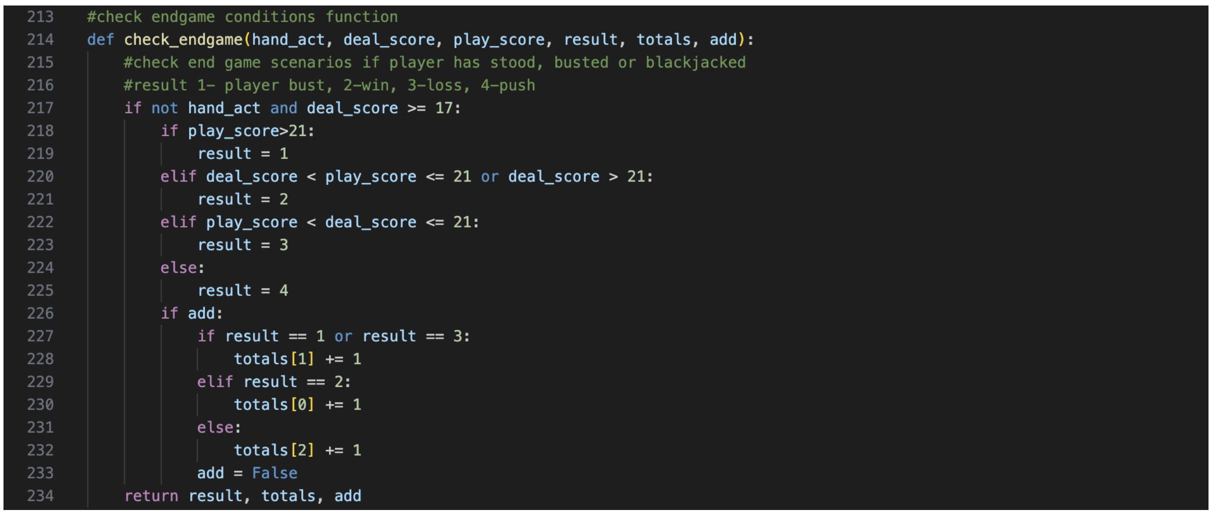


```python
#main game loop
run = True
while run:
    #run game at framerate and fill screen with bg color
    timer.tick(fps)
    screen.fill('#06402B')
    #initial deal to player and dealer
    if initial_deal:
        for i in range (2):
            my_hand, game_deck = deal_cards(my_hand, game_deck)
            dealer_hand, game_deck = deal_cards(dealer_hand, game_deck)
        initial_deal = False

    #once game is activated , and dealt, calculate scores and display cards
    if active:
        player_score = calculate_score(my_hand)
        draw_cards(my_hand, dealer_hand, reveal_dealer)
        if reveal_dealer:
            dealer_score = calculate_score(dealer_hand)
            if dealer_score < 17:
                dealer_hand, game_deck = deal_cards(dealer_hand, game_deck)
        draw_scores(player_score, dealer_score)
    buttons = draw_game(active, records, outcome)
```

- The initial deal flag marks the first two cards that are dealt to both the player and the dealer.
- Once the game is active, the player's and dealer's hands are drawn, and the scores are updated and rendered based on the cards in hand, but on the other hand if the “active” changes to “not active” the card and score will not be rendered and the game loop also will not run.

10. For the next section it is about ending the round and score calculation, the winner is determined based on the score that coded like this:



```python
#check endgame conditions function
def check_endgame(hand_act, deal_score, play_score, result, totals, add):
    #check end game scenarios if player has stood, busted or blackjacked
    #result 1- player bust, 2-win, 3-loss, 4-push
    if not hand_act and deal_score >= 17:
        if play_score>21:
            result = 1
        elif deal_score < play_score <= 21 or deal_score > 21:
            result = 2
        elif play_score < deal_score <= 21:
            result = 3
        else:
            result = 4
        if add:
            if result == 1 or result == 3:
                totals[1] += 1
            elif result == 2:
                totals[0] += 1
            else:
                totals[2] += 1
            add = False
    return result, totals, add
```

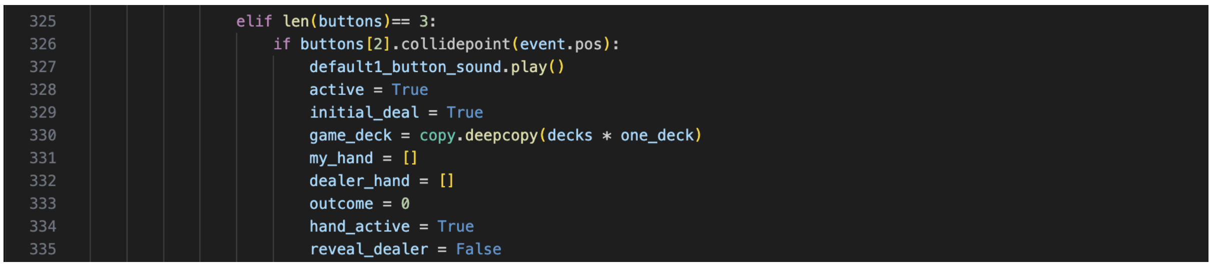
- if play score>21:

result = 1

This code will check if the sum of player cards have exceeded 21 or not, if yes it will result in a pop up “PLAYER BUSTED!”

- The game checks if one of them wins based on the score comparison.

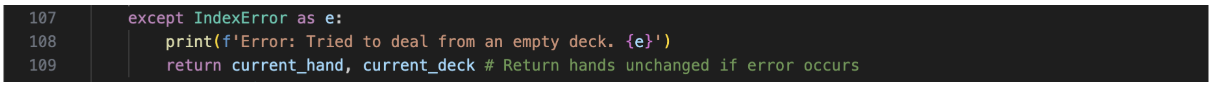
11. The last section is about restarting the game (New Hand), so after the round ends, it will allow the player to start a new hand and reset the game state.



```
elif len(buttons)== 3:
    if buttons[2].collidepoint(event.pos):
        default1_button_sound.play()
        active = True
        initial_deal = True
        game_deck = copy.deepcopy(decks * one_deck)
        my_hand = []
        dealer_hand = []
        outcome = 0
        hand_active = True
        reveal_dealer = False
```

- If this code runs, the game deck is re-initialized with a shuffled card.
- After that, the player's and dealer’s hands are cleared.
- Then the outcome is reset, and the game is ready for the next round.

12. There is also an exception handling mechanism on this Pixel Jack code for an IndexError in Python. One of the example on the code is on the line 109:



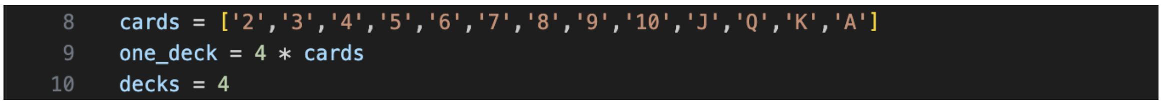
```
107     except IndexError as e:
108         print(f'Error: Tried to deal from an empty deck. {e}')
109         return current_hand, current_deck # Return hands unchanged if error occurs
```

- The line except IndexError as e : is used to catch an IndexError that occurs if the program tries to access an invalid index in a list. In this case, it's specifically used in the context of dealing cards in the Blackjack game. This line has a purpose so if the deck is empty and the game tries to deal a card (e.g., by selecting a random index), an IndexError is raised because there are no more cards left. This line catches that error and the code will not be run again. For the as e part stores the error message in the variable e and print(f'Error: Tried to deal from an empty deck. {e}') : prints a detailed error message for debugging. This exception handling prevents the game from crashing if the deck runs out of cards while dealing (the probability is very small, but it is important), and will inform about the issue, and keeps the game running by returning the current game state.

Data Structures Used

In the Pixel Jack Game, there are several data structures that are used to handle game logic, store information, and also facilitate the interaction between different parts of the game.

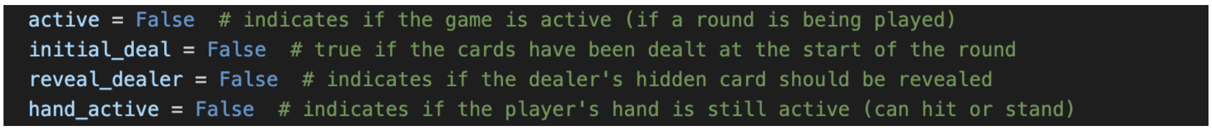
1. **List:** For store dynamic collections such as cards, hands, and game records. Pixel Jack Game uses lists extensively. This is one of the example:



```
cards = ['2','3','4','5','6','7','8','9','10','J','Q','K','A']
one_deck = 4 * cards
decks = 4
```

   - Card Deck. The deck of cards is represented as a list of strings, where each string is a card (such as s “3”, “4”, “5”, “6”, “7”, “8”, “9”, “10”, “J”, “Q”, “K”, “A”, “2”).
      a. game_deck : the list that holds all the cards available in the current game. It is shuffled at the start of each round and cards are drawn from it as players and dealers receive cards.
      b. one_deck : this is a list that represents a single deck of cards. It contains 52 cards, repeated four times (one for each deck in the game).

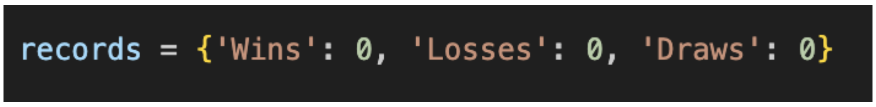
2. **Boolean Variables:** This is used to track the state of the game and control the flow of actions.



```python
active = False  # indicates if the game is active (if a round is being played)
initial_deal = False  # true if the cards have been dealt at the start of the round
reveal_dealer = False  # indicates if the dealer's hidden card should be revealed
hand_active = False  # indicates if the player's hand is still active (can hit or stand)
```

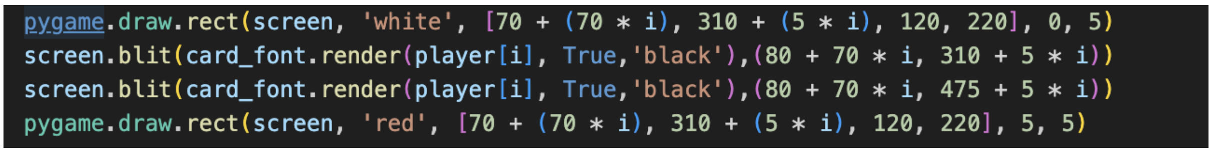
- active : this boolean tracks if the current round is active. When set to True, the player can interact with the game (hit, stand, etc.).
- initial_deal : this boolean used to check if the cards have been dealt to the player and dealer at the start of the round.
- reveal_dealer : this boolean controls whether the dealer's hidden card should be revealed, which is done when the player stands or busts.
- hand_active : this boolean tracks if the player can continue taking actions (e.g., hit or stand). It is True when the player can still hit, and it is False when the hand has ended.

3. **Dictionaries:** Dictionaries are used in Pixel Jack Game to use in the game records (win/loss/draw) to give a better clarity about what each record represents.



```python
records = {'Wins': 0, 'Losses': 0, 'Draws': 0}
```

4. **Tuples:** The Pixel Jack Game use tuples to store the coordinates for UI elements, such as the positioning of cards and buttons on the screen.

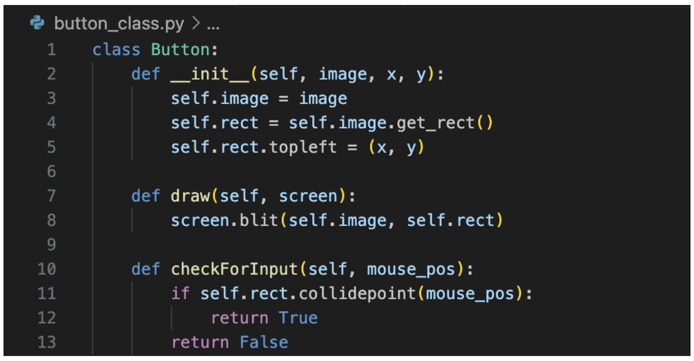


```python
pygame.draw.rect(screen, 'white', [70 + (70 * i), 310 + (5 * i), 120, 220], 0, 5)
screen.blit(card_font.render(player[i], True,'black'),(80 + 70 * i, 310 + 5 * i))
screen.blit(card_font.render(player[i], True,'black'),(80 + 70 * i, 475 + 5 * i))
pygame.draw.rect(screen, 'red', [70 + (70 * i), 310 + (5 * i), 120, 220], 5, 5)
```

5. **Pygame Button Class:** The Button class is used to create interactive buttons in the game, such for hitting, standing, quitting buttons, etc. The Button Class also used to

be a reusable component that simplifies creating and handling the interactive buttons.

Button Class is used in the Pixel Jack Game like this:



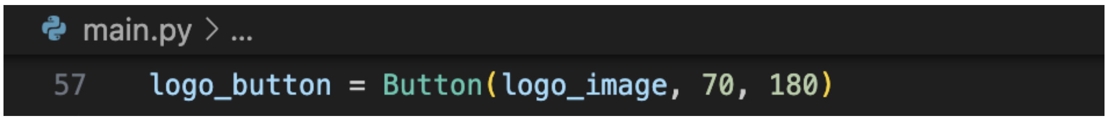
button_class.py > ...

```python
class Button:
    def __init__(self, image, x, y):
        self.image = image
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)

    def draw(self, screen):
        screen.blit(self.image, self.rect)

    def checkForInput(self, mouse_pos):
        if self.rect.collidepoint(mouse_pos):
            return True
        return False
```

The Button is used in the main game loop:

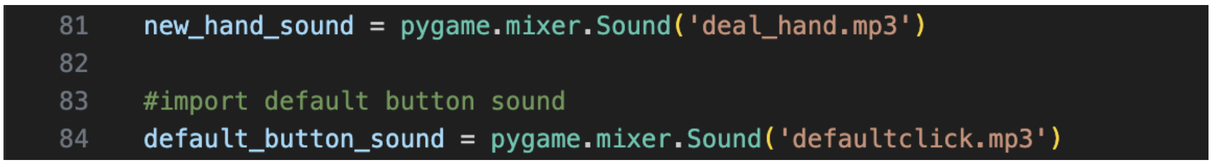


main.py > ...

```python
logo_button = Button(logo_image, 70, 180)
```

The Button on the button_class.py manages the appearance and interaction of buttons, by using draw() for rendering buttons on the screen and then checkForInput() for detecting if a button was clicked based on mouse position. In the main game loop (main.py): the Buttons are used for the logo button, music button, etc. Buttons are drawn to the screen in each frame. So by using this method, it can create multiple buttons without repeating the code. Example, if you need to add new buttons (e.g., for "HIT", "STAND", or "NEW HAND"), you can simply create new Button objects without writing the same logic every time on the main game loop.

6. **Sounds**: Pixel Jack uses Pygame's Sound class to manage sound effects like button clicks, music, and card dealing.

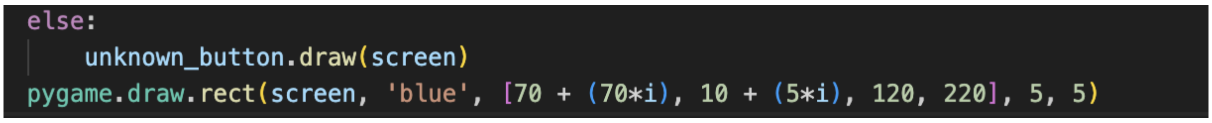


```python
new_hand_sound = pygame.mixer.Sound('deal_hand.mp3')

#import default button sound
default_button_sound = pygame.mixer.Sound('defaultclick.mp3')
```

- Sound : Each sound effect is represented by a *Sound* object in Pygame. These sounds will be played during various actions like button clicks or card dealing to make the game more interactive.

7. **Pygame *Rect* Objects:** This is used for managing the position and dimensions of various UI elements such as buttons and cards. The *Rect* object is playing a role for handling mouse events like button clicks.



```
else:
    unknown_button.draw(screen)
pygame.draw.rect(screen, 'blue', [70 + (70*i), 10 + (5*i), 120, 220], 5, 5)
```

The Pygame *Rect* class is used to define the rectangular regions on the screen. The template is {button_rect = pygame.Rect(x, y, width, height)}. These regions can be used for collision detection, like whether a mouse click intersects with the area of a button in game.

Resources

- Link to GITHUB Repository