# CSA0811-Python programming for-Slod D

## Black Jack Game


Guided By,
T. Vincent Gnanaraj
(Course Faculty)
Python Programming
SSE,SIMATS

Projected By,
Sugumar . D
192224248
(AI&DS)
SSE,SIMATS

# Abstract

The **Blackjack Game In Python** is a fully functional console based application project that includes all of the components that IT students and computer-related courses will need for their college projects or for leisure time purposes. This **Blackjack Python** is a card game in which participants try to get as close to 21 as possible without exceeding it. This **Python Blackjack** is beneficial for practicing Java development and gaining new skills. This project is quite useful, and the concept and logic are simple to grasp. The source code is open source and is free to use. Simply scroll down and click the download button..

# Introduction

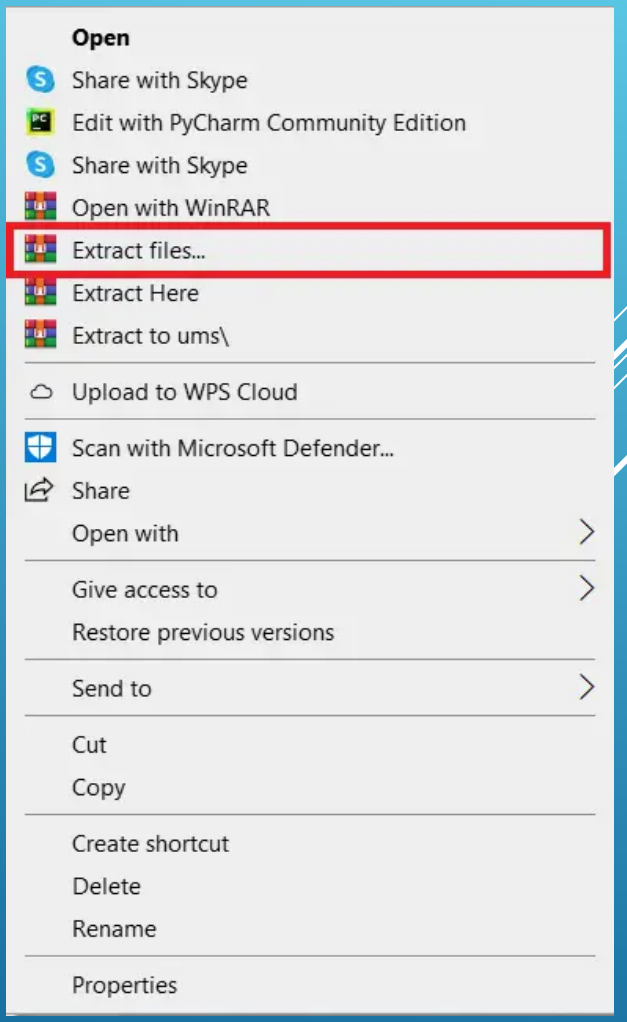
➢The **Blackjack Game In Python** is a fully functional console based application project that includes all of the components that IT students and computer-related courses will need for their college projects or for leisure time purposes. This **Blackjack Python** is a card game in which participants try to get as close to 21 as possible without exceeding it. This **Python Blackjack** is beneficial for practicing Java development and gaining new skills. This project is quite useful, and the concept and logic are simple to grasp. The source code is open source and is free to use. Simply scroll down and click the download button.

# Rules of Black Jack Game in Python

- If a player receives an exact 21 against the dealer, he or she wins. Otherwise, to win, the sum of the player’s cards must be greater than the sum of the dealer’s cards.
- Each face card has a constant value of 10, but depending on the player’s odds of winning, the ace can be counted as one or eleven. The quantity of the remaining cards determines their value.
- In blackjack at a casino, the dealer faces between five and nine (commonly seven) playing positions from behind a semicircular table. Each position may be occupied by up to three players. A single player is often permitted to control or bet on as many positions as desired. At the beginning of each round, bets are placed in the "betting box" at each position in play.

# Steps on how to run **Blackjack Game In Python**

- **Step 1: Download source code**
- **Step 2: Extract file**
- **Step 3: Open Project Path and Open CMD (Command Prompt)**

# Proposed System

Game Rules and Logic:

•Define the rules of Blackjack, including card values, the goal of the game, and special cases (e.g., blackjack, bust).

•Implement the core logic for the game, including dealing cards, calculating scores, and determining winners.

2. User Interface:

•Create a console-based user interface to display the game state and interact with players.

•Display the player's hand, the dealer's hand, and relevant information (e.g., current bet, score).

3. Player Actions:

•Allow players to perform actions such as hitting, standing, and placing bets.

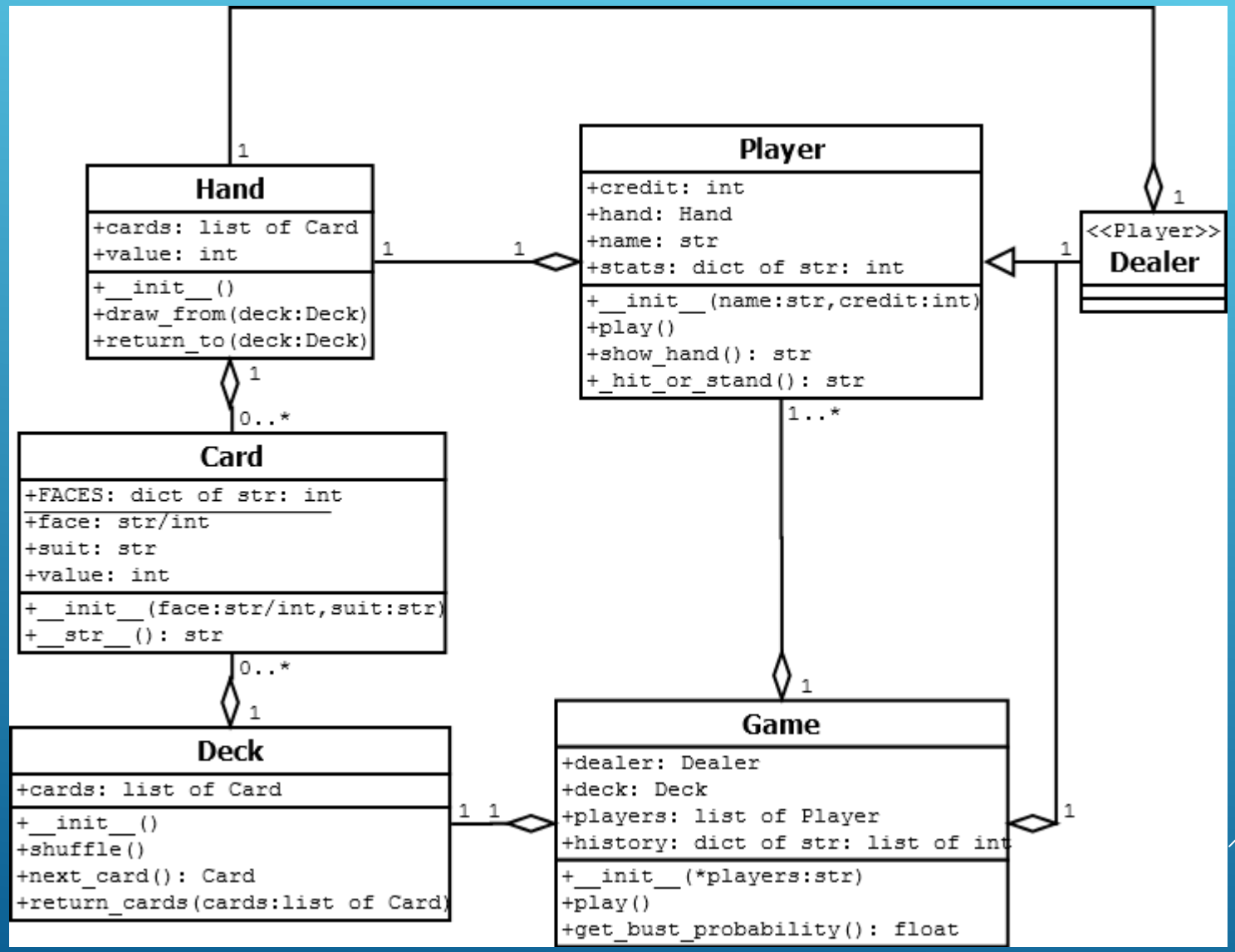
•Implement validation to ensure that actions are legal based on the game state.

4. Deck Management:

•Develop a mechanism to manage the deck of cards, including shuffling and dealing.

•Ensure that cards are drawn randomly and not repeated within a round.

# Architecture



# Design

**1.Card Class:**

1. Represents a playing card with attributes like rank and suit.

**2.Deck Class:**

1. Manages a deck of cards, including functions for generating, shuffling, and dealing cards.

**3.Hand Class:**

1. Represents a player's or dealer's hand of cards.
2. Manages adding cards to the hand and calculating the total score.

**4.Player Class:**

1. Represents a player with attributes such as name, bankroll, and a hand of cards.
2. Manages placing bets and handling win/loss scenarios.

**5.Dealer Class:**

1. Represents the dealer with a hand of cards.
2. Follows a set of rules for playing, e.g., hitting until 17.

**6.BlackjackGame Class:**

1. Orchestrates the game flow, including initializing the deck and players, dealing cards, and determining the winner.
2. Handles player and dealer turns and manages the game loop.

# Coding

```
import random

card_categories = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
cards_list = ['Ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Jack', 'Queen', 'King']
deck = [(card, category) for category in card_categories for card in cards_list]

def card_value(card):
    if card[0] in ['Jack', 'Queen', 'King']:
        return 10
    elif card[0] == 'Ace':
        return 11
    else:
        return int(card[0])

random.shuffle(deck)
player_card = [deck.pop(), deck.pop()]
dealer_card = [deck.pop(), deck.pop()]

while True:
    player_score = sum(card_value(card) for card in player_card)
    dealer_score = sum(card_value(card) for card in dealer_card)
    print("Cards Player Has:", player_card)
    print("Score Of The Player:", player_score)
    print("\n")
    choice = input('What do you want? ["play" to request another card, "stop" to stop]: ').lower()
    if choice == "play":
        new_card = deck.pop()
        player_card.append(new_card)
    elif choice == "stop":
        break
    else:
        print("Invalid choice. Please try again.")
        continue

    if player_score > 21:
        print("Cards Dealer Has:", dealer_card)
        print("Score Of The Dealer:", dealer_score)
        print("Cards Player Has:", player_card)
        print("Score Of The Player:", player_score)
        print("Dealer wins (Player Loss Because Player Score is exceeding 21)")
        break

while dealer_score < 17:
    new_card = deck.pop()
    dealer_card.append(new_card)
    dealer_score += card_value(new_card)
print("Cards Dealer Has:", dealer_card)
print("Score Of The Dealer:", dealer_score)
print("\n")

if dealer_score > 21:
    print("Cards Dealer Has:", dealer_card)
    print("Score Of The Dealer:", dealer_score)
    print("Cards Player Has:", player_card)
    print("Score Of The Player:", player_score)
    print("Player wins (Dealer Loss Because Dealer Score is exceeding 21)")
elif player_score > dealer_score:
    print("Cards Dealer Has:", dealer_card)
    print("Score Of The Dealer:", dealer_score)
    print("Cards Player Has:", player_card)
    print("Score Of The Player:", player_score)
    print("Player wins (Player Has High Score than Dealer)")
elif dealer_score > player_score:
    print("Cards Dealer Has:", dealer_card)
    print("Score Of The Dealer:", dealer_score)
    print("Cards Player Has:", player_card)
    print("Score Of The Player:", player_score)
    print("Dealer wins (Dealer Has High Score than Player)")
else:
    print("Cards Dealer Has:", dealer_card)
    print("Score Of The Dealer:", dealer_score)
    print("Cards Player Has:", player_card)
    print("Score Of The Player:", player_score)
    print("It's a tie.")
```

# Testing

**1.Unit Testing:**

1. **Card Class:**
    1. Test that a card is initialized with the correct rank and suit.
2. **Deck Class:**
    1. Test the generation of a deck with the expected number of cards.
    2. Test that shuffling the deck produces a different card order.
    3. Test that dealing a card reduces the deck size.

**2.Integration Testing:**

1. **Hand Class:**
    1. Integrate with the Deck class to test adding cards to the hand.
    2. Verify that the hand calculates the correct score based on the cards.

**3.Functional Testing:**

1. **Player Class:**
    1. Test placing bets and ensure the player's bankroll is updated accordingly.
    2. Verify that winning and losing scenarios update the player's bankroll correctly.
2. **Dealer Class:**
    1. Test the dealer's play logic to ensure it follows the specified rules.

# Implementation

•Implementing a complete Blackjack game in Python involves translating the design into actual code. Below is a simplified implementation of a console-based Blackjack game.

•Note that this example doesn't include all possible features and might require additional refinement and expansion based on your specific requirements


# Conclusion

•Players can enter their name and initial bankroll to start the game. The game uses a deck of cards with appropriate ranks and suits.
•Cards are shuffled and dealt to both the player and the dealer. Players can place bets, choose to hit or stand during their turn, and play multiple rounds. The game calculates scores for both the player and the dealer based on standard Blackjack rules.
•It handles scenarios such as busting (going over 21) and soft aces.
•The dealer follows a set of rules for playing (e.g., hitting until a score of 17 is reached). The game determines the winner based on the comparison of player and dealer scores.
•Player bets are handled accordingly, and the bankroll is updated.
•Players have the option to play additional rounds or exit the game

# Future Scope

•Implement a graphical interface using libraries such as Pygame, Tkinter, or PyQt.
•Enhance user interaction with visually appealing cards, buttons, and animations. Enable multiplayer gameplay to allow multiple players to participate in the same game.
•Implement networking features for online multiplayer or local multiplayer on the same device.
•Introduce additional betting options such as splitting pairs or doubling down to increase strategic depth. Implement different Blackjack variations with unique rules (e.g., European Blackjack, Spanish 21).
•Allow players to choose their preferred rule set or variation.
•Integrate a strategy advisor that provides suggestions to players based on optimal Blackjack strategies.
•Offer hints on whether to hit, stand, split, or double down for a given hand.
•Incorporate a system of achievements or rewards to motivate players and add a sense of accomplishment.
•Reward players for reaching milestones, executing strategic moves, or achieving high scores.
•Allow players to customize their gaming experience by choosing themes, card designs, or table backgrounds.
•Implement personalized profiles with statistics and achievements.
•Integrate sound effects and animations to make the gameplay more immersive and engaging.
•Add audio cues for card shuffling, dealing, and winning/losing scenarios.

# Output


File Edit Shell Debug Options Window Help
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct 2 2023, 13:03:39) [MSC v.1935
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\
Cards Player Has: [('Jack', 'Spades'), ('2', 'Diamonds')]
Score Of The Player: 12
What do you want? ["play" to request another card, "stop" to stop]:

# Reference


[1] Teck-Hua Ho and Kay-Yut Chen "**New Product Blockbusters: The Magic and Science of Prediction Markets" 2007**

[2]Bjoern Bartels, Ulrich Ermel, Peter Sandborn, Michael G. Pecht "**Strategies to the Prediction, Mitigation and Management of Product Obsolescence" 2005**

**[3]** Michiel van Wezel and Rob Potharst "Improved customer choice predictions using ensemble methods" 2007

[4] Wallace R. Blischke, D. N. Prabhakar Murthy "**Reliability: Modeling, Prediction, and Optimization" 2008**

**[5]** Edward C. Malthouse, Robert C. Blattberg **"Can we predict customer lifetime value" 2005**


Thank
you !