HARVEST INTERNATIONAL SCHOOL
BANGALORE

COMPUTER SCIENCE PROJECT REPORT

Done By: Amrutha Babu

Grade: 12

Year: 2023-2024

CERTIFICATE

This is to certify that Amrutha Babu student of class XII has successfully completed the project under the guidance of Mrs. Pravitha during the year 2023-24.

Teacher In Charge

Principal

ACKNOWLEDGEMENT

I would like to thank my **Principal** and the management for giving me the opportunity to do this project.

I would like to thank my Computer Teacher **Mrs. Pravitha** for constantly guiding and supporting me to do this project.

I would also like to thank my **Parents and Teammates** for their support and motivation.

We as a group consisting of Amrutha Babu, Arpita Sinha, Moksha Singh, Nihara Zaria of Grade 12 are happy to do this project which has motivated us to be creative and innovative.

Index

About Python

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed. It supports multiple programming paradigms, including structured, object oriented and functional programming.

Python is a popular programming language. it was created by Guido Van Rossum and released in 1991.

Uses of Python

- Web development
- Software development
- Gaming
- Data Visualization

Python Features: -

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

System Specifications

- Windows Operating System:
- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM.
- 5 GB free disk space.
- SVGA or higher-resolution monitors (XGA recommended)
- Mouse or other pointing device
- Python – IDLE (3.11 64 bit)

About the Project

The Game

Blackjack, also known as 21, is a popular and classic card game. The game is simple yet strategic, involving elements of skill, luck, and decision-making. The primary objective is to beat the dealer by having a hand value closer to 21 than theirs without exceeding it.

Objective:
Players aim to build a hand with a total value as close to 21 as possible without going over. Each card has a specific value: numbered cards are worth their face value, face cards (kings, queens, and jacks) are worth 10, and aces can be worth 1 or 11, depending on the player's choice.

Gameplay:
Dealing Cards: The game begins with each player, including the dealer, receiving two cards. In most variations, players are dealt face-up cards, while the dealer has one face-up (the "upcard") and one face-down (the "hole card").
Player's Turn: Players decide how to play their hand based on the total value of their cards. Options include "hit" (take another card), "stand" (keep the current hand), "double down" (double the initial bet and take one more card), and "split" (if dealt two cards of the same rank, split them into two separate hands).
Dealer's Turn: After all players have completed their turn, the dealer reveals their hole card. The dealer must follow specific rules, typically hitting until their hand totals 17 or higher and then standing.

Winning and Losing:
If a player's hand exceeds 21, they "bust" and lose the round.
If a player's hand is closer to 21 than the dealer's without busting, they win.
If the dealer busts, all remaining players win.
Ties result in a "push," and players retain their bets.

Card Counting:

Some players employ card counting strategies to track the ratio of high to low-value cards in the deck. While card counting is legal, casinos may take countermeasures against players who use such tactics.

The Code

In this menu driven program, we have used the concept of binary files and stacks. The three options are to start the game, display scores or exit the game. ‘random’ and ‘pickle’ module is imported for playing the game and storing the scores respectively.

First, the deck is initialized and the value of the hand is calculated. The cards are displayed on screen randomly. Before beginning the game, a player code must be provided so that the scores can be associated with a particular player.

‘blackjack_game()’ is the main game loop that encompasses the entire Blackjack game logic. It starts by loading scores, then allows the player to choose from options: start a new game, show scores, or quit. The game proceeds with player and dealer hands, and the player makes choices to hit or stand. The game outcome is determined, and scores are updated accordingly.

The game uses a simple ASCII art representation for card display and the scores are saved and loaded using the pickle module, which serializes Python objects to binary. The game loop continues until the player decides to quit.

SOURCE CODE

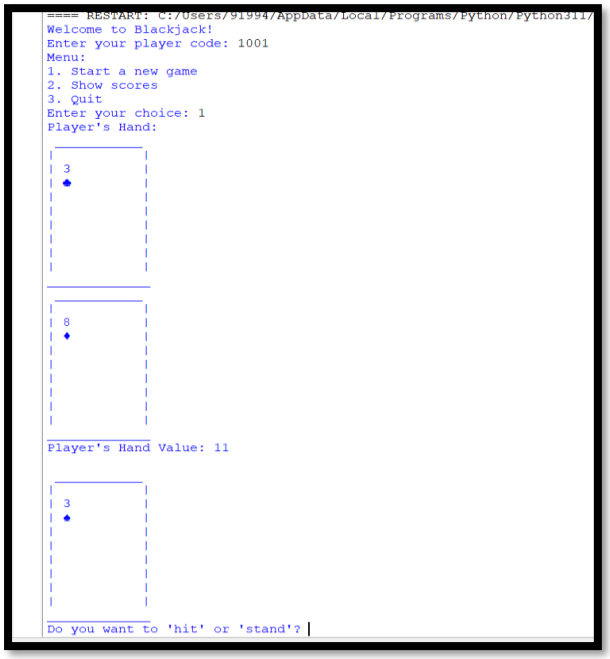
```
import random
import pickle

# Initialize the deck
def initialize_deck():
    ranks = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']
    suits = ['♥', '♦', '♣', '♠']
    deck = [{'rank': rank, 'suit': suit} for rank in ranks for suit in suits]
    random.shuffle(deck)
    return deck

# Calculate the value of a hand
def calculate_hand_value(hand):
    values = {'2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9, '10': 10, 'J': 10, 'Q':
10, 'K': 10, 'A': 11}
    value = sum(values[card['rank']] for card in hand)
    aces = [card for card in hand if card['rank'] == 'A']
    while value > 21 and aces:
        value -= 10
        aces.pop()
    return value


# Display card outlines
def display_card_outline(card):
    print(f" _____________")
    print(f"|             |")
    print(f"| {card['rank']:<2}          |")
    print(f"| {card['suit'][0]:<2}          |")
    print(f"|             |")
    print(f"|             |")
    print(f"|             |")
    print(f"|             |")
```

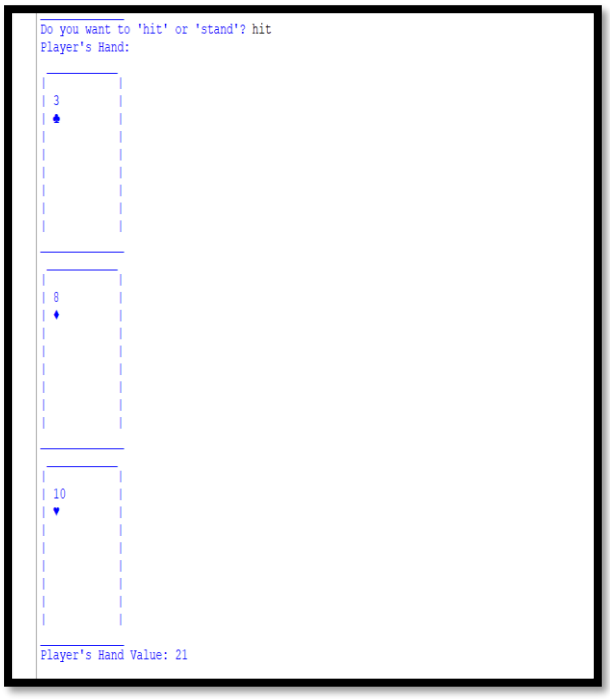
```
    print(f"|         |")
    print(f"|         |")
    print(f"_____________")

# Display the player's hand
def display_hand(player_hand, title):
    print(f"{title}'s Hand:")
    for card in player_hand:
        display_card_outline(card)
    print(f"{title}'s Hand Value: {calculate_hand_value(player_hand)}\n")

# Save scores to a binary file
def save_scores(scores):
    with open('scores.dat', 'wb') as file:
        pickle.dump(scores, file)

# Load scores from a binary file
def load_scores():
    try:
        with open('scores.dat', 'rb') as file:
            scores = pickle.load(file)
    except FileNotFoundError:
        scores = {}
    return scores

# Main game loop
def blackjack_game():
    scores = load_scores()
    while True:
        print("Welcome to Blackjack!")
        player_hand = []
        dealer_hand = []
        deck = initialize_deck()
        for _ in range(2):
            player_hand.append(deck.pop())
            dealer_hand.append(deck.pop())
```

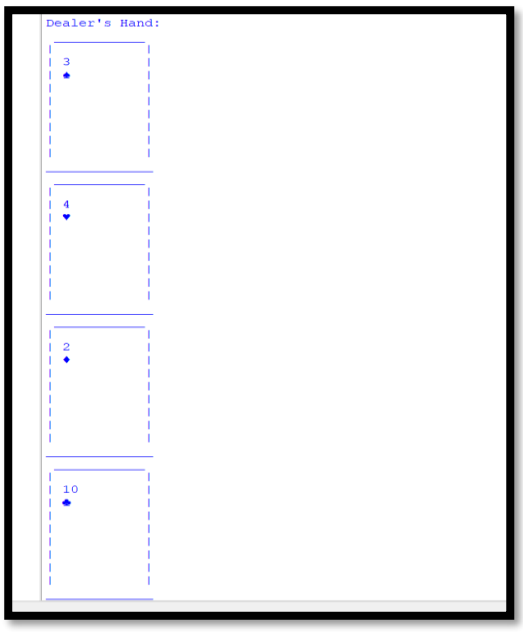
```
        player_code = input("Enter your player code: ")

        print("Menu:")
        print("1. Start a new game")
        print("2. Show scores")
        print("3. Quit")
        choice = input("Enter your choice: ")

        if choice == '1':
            player_value = calculate_hand_value(player_hand)
            dealer_value = calculate_hand_value(dealer_hand)

            while player_value < 21:
                display_hand(player_hand, "Player")
                display_card_outline(dealer_hand[0])
                action = input("Do you want to 'hit' or 'stand'? ").lower()
                if action == 'hit':
                    player_hand.append(deck.pop())
                    player_value = calculate_hand_value(player_hand)
                elif action == 'stand':
                    break

            while dealer_value < 17:
                dealer_hand.append(deck.pop())
                dealer_value = calculate_hand_value(dealer_hand)

            display_hand(player_hand, "Player")
            display_hand(dealer_hand, "Dealer")

            if player_value > 21 or (dealer_value <= 21 and dealer_value >=
player_value):
                print("Dealer wins!")
                if player_code in scores:
                    scores[player_code] -= 1
                else:
                    scores[player_code] = -1
            elif dealer_value > 21 or player_value > dealer_value:
```

```
                print("You win!")
                if player_code in scores:
                    scores[player_code] += 1
                else:
                    scores[player_code] = 1
            else:
                print("It's a tie!")
        elif choice == '2':
            print("Scores:")
            for player, score in scores.items():
                print(f"Player {player}: {score}")
        elif choice == '3':
            save_scores(scores)
            print("Thanks for playing! See you next time.")
            break

if __name__ == "__main__":
    blackjack_game()
```

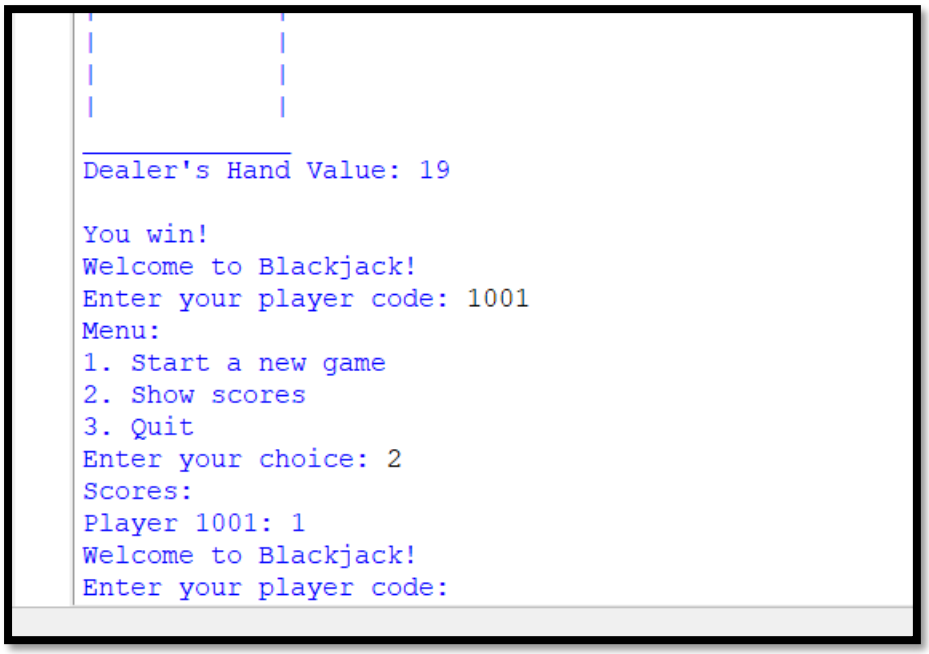
Playing the game



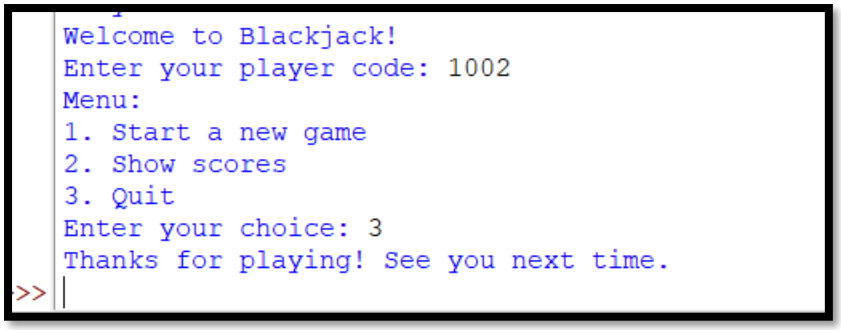




Showing Scores:



Quitting:



BIBLIOGRAPHY

Computer Science With Python – Textbook for Class XII (Sumita Arora)

https://www.britannica.com/topic/blackjack-card-game