T1A3 - Terminal Blackjack

Marianna Ilisevic

Usage and Features

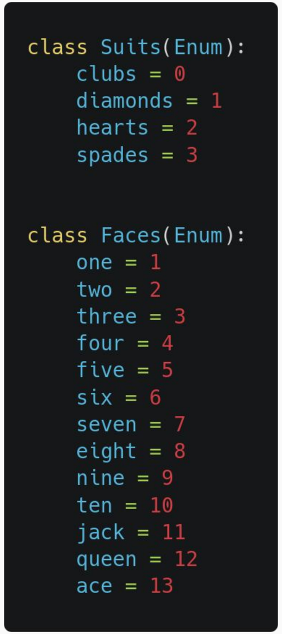
Interactive Mode

- Command line options for configuring game
- Interactive text-based terminal interface for gameplay
- Game will end once round limit reached or player/dealer runs out of funds
- Option to print summary of all previous game rounds on exit

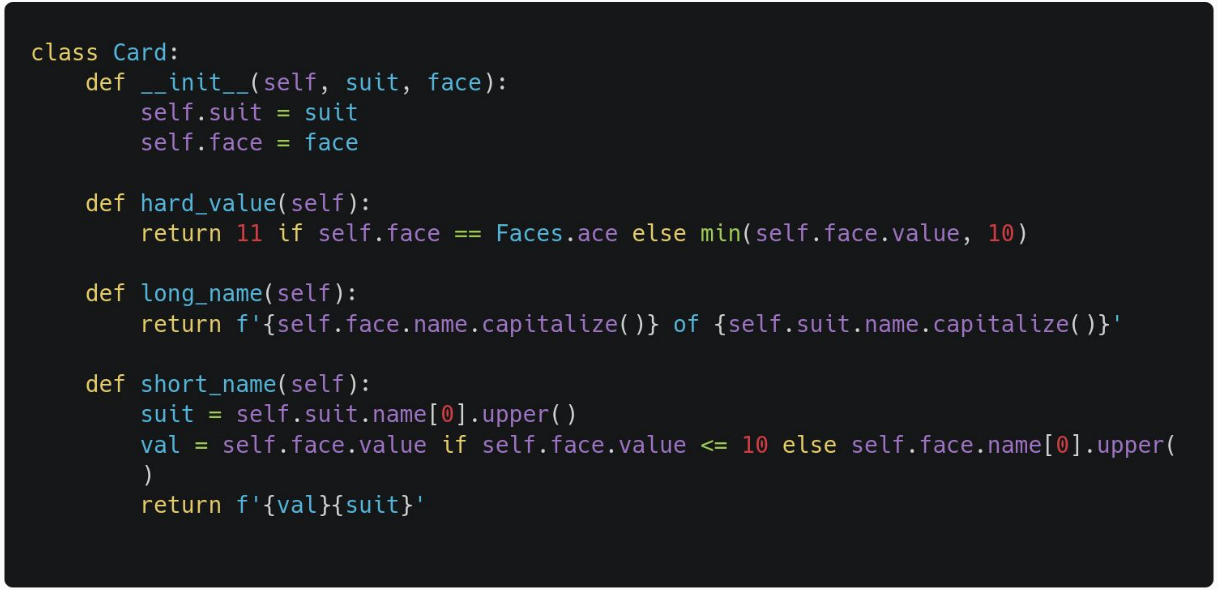
Simulation Mode

- Command line options for configuring game
- Will play specified number of rounds or until computer or dealer runs out of funds.
- Summary of rounds printed at end of game

Representing Playing Cards

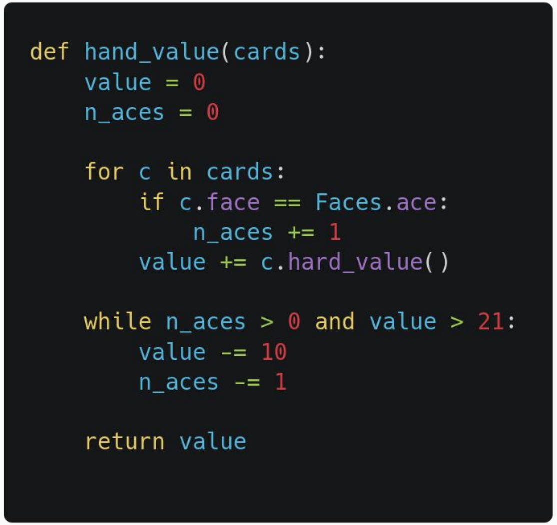


```python
class Suits(Enum):
    clubs = 0
    diamonds = 1
    hearts = 2
    spades = 3


class Faces(Enum):
    one = 1
    two = 2
    three = 3
    four = 4
    five = 5
    six = 6
    seven = 7
    eight = 8
    nine = 9
    ten = 10
    jack = 11
    queen = 12
    ace = 13
```



```python
class Card:
    def __init__(self, suit, face):
        self.suit = suit
        self.face = face

    def hard_value(self):
        return 11 if self.face == Faces.ace else min(self.face.value, 10)

    def long_name(self):
        return f'{self.face.name.capitalize()} of {self.suit.name.capitalize()}'

    def short_name(self):
        suit = self.suit.name[0].upper()
        val = self.face.value if self.face.value <= 10 else self.face.name[0].upper(
        )
        return f'{val}{suit}'
```

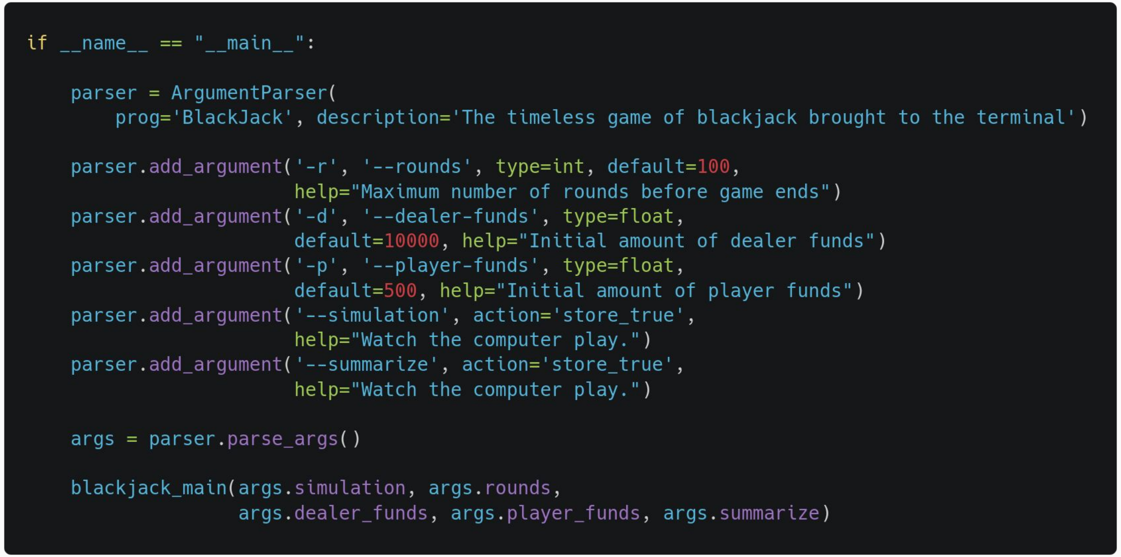
Calculating the Value of a Hand



```python
def hand_value(cards):
    value = 0
    n_aces = 0

    for c in cards:
        if c.face == Faces.ace:
            n_aces += 1
        value += c.hard_value()

    while n_aces > 0 and value > 21:
        value -= 10
        n_aces -= 1

    return value
```

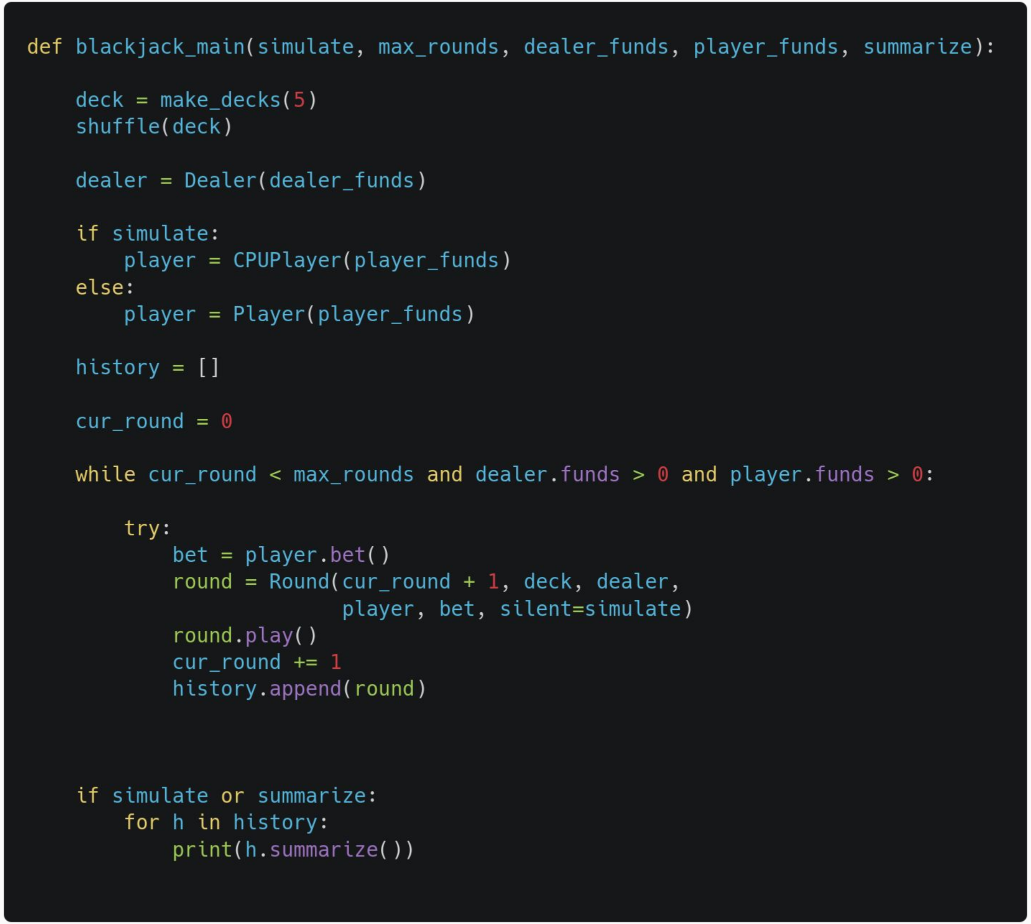
- Number cards are worth their face value
- Face cards except for aces (J,Q,K) are worth 10
- Aces are worth 11 unless it would cause a hand to go over 21, then they are worth 1

A Top Level View



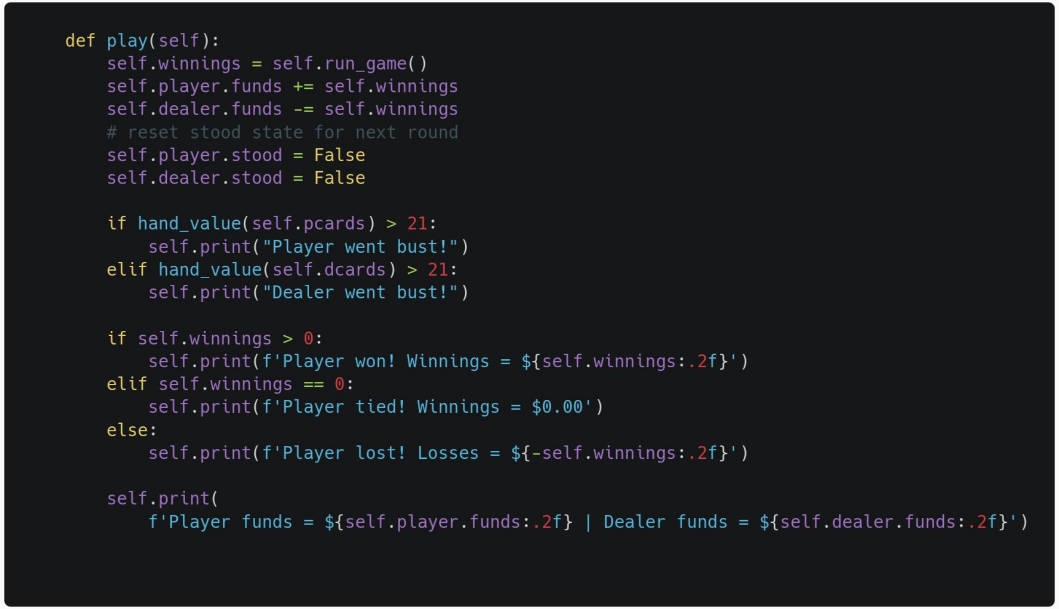
```
if __name__ == "__main__":

    parser = ArgumentParser(
        prog='BlackJack', description='The timeless game of blackjack brought to the terminal')

    parser.add_argument('-r', '--rounds', type=int, default=100,
                        help="Maximum number of rounds before game ends")
    parser.add_argument('-d', '--dealer-funds', type=float,
                        default=10000, help="Initial amount of dealer funds")
    parser.add_argument('-p', '--player-funds', type=float,
                        default=500, help="Initial amount of player funds")
    parser.add_argument('--simulation', action='store_true',
                        help="Watch the computer play.")
    parser.add_argument('--summarize', action='store_true',
                        help="Watch the computer play.")

    args = parser.parse_args()

    blackjack_main(args.simulation, args.rounds,
                   args.dealer_funds, args.player_funds, args.summarize)
```

Handling Multiple Modes & Rounds



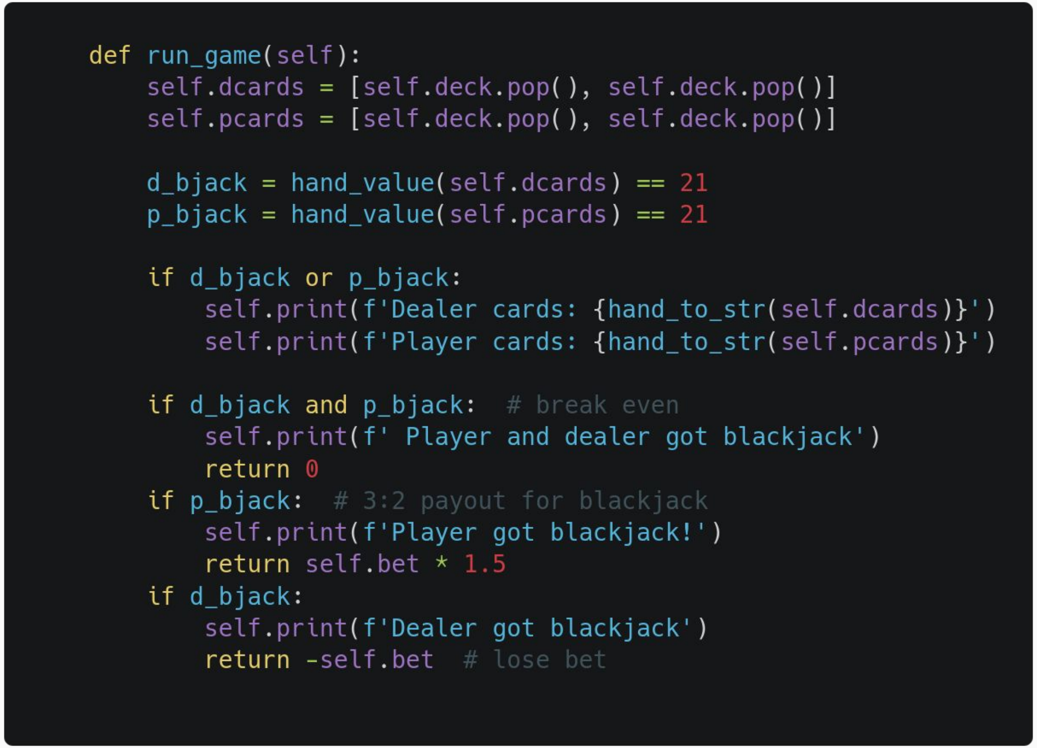
```
def blackjack_main(simulate, max_rounds, dealer_funds, player_funds, summarize):

    deck = make_decks(5)
    shuffle(deck)

    dealer = Dealer(dealer_funds)

    if simulate:
        player = CPUPlayer(player_funds)
    else:
        player = Player(player_funds)

    history = []

    cur_round = 0

    while cur_round < max_rounds and dealer.funds > 0 and player.funds > 0:

        try:
            bet = player.bet()
            round = Round(cur_round + 1, deck, dealer,
                          player, bet, silent=simulate)
            round.play()
            cur_round += 1
            history.append(round)



    if simulate or summarize:
        for h in history:
            print(h.summarize())
```

Starting and Ending a Round



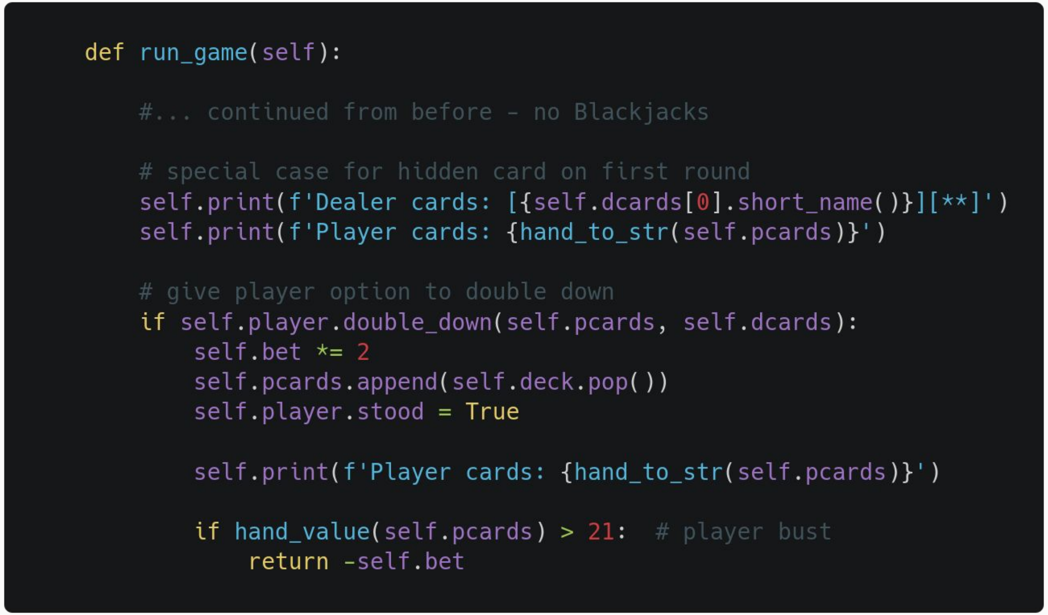
```python
def play(self):
    self.winnings = self.run_game()
    self.player.funds += self.winnings
    self.dealer.funds -= self.winnings
    # reset stood state for next round
    self.player.stood = False
    self.dealer.stood = False

    if hand_value(self.pcards) > 21:
        self.print("Player went bust!")
    elif hand_value(self.dcards) > 21:
        self.print("Dealer went bust!")

    if self.winnings > 0:
        self.print(f'Player won! Winnings = ${self.winnings:.2f}')
    elif self.winnings == 0:
        self.print(f'Player tied! Winnings = $0.00')
    else:
        self.print(f'Player lost! Losses = ${-self.winnings:.2f}')

    self.print(
        f'Player funds = ${self.player.funds:.2f} | Dealer funds = ${self.dealer.funds:.2f}')
```

Handling Blackjacks



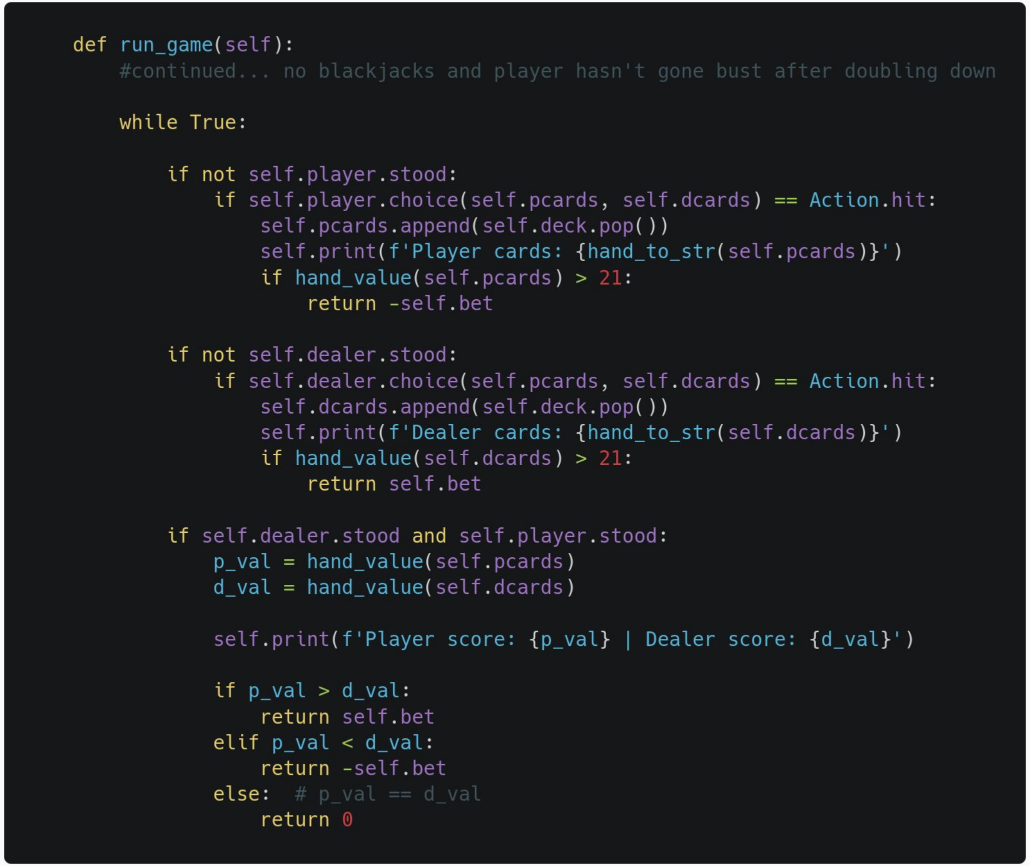
```python
def run_game(self):
    self.dcards = [self.deck.pop(), self.deck.pop()]
    self.pcards = [self.deck.pop(), self.deck.pop()]

    d_bjack = hand_value(self.dcards) == 21
    p_bjack = hand_value(self.pcards) == 21

    if d_bjack or p_bjack:
        self.print(f'Dealer cards: {hand_to_str(self.dcards)}')
        self.print(f'Player cards: {hand_to_str(self.pcards)}')

    if d_bjack and p_bjack:  # break even
        self.print(f' Player and dealer got blackjack')
        return 0
    if p_bjack:  # 3:2 payout for blackjack
        self.print(f'Player got blackjack!')
        return self.bet * 1.5
    if d_bjack:
        self.print(f'Dealer got blackjack')
        return -self.bet  # lose bet
```

Doubling Down



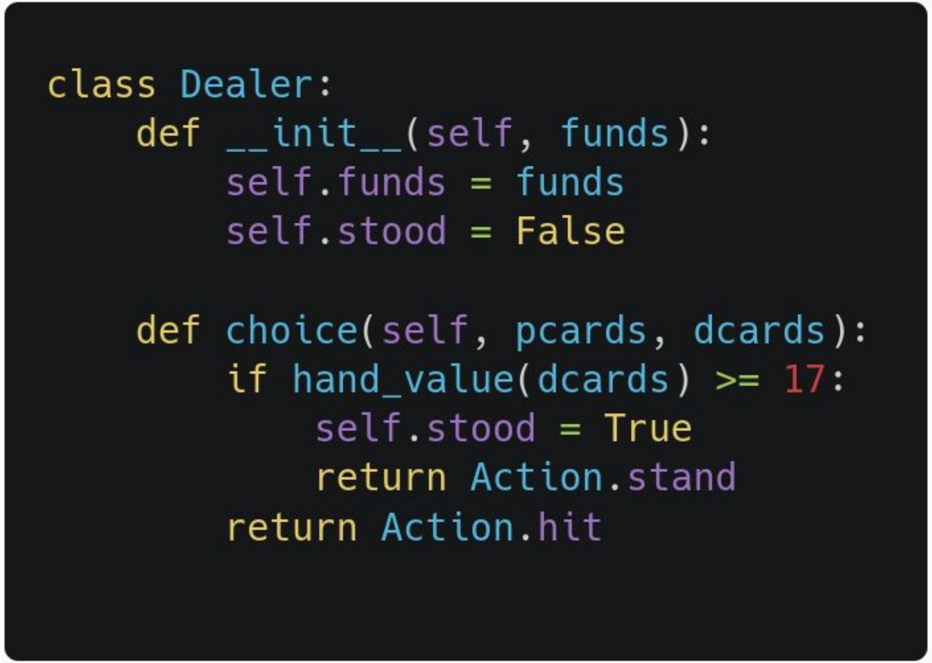
```python
def run_game(self):

    #... continued from before - no Blackjacks

    # special case for hidden card on first round
    self.print(f'Dealer cards: [{self.dcards[0].short_name()}][**]')
    self.print(f'Player cards: {hand_to_str(self.pcards)}')

    # give player option to double down
    if self.player.double_down(self.pcards, self.dcards):
        self.bet *= 2
        self.pcards.append(self.deck.pop())
        self.player.stood = True

        self.print(f'Player cards: {hand_to_str(self.pcards)}')

        if hand_value(self.pcards) > 21:  # player bust
            return -self.bet
```

Handling Turns



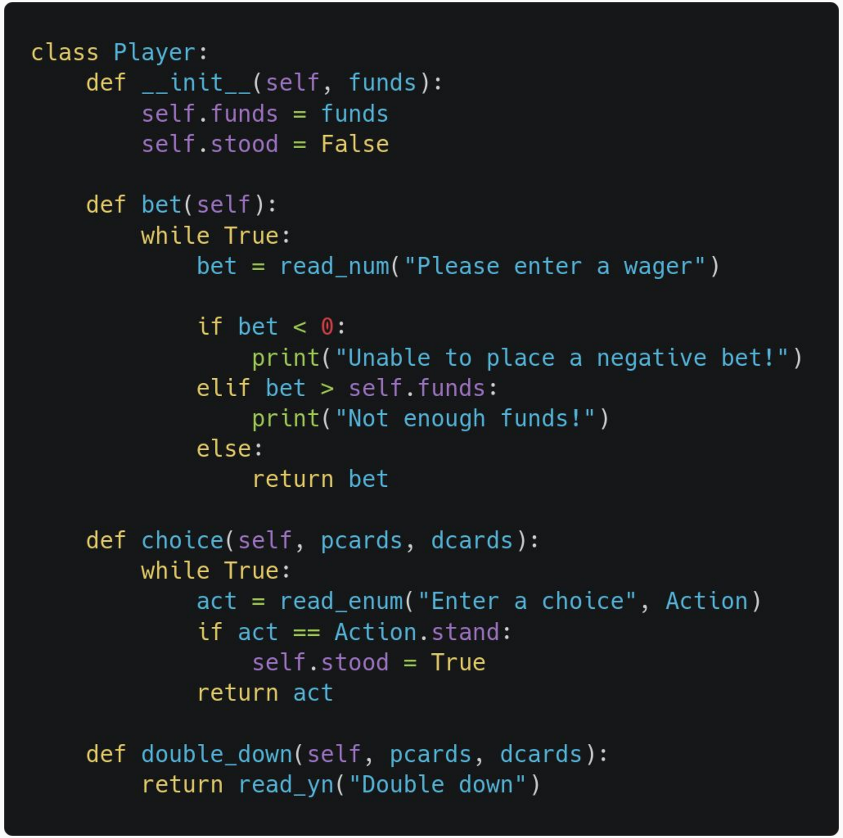
```python
def run_game(self):
    #continued... no blackjacks and player hasn't gone bust after doubling down

    while True:

        if not self.player.stood:
            if self.player.choice(self.pcards, self.dcards) == Action.hit:
                self.pcards.append(self.deck.pop())
                self.print(f'Player cards: {hand_to_str(self.pcards)}')
                if hand_value(self.pcards) > 21:
                    return -self.bet

        if not self.dealer.stood:
            if self.dealer.choice(self.pcards, self.dcards) == Action.hit:
                self.dcards.append(self.deck.pop())
                self.print(f'Dealer cards: {hand_to_str(self.dcards)}')
                if hand_value(self.dcards) > 21:
                    return self.bet

        if self.dealer.stood and self.player.stood:
            p_val = hand_value(self.pcards)
            d_val = hand_value(self.dcards)

            self.print(f'Player score: {p_val} | Dealer score: {d_val}')

            if p_val > d_val:
                return self.bet
            elif p_val < d_val:
                return -self.bet
            else:  # p_val == d_val
                return 0
```

The Dealer



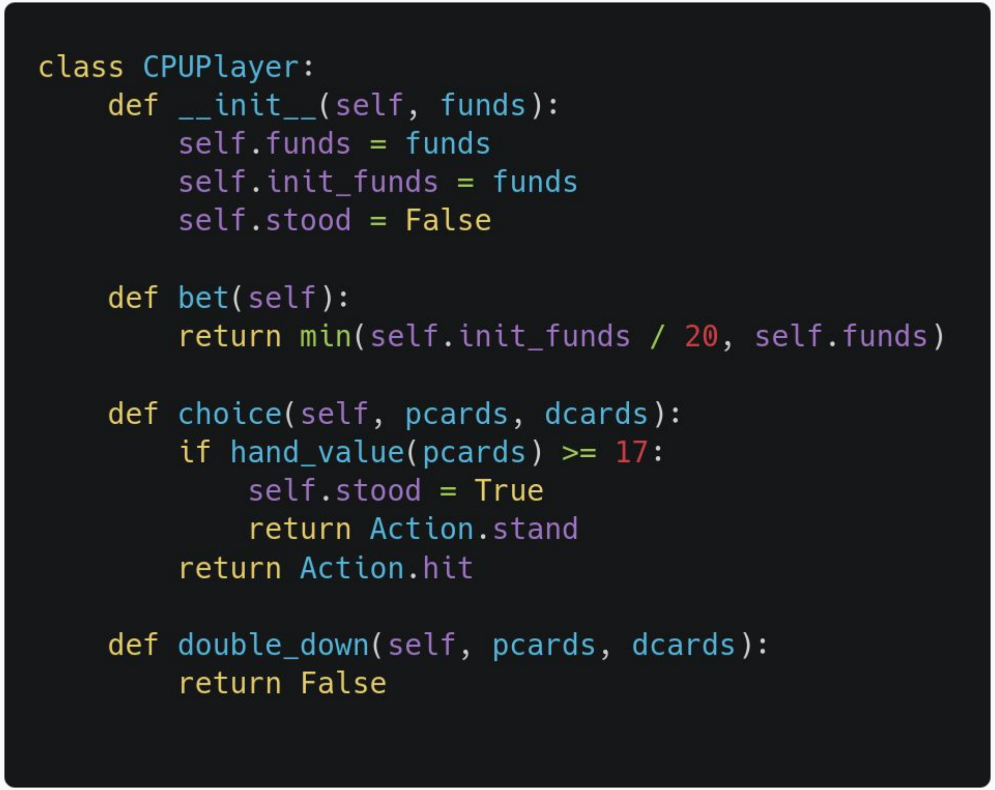
```python
class Dealer:
    def __init__(self, funds):
        self.funds = funds
        self.stood = False

    def choice(self, pcards, dcards):
        if hand_value(dcards) >= 17:
            self.stood = True
            return Action.stand
        return Action.hit
```

The Human Player



```
class Player:
    def __init__(self, funds):
        self.funds = funds
        self.stood = False

    def bet(self):
        while True:
            bet = read_num("Please enter a wager")

            if bet < 0:
                print("Unable to place a negative bet!")
            elif bet > self.funds:
                print("Not enough funds!")
            else:
                return bet

    def choice(self, pcards, dcards):
        while True:
            act = read_enum("Enter a choice", Action)
            if act == Action.stand:
                self.stood = True
            return act

    def double_down(self, pcards, dcards):
        return read_yn("Double down")
```

The Not so Human Player



```python
class CPUPlayer:
    def __init__(self, funds):
        self.funds = funds
        self.init_funds = funds
        self.stood = False

    def bet(self):
        return min(self.init_funds / 20, self.funds)

    def choice(self, pcards, dcards):
        if hand_value(pcards) >= 17:
            self.stood = True
            return Action.stand
        return Action.hit

    def double_down(self, pcards, dcards):
        return False
```

Development Process

- Researching Blackjack - Game was more complicated than I remembered
- Challenges
  - Handling the multiple phases of the game - not all turns are the same
  - Modeling the game in terms of objects
  - Ensuring the rules of the game are obeyed
  - Sanitising user input
- Future challenges
  - Implementing splitting - breaks the 1:1 connection between players and a single hand/turn
  - Implementing an optimal strategy for the computer player - many decision tables available

Development Process (cont.)

- Ethical issues
  - Promotion of gambling even if only for numbers on a terminal
- Fun parts
  - Watching the computer play itself - uncovered many bugs that we not triggered when testing the game by playing it.

Thank you for listening!